MIT Project Oxygen 2001
Research Papers

200 Technology Square
Cambridge, MA 02139
USA
http://oxygen.lcs.mit.edu

Dedicated to Michael L. Dertouzous and his
unfinished revolution.

Preface

MIT Project Oxygen is in its second full year of operation. A large number of faculty from the Laboratory for Computer Science and the Artificial Intelligence Laboratory have brought their work together to realize the dream of pervasive human centered computing. With our partners in the Oxygen Alliance we are inventing a new computational environment that we hope will revolutionize the way in which humans interact with computation.

The impetus for the Oxygen effort came from a number of different perspectives:

- that we should be able to do more by doing less,
- that speech and vision interfaces are the key to ease of use,
- that the computer should be brought out into the human world rather than vice-versa,
- and that computation and communication will continue its exponential drop in price.

With these assumptions in mind we constructed a framework for Oxygen around three sorts of artifacts (morphable hand-held units: the Handy-21; environmental intelligent spaces: the Enviro-21; and new layers of network capabilities; the Network-21), around the hardware and software infrastructures for these, and around four user technologies (speech and vision, automation, collaboration, and individual knowledge access). We did not expect to devote precisely equal resources across all nine of these subdivisions, nor within each of the categories. However, we have worked in all of these areas, and will continue to work in them all. As we have proceeded, the boundaries between these areas of work have become fuzzier and fuzzier as we are building bigger and more integrated prototype systems.

This book captures the first stages of work across all the areas of Oxygen. These are research papers recently published, or about to be published by faculty and students at LCS and AI who are working on Oxygen. The papers naturally fit into six rather than nine sections:

- **Handy 21:** technologies for the current generations of Handy's which are based on commercial off the shelf technologies with some custom glue, and low-power high-performance computation technologies for the future custom built generations.

- **Network 21:** new technologies for intentional naming, organization of ad-hoc networks, location tracking, vertical handoff, and network security.
- **Systems:** both hardware and software technologies to support the Oxygen vision, with a good number of technologies that straddle the hardware/software boundary.
- **Perceptual interfaces:** speech based systems, vision based trackers, microphone arrays, visual disambiguation of speech signals and sketching interfaces.
- **Knowledge access:** natural language and semantic bases for access to the vast amounts of knowledge that are available on the network.
- **Collaboration:** tools for helping people to be more productive in meetings or giving presentations.

While written papers give the technical details of the work, it is often hard to convey in them the way in which the work changes the way it feels to be in a human centric pervasive computing environment. For that one needs to experience it. Some of our technologies are already in our partners' hands. Others can be seen in movies on the Oxygen web site at http://oxygen.lcs.mit.edu/.

Onward, with the revolution!

Anant Agarwal
Rodney Brooks
Victor Zue
December 14, 2001

This work was funded by The Acer Group, Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under the MIT Project Oxygen partnership, and by DARPA through the Office of Naval Research under contract number N66001- 99-2-891702. Some of the work was partially or wholly funded under previous arrangements.

Nick Matsakis produced the Latex style files for this publication, and debugged the most arcane Latex problems. Annika Pfluger did the lion's share of the work. She wrestled with many people's Latex source files and got all the papers into a uniform format. She deserves the major credit for putting this book of papers together.

Table of Contents

Preface

Handy 21

Network 21

Systems

Perceptual Interfaces

Knowledge Access

Collaboration

Author Index

Handy 21

Proceedings of Hotchips 13, August 2001.

The Raw Processor: A Composeable 32-Bit Fabric for Embedded and General Purpose Computing

Michael Taylor, Jason Kim, Jason Miller, Fae Ghodrat,
Ben Greenwald, Paul Johnson, Walter Lee, Albert Ma,
Nathan Shnidman, David Wentzlaff, Matt Frank,
Saman Amarasinghe and Anant Agarwal

MIT Laboratory for Computer Science
{`mtaylor, jkim, jasonm, fghodrat, beng, prj, walt, ama, naters, wentzlaff, mfrank, saman, agarwal`}`@lcs.mit.edu`

The Raw project is attempting to create a scalable processor architecture that is suitable for both general purpose and embedded computations. Current general purpose processors differ from embedded devices in that they provide large amounts of hardware support to discover and manipulate instruction-level parallelism and unstructured memory accesses. Because the parallelism in embedded computations is much more predictable, embedded devices such as DSPs do not offer a rich set of mechanisms, rather they devote their area to computational resources such as pipelined floating point, thereby achieving significantly better area and energy efficiency. However, their best performance is achieved for regular data access patterns such as streams, and they often require assembly code manipulation. Embedded FPGAs and ASICs go one step further, and can offer even better results for many classes of computations, but require a hardware design step in mapping their applications into silicon.

Raw will support many classes of computations that traditionally have run on microprocessors, DSPs, FPGAs and ASICs. Raw implements a simple, highly parallel, tiled architecture, and exposes its interconnect, I/O, memory and computational elements to the compiler [5]. This exposure allows the software system to allocate resources and coordinate data flow within the chip in an application-specific manner. Furthermore, the tiled, replicated architecture of Raw allows it to scale with increasing silicon densities.

As depicted in Figure 1, the Raw processor is a single chip containing 16 identical processor-sized tiles connected in a 4-by-4 mesh configuration by four nearest neighbor point-to-point pipelined high-speed networks (two static, two dynamic). Implemented in the .15 micron IBM SA-27E ASIC process, the design occupies an 18.2x18.2mm die, has 1080 HSTL signal I/Os, consumes 45W[1], and runs at a target frequency of 250 MHz. Because tiles are only connected to their nearest neighbors, the longest wire on the chip runs only the length of a single tile.

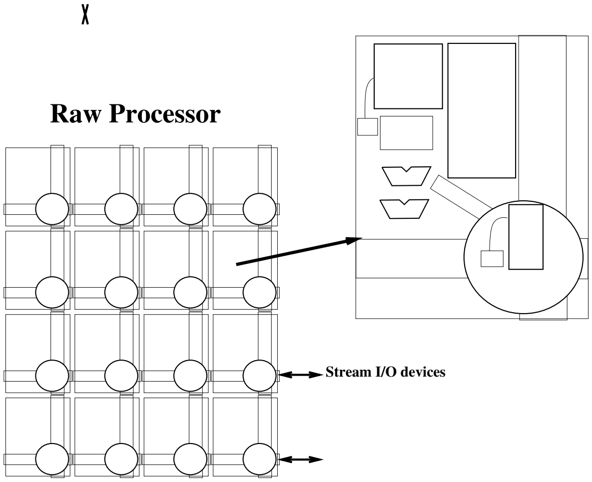


Fig. 1. Raw processor composition. A typical Raw system might include a Raw processor coupled with off-chip RDRAM and stream-IO devices.

Each tile contains a general-purpose processor, which is connected to its neighbors by a static router and a dynamic router. The processor is an eight stage single-issue MIPS-style pipeline. It has a four stage pipelined FPU, a 32 KByte two-way associative SRAM data cache and 32 KBytes of instruction SRAM that is virtualized via a binary rewriting system. When the data access patterns are known at compile time, the software implements software data caching for predictable memory access by using a portion of the instruction SRAM memory.

The static router controls two independent 32-bit pipelined channels in each direction (North, East, South, West, and Processor). It

---

[1] To reduce project risk, we have not focussed on low-power design in our first experimental Raw prototype. However, its tiled, pipelined nature allows application-specific orchestration of power, and we plan to undertake a low-power design in a follow-on project.

sequences a 64-bit instruction that simultaneously specifies an operation (branch, no-op, or move) and 12 routes between these channels. The static router's local SRAM contains 8K of these instructions, and is also virtualized in software. Like the instruction SRAM, the software can also use a portion of the static router SRAM to store data with predictable access patterns.

In order to route a value between two tiles on this network, one inserts instructions on each intermediate node specifying the appropriate route. The static router allows single word messages to be sent between tiles with a guaranteed relative ordering. The purpose of this network is to connect the Raw tiles in a manner that can exploit both ILP (instruction level parallelism) and streaming data parallelism. The parallelizing Raw compiler, RAWCC [4, 3, 2, 1], uses these routers as an operand network between the ALUs of the processors to parallelize SpecFP 95 and multimedia applications.

The dynamic router uses a dimensioned-ordered wormhole routing protocol to control two independent 32-bit pipelined channels in each direction (North, East, South, West, and Processor). These channels allow messages of up to 31 words (plus a header specifying the destination tile number, source tile number, message length, and type) to be sent between tiles and outside of the chip. These messages take one cycle per hop when going straight, and two cycles on turns. The processor dedicates one of these two dynamic channels to external communication: memory (i.e., cache misses), interrupts, and PCI transactions. The other channel is utilized for user-level messaging between tiles. The processor supports OS-level transparent deadlock recovery in the event that the user over-commits the buffer resources of the network.

The pins of the chip are connected directly to the edges of the mesh networks, and run at the speed of the chip. When a message is routed off the side of the chip, it appears on the pins. Because the number of network signals (38 signals x 4 networks x 16 tile-sides x 2 directions = 4864) exceeds the number of pins available, we transparently multiplex the 2 static and 2 dynamic networks on each port down to one physical channel. We also multiplex the middle two channels in the vertical direction, which results in 14 channels, or 112 Gbits of bandwidth in each of the output and input directions.

The Raw chip can be composed into power-of-two dimensioned meshes, for systems of up to 64 chips. The system is virtualized in such a way that this system will appear to the programmer to be exactly like a single Raw chip of 64 times the size, except that certain hops on the networks have an extra cycle (or two, for the shared middle channel) of latency. This allows us a glimpse into the future: we can ascertain the scalability of our architecture as well as run applications that are beyond the capabilities of modern day microprocessors.

Acknowledgments: The Raw project is funded by Darpa, NSF, and the Oxygen Alliance. IBM is supporting the ASIC fabrication of Raw.

References

1. Anant Agarwal, Saman Amarasinghe, Rajeev Barua, Matt Frank, Walter Lee, Vivek Sarkar, and Michael Taylor. The raw compiler project. In *Proceedings of the Second SUIF Compiler Workshop*, Stanford CA, 1997.
2. Rajeev Barua, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Memory Bank Disambiguation using Modulo Unrolling for Raw Machines. In *Proceedings of the ACM/IEEE Fifth Int'l Conference on High Performance Computing(HIPC)*, Dec 1998. Also http://www.cag.lcs.mit.edu/raw/.
3. Rajeev Barua, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Maps: A Compiler-Managed Memory System for Raw Machines. In *Proceedings of the 26th International Symposium on Computer Architecture*, Atlanta, GA, May 1999.
4. Walter Lee, Rajeev Barua, Matthew Frank, Devabhatuni Srikrishna, Jonathan Babb, Vivek Sarkar, and Saman Amarasinghe. Space-Time Scheduling of Instruction-Level Parallelism on a Raw Machine. In *Proceedings of the Eighth ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 46–57, San Jose, CA, October 1998.
5. Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal. Baring It All to Software: Raw Machines. *IEEE Computer*, 30(9):86–93, September 1997. Also available as MIT-LCS-TR-709.

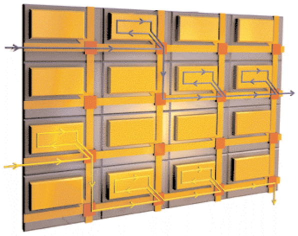
The Raw Processor: A Scalable 32 bit Fabric for General Purpose and Embedded Computing

Presented at Hotchips 13
On August 21, 2001
by

Michael Bedford Taylor

(http://cag.lcs.mit.edu/~ mtaylor)

The Raw Processor

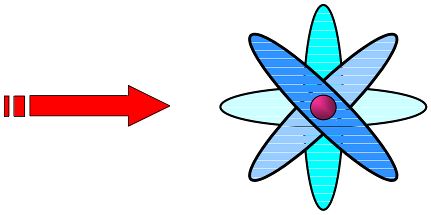


A Scalable 32 bit Fabric
for General Purpose and
Embedded Computing

Michael Taylor , Jason Kim, Jason Miller,
Fae Ghodrat, Ben Greenwald, Paul Johnson,Walter Lee,
Albert Ma, Nathan Shnidman, Volker Strumpen, David Wentzlaff,
Matt Frank, Saman Amarasinghe, and Anant Agarwal

MIT Laboratory for Computer Science
http://cag.lcs.mit.edu/raw

Computer Architecture from 10,000 feet



```
foo(int x)
{ .. }
```

class of computation

convenient physical phenomenon

… **we use abstractions to make this easier**

The Abstraction Layers That Make This Easier

`foo(int x) { .. }`

Computation
Language / API
Compiler / OS — Fortran
ISA — IBM 360 /RISC/ Transmeta/x
Micro Architecture
Layout
Design Style
Design Rules — Mead & Conway
Process
Materials Science
Physics

Abstractions protect us from change
-- but must also change as the world changes

Language / API
Compiler / OS
ISA
Micro Architecture
Layout
Design Style
Design Rules
Process
Materials Science

Changes in physical constraints

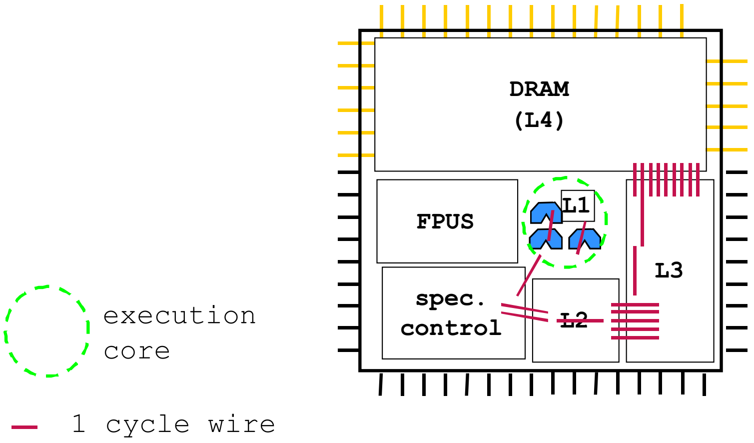
Wire
Delay

More Resources:
Gates
Wires
Pins

Wire delay is crashing through the abstraction layers

Language / API	
Compiler / OS	
ISA	
Micro Architecture	Partitioning(21264)
	Pipelining (P4)
Layout	Timing Driven Placement
Design Rules	Fatter wires
Process	Deeper wires
Materials Science	Cu wires

The future of wire delay handled in micro-architecture



The bottom line

Language / API
Compiler / OS
ISA

Micro Architecture

Floorplan / Layout
Design Rules
Process
Materials Science

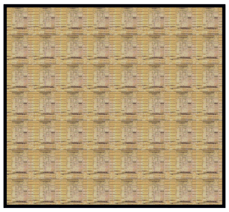
Raw handles this change by exposing the underlying resources (e.g. wires) with a scalable, parallel ISA.

It orchestrates these resources with spatially-aware compilers.

Wire Delay

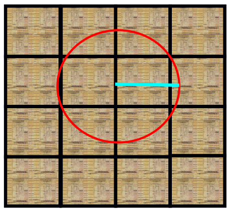
More Resources :
Gates
Wires
Pins

How does Raw expose the resources?



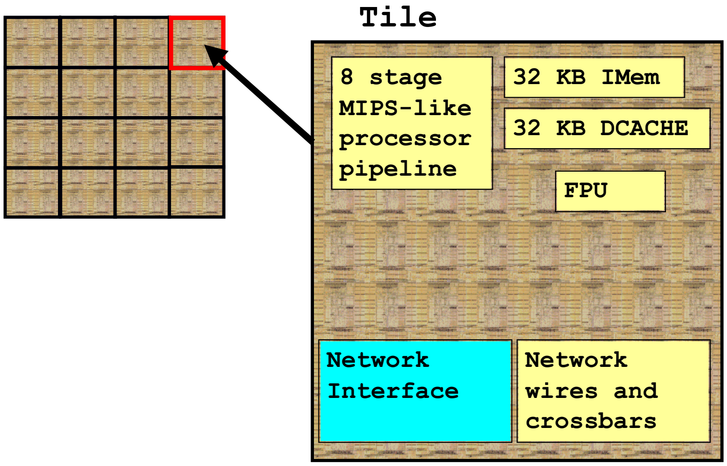
We started with a blank sheet of silicon.

Expose the gates



Cut the silicon up into an array of 16 identical, programmable tiles.

What's inside a tile?



Tile

8 stage MIPS-like processor pipeline

32 KB IMem

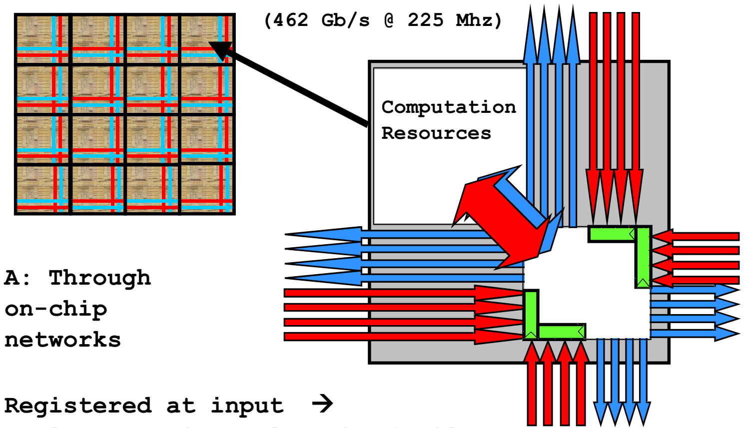
32 KB DCACHE

FPU

Network Interface

Network wires and crossbars

How do we expose the wires?



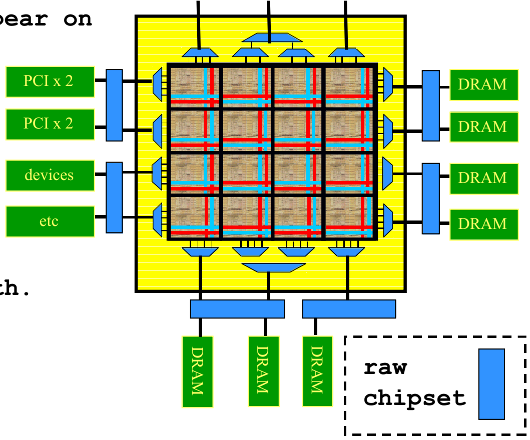
A: Through
on-chip
networks

Registered at input →
longest wire = length of tile

How do we expose the pins?

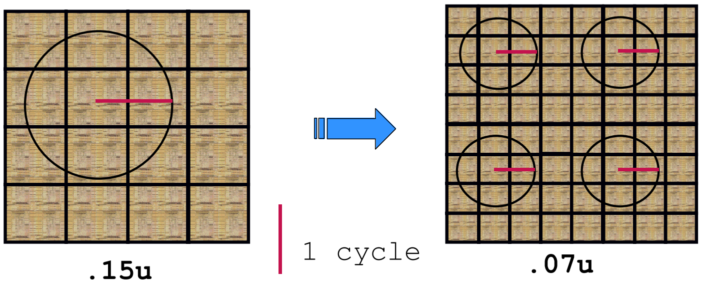
14 7.2 Gb/s channels
(201 Gb/s @ 225 Mhz)

Routes off the edge of the chip appear on the pins.



Gives user direct access to pin bandwidth.

The Raw ISA scales



- longest wire
- Design complexity
- Verification complexity

… are all independent of transistor count.

tiles, network bandwidth and I/O bandwidth scale

Raw is also backwards-compatible.

How well does Raw expose the resources?

Raw Chip (ASIC @225 MHz)

16 OPS/FLOPS per cycle

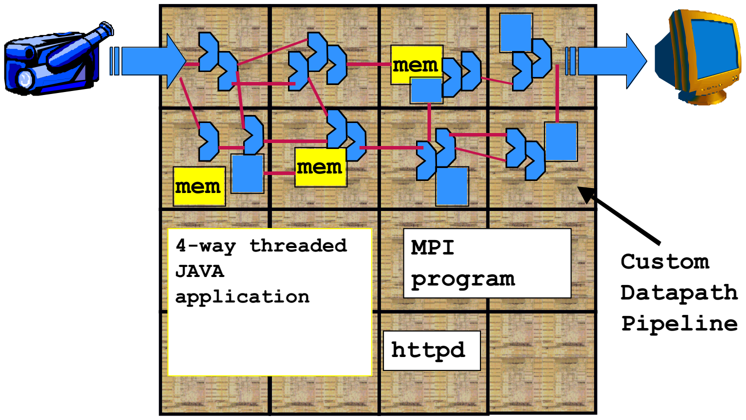
462 Gb/s of on-chip “bisection bandwidth ”

201 Gb/s I/O bandwidth

57 GB/s of on-chip memory bandwidth

... but how are the resources going to be coordinated?

Raw: How we want to use the tiles



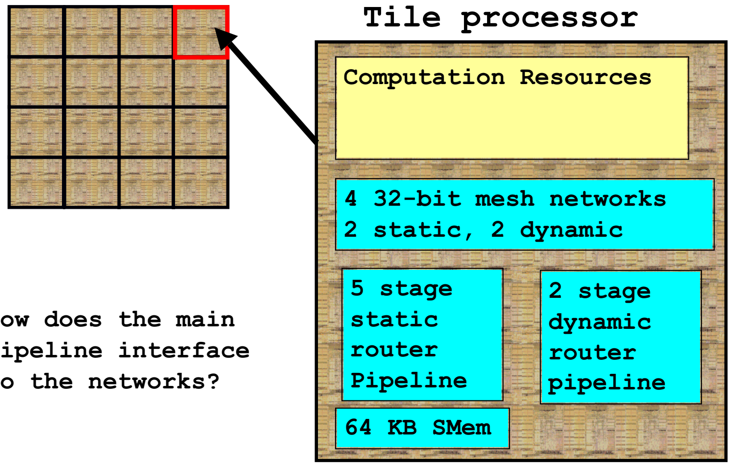
4-way threaded JAVA application

MPI program

httpd

Custom Datapath Pipeline

The Raw Tile network support



Tile processor

Computation Resources

4 32-bit mesh networks
2 static, 2 dynamic

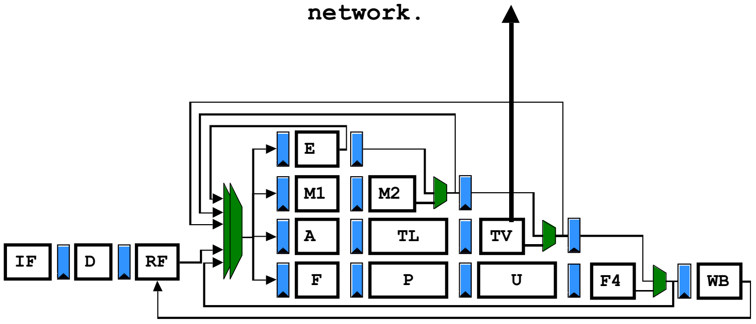
5 stage static router Pipeline

2 stage dynamic router pipeline

64 KB SMem

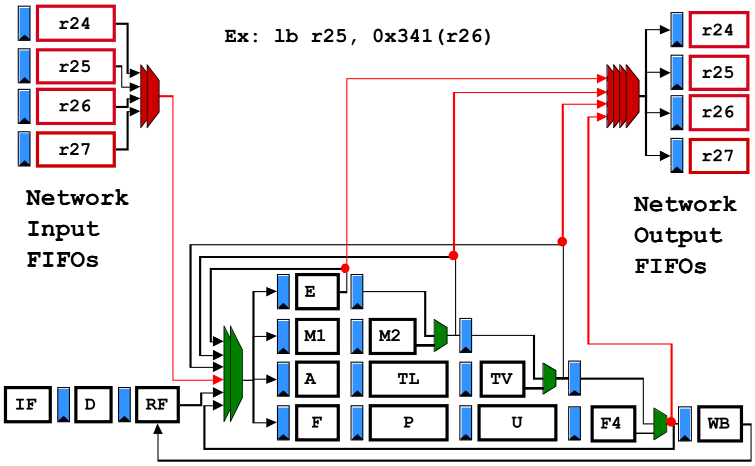
How does the main pipeline interface to the networks?

Memory mapped networks are not first class citizens.



To other tiles, through memory system that happens to go over a network.

Instead, Raw's networks are tightly coupled into the bypass paths



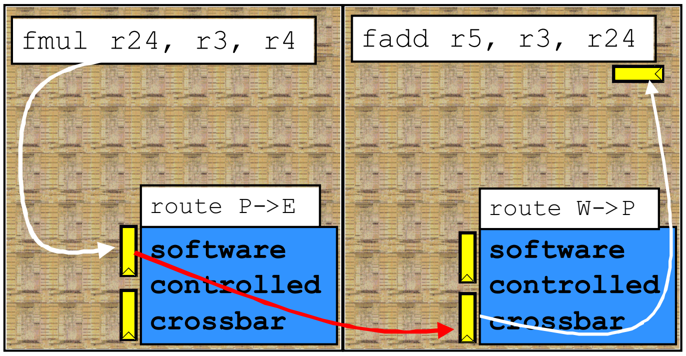
Ex: lb r25, 0x341(r26)

Network Input FIFOs

Network Output FIFOs

How the static router works.

Goal: flow controlled,
in order delivery of operands

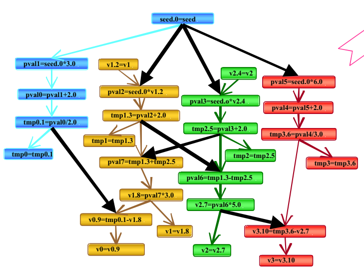


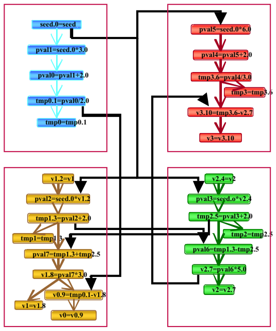
```
tmp0 = (seed*3+2)/2
tmp1 = seed*v1+2
tmp2 = seed*v2 + 2
tmp3 = (seed*6+2)/3
v2 = (tmp1 - tmp3)*5
v1 = (tmp1 + tmp2)*3
v0 = tmp0 - v1
v3 = tmp3 - v2
```

RawCC Operation:

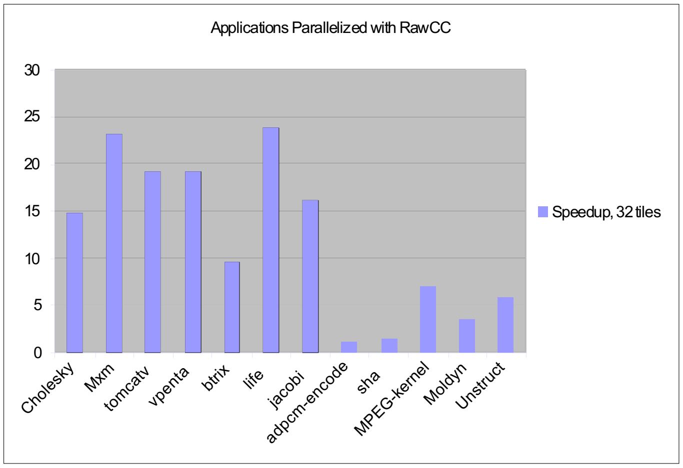
Parallelizes C code
onto static network

Black arrows =
Static Network

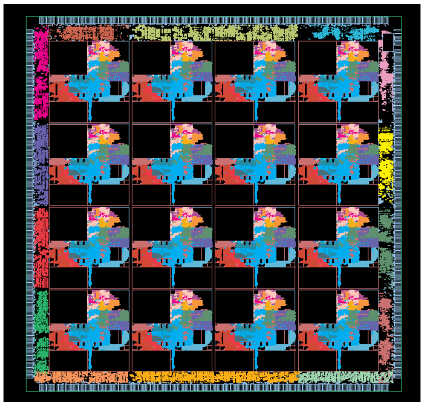




Low Network latency
important.



Raw Stats



IBM SA-27E .15u 6L Cu

18.2mm x 18.2mm die.

.122 Billion Transistors

16 Tiles

2048 KB SRAM Onchip

1657 Pin CCGA Package
(1080 HSTL signal IO)

~225 MHz

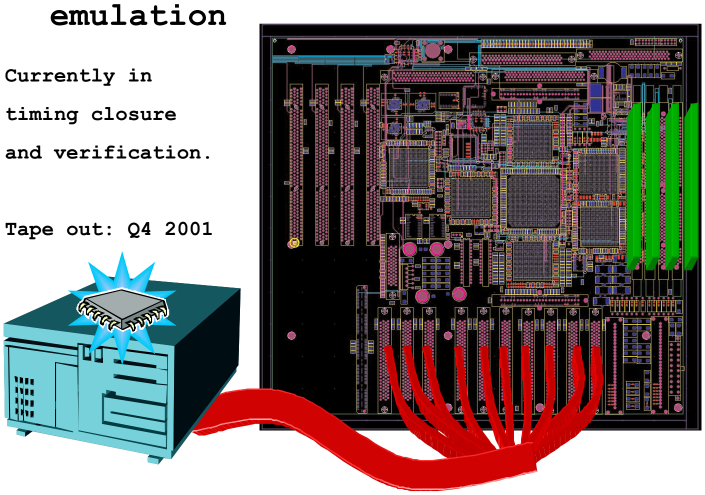
~25 Watts

For architectural details, see:
http://cag.lcs.mit.edu/pub/raw/documents/RawSpec99.pdf

Raw Board with IKOS logic emulation

Currently in
timing closure
and verification.

Tape out: Q4 2001



Enabler: The Raw Networks

The Raw ISA treats the networks as first class citizens, just like registers:

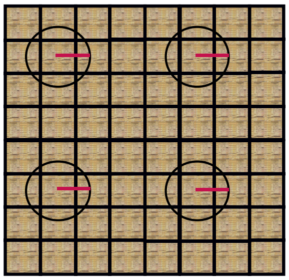
software managed,
bypassed,
encoding space in every instruction

Static Network :

1. routes compiled into static router SMEM
2. Messages arrive in known order

Latency: 2 + # hops
Throughput: 1 word/cycle per dir. per network

Summary



Raw exposes wire delay at the ISA level. This allows the compiler to explicitly manage gates in a scalable fashion.

Raw provides a direct, parallel interface to all of the chip resources: gates, wires, and pins.

Raw enables the use of these gates by providing tightly coupled network communication mechanisms in the ISA.

34th International Symposium on Microarchitecture (MICRO-34), Austin, TX, December 2001.

Direct Addressed Caches for Reduced Power Consumption

Emmett Witchel, Sam Larsen, C. Scott Ananian and Krste Asanović

MIT Laboratory for Computer Science
{`witchel, slarsen, cananian, krste`}`@lcs.mit.edu`

Abstract. A *direct addressed cache* is a hardware-software design for an energy-efficient microprocessor data cache. Direct addressing allows software to access cache data without a hardware cache tag check. These tag-unchecked loads and stores save the energy of a tag check when the compiler can guarantee an access will be to the same line as an earlier access. We have added support for tag-unchecked loads and stores to C and Java compilers. For Mediabench C programs, the compiler eliminates 16–76% of data cache tag accesses, with half of the benchmarks avoiding over 40% of the data tag checks. For SPECjvm98 Java programs, the compiler eliminates 18–63% of data cache tag checks. These tag check reductions translate into data cache energy savings of 9–40%, and overall processor and cache energy savings of 2–8%.

1 Introduction

Reducing energy consumption is an important goal for processors that will be used in battery-powered devices. Caches consume a large portion of total energy in processors targeted at low-power applications. For example, 16% of the total processor and cache power for the StrongARM microprocessor is dissipated in the data cache [6].

Commercial low-power processors usually employ associative caches [2, 6, 10, 13, 18]. For associative caches, a significant portion of the total access energy is spent checking multiple tags to find where data resides in the cache. For example, the highly-associative low-power cache designs used by the StrongARM and Xscale processors expend over 50% of the total cache access energy in the tag check [20].

In this paper, we propose a new hardware-software interface to reduce the energy cost of accessing cache data. *Direct addressing* allows software to access cache data without the hardware performing a cache tag check. These *tag-unchecked loads and stores* save the energy of performing a tag check when the compiler can guarantee an access will be to the same line as an earlier access. If the compiler cannot determine this information, or if cache lines are evicted due to interrupts or cache invalidations, direct addressing gracefully degrades back to conventional tag-checked accesses.

We have implemented compiler support for direct addressing in the SUIF C compiler [8], and in FLEX, a Java bytecode-to-native compiler [7]. We evaluate our compiler algorithms using C programs from Mediabench, and Java programs from SPECjvm98. Our results show we can eliminate 16–76% of all data cache tag accesses in C, and 18–63% of data cache tags checks in Java. We have developed a detailed energy model of a power-optimized microprocessor and caches. The reduction in cache tag checks results in data cache energy savings of 9–40% in C and 9–31% in Java. The total processor plus cache energy savings is 2–8%.

The paper is structured as follows. First we review current cache design in Section 2. Section 3 describes the changes needed to implement direct addressing. General compiler algorithms to support direct addressing are discussed in Section 4. The algorithms and results specific to C are described in Section 5, and the algorithms and results for Java in Section 6. Section 7 compares direct addressing to hardware schemes that remove tag checks. Finally, we discuss related work and conclude.

2 Low-power cache designs

Figure 1 shows the structure of a conventional virtually-indexed, virtually-tagged set-associative RAM-tagged cache (for brevity, only virtual caches are considered here, but direct addressing can be applied to other types of caches). An index taken from the virtual address is used to select a set consisting of several ways, and the tag field of the virtual address is compared against the tags in all ways to determine the location of the data. An n-way associative cache performs n tag checks and n data reads in parallel, discarding all but one of the data values depending on the tag compares.

An alternative approach, used in many low-power microprocessors [2, 6, 10, 13, 18], is to store the tags in content-addressable memory (CAM). The tag is broadcast across the cache lines and only the line whose tag matches has its data read out. The energy consumption of a 32-way CAM-tag search is approximately the same as a 2-way set

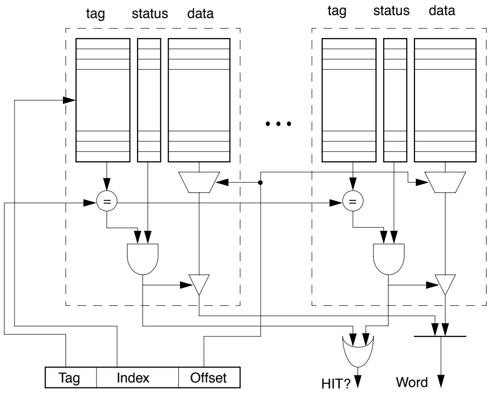


Fig. 1. A set-associative RAM-tag cache.

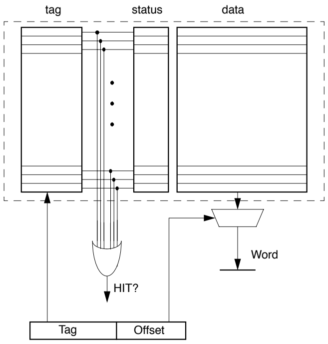


Fig. 2. A highly-associative CAM-tag cache subbank.

associative RAM-tag search [20, 2] but has lower miss rates. Caches are often subbanked to save energy and reduce delay, and a CAM-tag cache subbank is shown in Figure 2. Although CAM-tag caches reduce miss rates and hence total absolute memory access energy, they expend relatively greater energy in tag checks. Detailed HSpice simulations of a 16 KB CAM-tagged data cache divided into 1 KB subbanks, shows that the tag check consumes 54% of cache energy for loads and 43% for stores.

For both RAM and CAM tag caches, searching tags is expensive. If we could shortcut the process, by letting software tell the hardware in which way the line is located, we could save significant energy. The problem is how to let software directly access cache lines without compromising inter-process protection and while preserving correct operation in the face of cache replacements or other cache coherence actions.

3 Direct addressing

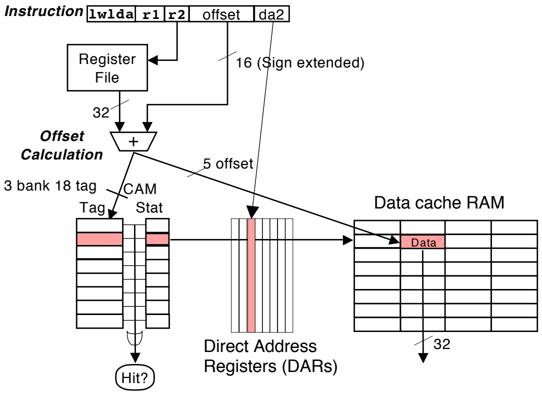


Fig. 3. A CAM-tagged data cache with direct addressing. The `lwlda` instruction causes `da2` to memoize the location of the data. A subsequent `lwda` that used `da2` would not power up the CAM bank on the left, but use the shaded DAR to pick this line.

Our approach to eliminating tag checks is to let software tell the hardware to remember the location of a cache line, so when software accesses the line again, hardware can access the data directly without searching tags. We augment the processor state with some number of *direct address registers* (DARs). These registers are set and used by software, and contain enough information to specify the exact location of a cache line in the cache data RAM as well as a valid bit. The exact width and data layout of the DARs is hidden from software to avoid exposing the implementation-dependent structure of the cache. In particular, software is only made aware of the length of a cache line, but not the total cache capacity or associativity.

Table 1 shows the instruction extensions for using DARs. Software places values in the DARs as an optional side-effect of performing a load or store. A tag-unchecked load or store specifies a full effective virtual address in addition to a DAR number. If the DAR is valid, its contents are used to avoid a tag search; if it is invalid, hardware

Instruction	*Explanation*
`(l|s)wlda rt, off(rs), da`	Load or store word, load direct address. Perform regular load or store, and also set the direct address register `da` to the location of the referenced line.
`(l|s)wda rt, off(rs), da`	Load or store word, using direct address. Data from the cache line pointed to by `da` is transferred to register `rt` (or the contents of `rt` is stored into the line specified by `da`). The line offset bits of `rs + off` are used to pick the proper word in the line. If `da` is invalid, the instruction acts like `(l|s)wlda`, accessing memory and setting the `da` register.
`jr.dainv rs, da_mask`	Jump register and invalidate direct address registers. It acts like a jump register instruction, and also clears the valid bit on the DARs specified in the bitmask. It is used on function return to invalidate the DARs used by the function.

Table 1. A table of instruction set extensions for manipulating direct address registers. MIPS is the base ISA and a machine with 8 DARs is described. Only word accesses are shown, but half-word and byte accesses are handled analogously.

falls back to a full tag search using the entire virtual address. The implementation described here uses a separate DAR specifier in each instruction, which takes 3 bits from the 16-bit immediate offset. An alternative encoding is to implicitly associate a DAR with some set of base registers, which reduces ISA changes at the cost of complicating compiler register allocation. We do not consider this option further in this paper.

Direct addressing is only used for data caches. Instruction caches have very regular access patterns and are only accessed via the program counter, and hence are amenable to software-invisible microarchitectural techniques to remove tag checks [16, 18, 19].

As an example, consider the function entry code in Figure 4, and a transformation of that code using direct addressing. The `swlda` instruction sets up the `da0` DAR, which is then used by the following `swda` instructions to eliminate cache tag checks. Note that no additional instructions were added and that performance is identical.

```
Old Code           New Code
sub $sp,64        sub   $sp,64
sw  $ra,60($sp)   swlda $ra,60($sp),$da0
sw  $fp,56($sp)   swda  $fp,56($sp),$da0
sw  $s0,52($sp)   swda  $s0,52($sp),$da0
```

Fig. 4. Example function entry code transformed to use DARs.

3.1 DAR implementation

At minimum, a DAR need only record the matching way within the cache set. In this case, the effective address is used to obtain the subbank number, the set index, and the offset within the cache line. In some implementations, however, it will be advantageous to also record subbank and set index information in the DARs and to physically distribute the DARs among the cache subbanks. This avoids recalculating and retransmitting these portions of the virtual address for tag-unchecked accesses.

The DARs incur additional area, energy, and delay overheads. The primary energy penalty is the parasitic load of the DARs on the signal lines driving the cache, but this should be a negligible fraction of overall cache access energy. The delay penalty is a single mux to select either one of the DARs or the normal cache access signal.

For a RAM-tag cache, the DARs can record way hit/miss information locally in each way (each way is a subbank). For a tag-unchecked access, the DAR specifier is broadcast to the ways, which replay the hit/miss information recorded in the local DAR latches without performing a tag check. The area and energy overhead of the DAR bits is small compared to the cache itself. The delay penalty is only a fraction of a gate delay as the DAR hit/miss signal can be folded into the existing precharged tag comparator.

For a CAM-tag cache, a DAR would be implemented as a unary bit vector with a single bit set on the matching row. Each cache row would locally store one bit per DAR. The DARs would be written with the local hit/miss signals generated by the CAM tag in each row.

For regular accesses, the parasitic energy overhead of the DARs is small because at most only one row's hit signal transitions high and one row's hit signal transitions low on any search. There is an additional energy cost to writing a DAR, where the DAR clock line has to transition high and low, but this overhead is small compared to the saving from not driving multiple bits of address across the tag array when the DAR is next used. As with the RAM-tag cache, the delay penalty is small if the DAR hit/miss signal is folded into the precharged match comparator.

3.2 DAR coherence

The DARs must be kept coherent with the state of the cache. If a line pointed to by a DAR is evicted, the DAR contents are no longer valid and cannot be used in a tag-unchecked access. Lines may be evicted either as a result of cache line replacement, or by external invalidate requests to maintain cache coherence with other processors or DMA I/O traffic.

To maintain coherence, each DAR can be tagged with the address of the cache line to which it points. On any eviction, the DAR tags are searched associatively and matching DARs are invalidated. The next use of an invalid DAR will cause a regular tag-checked access (which will usually miss). The DAR address tags need hold only a portion of the entire address allowing only a partial compare against the victim address, trading off some additional spurious invalidations for reduced complexity. In the extreme case, the DAR tags can be omitted with all DARs invalidated on any eviction.

The validity of the DARs can be checked right after the instruction decode of a tag-unchecked access. If the register is not valid, the access is converted into a regular tag-checked access early in the instruction pipeline, well before reaching the memory access stage. This avoids any additional memory access latency for checking valid bits.

4 Compiler algorithms for using DARs

Direct addressing has been implemented in two compiler systems, a SUIF-based C compiler and the FLEX Java native compiler. This section describes compiler algorithms common to both systems.

Both compilers use the same two step approach to eliminate tag checks with direct addressing. First, find two references, one of which dominates the other, so all paths that cause the subordinate access to be executed cause the dominant reference to be executed first. Second, prove that the two references always point to the same cache line. The second reference can then skip the tag check, by having the dominant reference write a DAR that the subordinate reference reads. Any other code between the two references, including assignments, control flow, or even function calls, can not affect correctness because hardware will invalidate DARs that point to lines that get evicted between the definition and the use of a DAR (as discussed in Section 3.2 above).

Both compilers control the stack pointer, ensuring it remains aligned to a cache boundary to simplify the determination of when two stack variables are on the same cache line. This allows easy transformation of function entry/exit code (as in Figure 4), spill code, parameter passing code, and access to automatic variables. The C and Java compilers use different methods to determine if two references to non-stack data (heap and static data) are to the same cache line. These are discussed in Sections 5.1 and 6.1 respectively.

4.1 DAR allocation

Each dominant reference with at least one subordinate reference to the same cache line is marked as a candidate for a DAR. The DAR allocation problem is an instance of the standard register allocation problem — DAR candidates that are live at the the same program point interfere and need to be allocated to different hardware DARs. DAR allocation is simpler than processor register allocation because DARs can not be spilled. Instead of spilling, a DAR is simply not allocated to a problematic DAR candidate.

The metric of utility we use for allocation is the number of tag checks eliminated by a certain DAR candidate minus the number of tag checks eliminated by the DAR candidates with which it interferes. This causes small, non-interfering ranges to get good coverage, and the most important variables in regions of heavy DAR use are prioritized.

4.2 DARs and calling conventions

The compilers analyze one function at a time, and the DARs are caller invalidated—at function exit, the compiler invalidates the DARs used in the function. If a function has a DAR live (say `da3`), and it makes a function call, the called function might invalidate `da3`, forcing a tag check on the use of that register. To reduce the impact of inter-procedural DAR invalidates, we randomly permute the DAR numbers used by the allocator. So one function might use registers in the order 7,2,3,6,0,5,1,4, another in the order 5,1,0,7,2,6,4,3.

Random permutation is much simpler than inter-procedural analysis, and makes collisions between register numbers much less likely than if every function used the same order. Interference is very low, and is quantified for C programs in Table 2 and for Java programs in Table 3.

5 C compiler implementation

We employ alignment and distance analysis for C to determine if two references are to the same cache line. This section first describes alignment and distance analysis in our C compiler, and then discusses the results of our experiments.

5.1 Alignment analysis in C programs

Alignment analysis attempts to determine the address alignment of each static memory reference relative to a cache line boundary. A value of 24 would indicate that the associated memory reference always accesses an address that is 24 bytes offset from the start of the cache line. A load or store instruction is considered *aligned* when its cache alignment is the same for each dynamic execution of the instruction. For instance, a global scalar resides in a static memory location and therefore always occupies a set alignment within the cache. For the majority of memory operations however, this will not be the case. Consider the loop in Figure 5(a). Here, the store instruction will access sequential cache locations in each loop iteration and is therefore *unaligned*.

In order to increase the percentage of aligned memory operations, our compiler performs a series of alignment-increasing transformations. One of the most important is loop unrolling. The code in Figure 5(b) shows the original loop with unrolling. After unrolling the loop by a factor consistent with the size of the cache line, we can guarantee that each memory operation in the loop only accesses the cache with a certain alignment. This is the case in our example assuming that A is an array of 64-bit data, and the cache line size is 32 bytes.

Since inner loops comprise the majority of dynamically executed instructions, it is very important that we uncover as much alignment information as possible from the body of an inner loop. Loop unrolling is effective for array references when the array is a local or global variable. However, if the array in Figure 5 is passed as an argument to the enclosing function, then loop unrolling does not enable the analysis to guarantee alignment for the memory references within the loop since the base of the array is unknown. Even worse is the case when the base

```
for (i=0; i<N; i++) {  for (i=0; i<N; i++) {
  A[i] = 0;                if (&A[i]
}                              % line_size == 0)
                           break;
 (a)                        A[i] = 0;
                       }
for(i=0; i<N;          for (; i<N; i += 4) {
         i += 4) {         A[i + 0] = 0;
A[i + 0] = 0;              A[i + 1] = 0;
A[i + 1] = 0;              A[i + 2] = 0;
A[i + 2] = 0;              A[i + 3] = 0;
A[i + 3] = 0;          }
}
 (b)                                       (c)
```

Fig. 5. (a) A simple loop with a single memory reference. **(b)** After loop unrolling. **(c)** A pre-loop inserted to guarantee alignment in the unrolled loop body.

of the array is actually aligned differently for different invocations of the function.

To overcome this limitation, our compiler inserts a pre-loop that runs for a small number of iterations until the references within the loop reach a known alignment. The code then jumps to an unrolled version of the loop where the alignment of references within the unrolled body are guaranteed (Figure 5(c)). Using this technique, the alignment analysis can determine the alignment for the majority of dynamically executed memory accesses. In order to limit the number of pre-loop iterations that are executed, our compiler also uses profile-driven feedback to determine the best conditions to begin execution of the unrolled loop.

One disadvantage of using loop unrolling to obtain alignment information is that too much unrolling can increase I-cache pressure [11]. We did not measure the impact of this effect.

5.2 Distance analysis

Distance analysis attempts to determine the byte distance between the addresses of two static memory references. The algorithm is implemented as a dataflow analysis that operates on low-level address calculations. If the difference between address calculations is a constant, then we know the distance between the references.

In the initial compiler passes, when array accesses are represented at a high level, we tag them with their source array to aid in distance analysis. We use this tag once the array access has been decomposed into pointer manipulation. For accesses of the form $A[i]$ and $A[i+c]$, our tagging allows us to compute the distance as c. This pattern occurs very frequently in unrolled loops.

We deal with aliasing using local information. To be conservative, we assume a pointer variable can point to any globally visible address. So a DAR definition and use will not span a pointer store to a base with a globally visible address.

Once we know the distance, we can use the alignment to determine if two references are to the same cache line. We find the alignment of the dominant reference relative to the cache line boundary and then find the distance between the subordinate access and the dominant access. Simple arithmetic indicates if the references are on the same cache line. An important special case is when the distance is 0, in which case we do not need to consult the alignment information.

5.3 C evaluation

We used the SUIF compiler [8] to output instrumented C code. It acts like a C compiler with C as its target architecture. A disadvantage of this approach is that the instrumented C code does not capture stack references for function entry/exit, spill code and parameter passing. This will tend to underestimate the benefit of direct addressing as stack references provide many direct addressing opportunities, as quantified below in the Java evaluation.

The instrumented code has loops unrolled and is augmented with statistics gathering code. Every load and store in the program is analyzed and converted into a function call to our model. We verify at runtime that our static analysis was accurate.

5.4 C results

Figure 6 shows how many tag checks were eliminated for loads and stores for the Mediabench programs. From the number of tag checks eliminated, we computed the D-cache energy savings based on our extracted layout for the cache [15]. This model has tag search consuming 54% of a load and 43% of a store, broken down further into 10%/8% (load/store) for address bus, 25%/40% for data access, and 11%/9% for data bus.

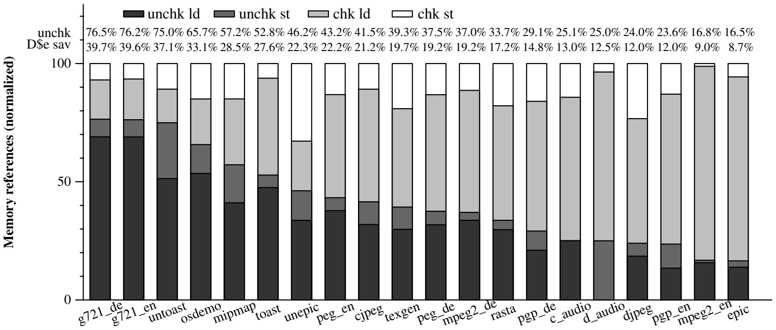


Fig. 6. Tag check elimination for Mediabench programs compiled by SUIF, using 8 DARs. The lowest part of the bar is tag unchecked loads, then unchecked stores. Over that are tag checked loads and stores. The number on top of each bar (unchk) is the percentage of tag checks eliminated. The number under that (D$e sav) is the percentage of dcache energy saved by eliminating the checks.

The results vary widely, with over 76% of checks eliminated for g721 decode (39.7% savings in data cache energy), down to 16.5% for epic. Direct addressing saves some energy on every application and even the small 8.7% energy savings on epic is likely to be larger than any overhead direct addressing introduces.

One reason for the spread is that some codes are more difficult to analyze, mostly due to pointer manipulation. One example is mpeg2 decode, for which the compiler was unsuccessful on the code as distributed with Mediabench. The code had one key loop which was manually unrolled, with a key matrix traversed in column-major order. By making four small edits to the source code to express the loop in a natural way, and to traverse the matrix in row-major order (which is also better for cache performance), the percentage of tag checks eliminated went from 6.2% to 37%.

Table 2 shows the data cache energy saved, and also the energy saving for the whole processor core including instruction and data caches. The energy consumption of the data cache relative to the entire core is highly dependent on the implementation. Our core design is highly optimized for low-power, consuming 100–250 pJ per instruction at 300 MHz in a 0.25 μm technology (<100 mW). For our design, we measured average data cache tag energy at 10% of the total core energy for Mediabench [15].

Benchmark	De^-	P+I +De^-	0off	8DAR lim	f()	r/w	# inst	# ld	# st	input
g721_de	39.7%	7.9%	11.6%	0.0%	0.0%	5.0	568719607	27155521	4567915	clinton.pcm
g721_en	39.6%	7.9%	12.0%	0.0%	0.0%	4.8	602714433	28470293	4557182	clinton.g721
untoast	37.1%	7.4%	2.7%	0.0%	0.0%	7.1	164673415	5211304	2741109	clinton.pcm
osdemo	33.1%	6.6%	6.0%	0.1%	0.2%	6.0	17768196	1005300	375520	out.ppm
mipmap	28.5%	5.7%	4.1%	0.0%	1.1%	5.6	50705063	3728696	1677708	out.ppm
toast	27.6%	5.5%	21.2%	28.0%	0.0%	11.6	325450576	33490458	4344146	clinton.gsm
unepic	22.3%	4.5%	40.8%	0.0%	0.0%	2.1	16471762	816031	676550	test_image.E
peg_en	22.2%	4.4%	61.4%	5.9%	0.0%	3.3	84217188	6191963	1415461	pgptest.plain
cjpeg	21.2%	4.2%	10.0%	1.9%	0.9%	5.9	35620933	2949426	758928	testimg.ppm
texgen	19.7%	3.9%	19.9%	2.1%	0.4%	2.2	146657184	9933559	3960265	out.ppm
peg_de	19.2%	3.8%	72.3%	0.2%	0.4%	2.9	46589722	3508783	818128	pegwit.enc
mpeg2_de	19.2%	3.8%	3.5%	0.0%	0.0%	2.7	270350477	19967230	3440148	mei16v2.m2v
rasta	17.2%	3.4%	35.5%	2.7%	0.0%	2.6	30132991	2866589	802709	map_weights.dat
pgp_de	14.8%	3.0%	70.6%	2.9%	0.1%	1.8	16299047	905321	287437	pgptext.pgp
c_audio	13.0%	2.6%	0.1%	0.0%	0.0%	13.7	18686936	443006	74056	clinton.pcm
d_audio	12.5%	2.5%	0.1%	0.0%	0.0%	13.7	17259137	369246	147816	clinton.adpcm
djpeg	12.0%	2.4%	14.4%	0.6%	0.3%	3.0	8882489	755752	305428	testimage.jpg
pgp_en	12.0%	2.4%	68.2%	2.0%	0.1%	1.8	28908438	1423749	428303	pgptest.plain
mpeg2_en	9.0%	1.8%	2.7%	0.0%	0.0%	5.6	3587002898	235379785	5349441	options.par
epic	8.7%	1.7%	24.7%	0.0%	0.0%	3.3	118204938	5971476	542458	test_image.pgm
average	21.4%	4.3%	24.1%							

Table 2. De^- is the data cache energy saved from eliminating tag checks. P+I+De^- is the energy saved for the processor plus instruction and data caches. 0off shows the percentage of tag unchecked accesses where the dominant and subordinate accesses were to the same address. f() shows how many tag checks happened as a result of function calls invalidating DARs. r/w gives the ratio of DAR reads to DAR writes. # inst gives the number of SUIF instructions executed by the benchmarks, and ld/st give the number of loads and stores.

The Table clearly shows the importance of offset information. While the results vary across benchmarks, most of the benefit of the DARs is not just from the program reusing the same location (0off column).

Our initial experiments indicated that 8 DARs captured most direct addressing opportunities across a range of benchmarks. The 8DARlim column shows how many more tag checks could be eliminated with an unlimited number of DARs versus the 8 used for the rest of the results. We compute this number by emitting liveness information for DAR candidates and doing post-hoc optimal register allocation. Only toast is able to soak up many more tag checks with more registers (it can profitably use 44). Every benchmark could make use of at least two DARs. Random permutation of register numbers makes the interference of function calls very small, as seen in the f() column. Finally, we see that each DAR value written is usually reused several times (r/w column), sometimes over 13 times, but averaging around 2–3 times.

6 Java implementation

Java bytecodes are normally interpreted directly or fed to a just-in-time compiler, but instead we used the FLEX compiler to compile Java bytecodes to MIPS assembly code. Java-to-native compilation is a good alternative for low-power environments if Java's dynamic loading capabilities are not usually needed, as the code can be highly optimized for low energy consumption.

The FLEX implementation used the same dominance analysis and DAR allocation algorithms as the SUIF implementation. The following sections first describe how heap memory references are mapped onto cache lines for Java programs, and then discuss the results of our experiments.

6.1 Object identity in Java programs

Our approach to finding references to the same cache line is different in Java than it was in C. Java's type-safety and object-orientation means there is additional pointer information available to the compiler.

All memory for Java objects comes from the system allocator. We modify the memory allocator to ensure that small objects are never split across cache lines and that larger objects are always aligned to the start of a cache line. The compiler can then simply determine cache-line equivalence based on object type and member field offset. This determination is performed on a very low-level representation just prior to instruction selection, so even access to object header words (like the class descriptor and hashcode) are visible to this "cache-line equivalence" analysis. This modified allocation policy potentially introduces fragmentation, which the allocator could deal with, e.g., by tracking "holes" and filling them in with small objects.

This type-based analysis is very simple, but accounts for a large number of eliminated tag checks in strongly object-oriented benchmarks like jess or jack. For more traditionally coded benchmarks, such as compress, there is need for further cache-line equivalence analysis of indexed array operations.

As with the C implementation, loops are unrolled in Java to expose more direct addressing opportunities. The unrolling strategy in Java is simpler: each loop which mentions an array is unrolled C/E times, where C is the cache line length, and E is the element size of the array with the smallest elements in the loop. This may over-unroll some loops, but guarantees that almost all the direct addressing opportunities are exposed. If the first element accessed in the loop is not cache-line aligned, extra checks are placed within the unrolled loop to catch cache-line boundary crossings.

To further expose direct addressing opportunities and improve performance, the FLEX compiler inlines small final methods.

6.2 Java evaluation

FLEX outputs the MIPS instruction extensions for direct addressing (Table 1). Due to the limited number of offset bits in the instruction encoding, some loads (that use the global pointer) take one instruction while some loads (to data that is further than 32 KB from any register) take two instructions. The GNU assembler was modified to accept these instructions, and our extended MIPS ISA simulator models the state of the DARs (with dynamic correctness checks of DAR use). The Java runtime is written in C, and was compiled with gcc 2.7.2 with a MIPS target. The runtime is linked with the assembled Java code to give a MIPS binary that is run on the simulator.

The Java garbage collector was disabled for all runs. The collector, like the runtime, is written in C. The collector moves large amounts of data in memory with exact knowledge of object size and alignment, and so we expect that it could make heavy use of direct addressing. Modifying the collector was beyond the scope of these experiments, but including the modified collector should only improve the relative benefits of direct addressing.

Instead of modifying the system memory allocator to ensure cache alignment of heap data, we instead used conventional malloc and modified our checking code to ensure that all references are to the same 32-byte block of memory regardless of alignment.

6.3 Java results

Table 3 shows the percentage of tag checks eliminated for Java SPECjvm98 programs. Unlike our C evaluation, we ran each Java binary on the detailed energy simulator [15] to get exact energy dissipation numbers (except for mpegaudio which ran for too long and was estimated at 10%, as with the C benchmarks). Data cache tag check energy consumption was computed to be almost exactly 10% for every benchmark except raytrace, which has many memory accesses, and dissipates 13% of its energy in data cache tag checks.

The nSP column shows how many of our eliminated tag checks are to non-stack memory accesses. Most of the stack accesses are function entry/exit, and these are easy for the compiler to transform. The data

Benchmark	ntag	D\$$e^-$	Te^-	nSP	0off	f()
jess	62.8%	31.0%	6.2%	12.6%	2.0%	1.3%
jack	58.2%	28.0%	6.1%	43.3%	15.9%	0.4%
raytrace	56.7%	27.6%	7.6%	4.7%	0.6%	0.1%
compress	53.4%	26.3%	5.5%	26.4%	4.8%	1.2%
db	51.8%	25.7%	5.5%	5.2%	2.1%	1.6%
mpegaudio	18.0%	9.3%	1.8%	50.3%	25.2%	1.2%

Table 3. All benchmarks were run with -s10, which is the middle sized spec input. ntag is the number of data cache tag accesses eliminated. D\$$e^-$ is the data cache energy saved from these eliminated tag checks. Te^- is the energy saved for the processor plus instruction and data cache. nSP is the percentage of memory references that were tag unchecked, but did not reference the stack. 0off is the percentage of tag checks eliminated whose dominant and subordinate reference were to the exact same address. f() is the percentage of tag checks caused by having a function call invalidate a live DAR.

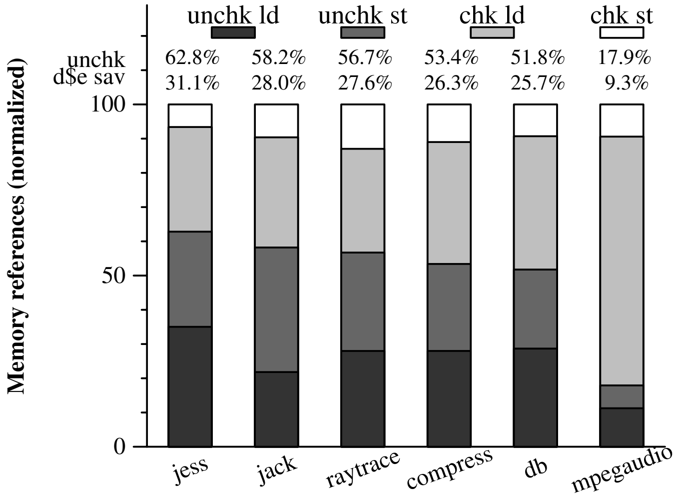


Fig. 7. Tag check elimination for SPECjvm98 programs compiled by FLEX using eight DARs.

for Java shows that stack references are about half (46–79%) of all memory references for SPECjvm98, and our analysis eliminates 67–82% of tag checks for these references. This gives an indication of the expected improvement if stack accesses were included in the SUIF C evaluation.

Table 3, like Table 2, shows the necessity of offset information. The number of zero offset references (where the dominant and subordinate access are to the same location) is lower in Java than in C because much of the tag check elimination comes from stack accesses on function entry and exit. These accesses load or store registers to sequential locations on the stack.

The f() column is the percentage of accesses that have to be tag checked because a function call between a DAR definition and use invalidated the DAR. As with our C benchmarks, random permutation of DAR numbers keeps this interference low.

Finally in Table 4, the mSP column shows that by ignoring spill code and parameter accesses, we are not missing a major opportunity. The generally low numbers indicate that the register allocator is not doing excessive spilling.

Mpegaudio sticks out because there is excessive spilling in this benchmark. Transforming the spill code to use direct addressing would get us a large part of the 52.0% of stack references which are not being analyzed. This would bring mpegaudio into the 50–60% tag elimination range of the other applications.

In order to transform spill code, we would generalize our direct register analysis and allocation to work on the post-register allocated version of the program (all the needed information is still available in FLEX).

Benchmark	Jinst	Jrefs	JavaSP	RunSP	mSP	# inst	# ld	# st
jess	44.6%	45.9%	66.2%	59.1%	0.5%	386362871	74217394	38927873
jack	60.0%	51.6%	45.2%	55.1%	4.4%	742795569	97902751	83399493
raytrace	19.1%	12.8%	79.7%	26.2%	6.9%	711506624	121545307	87011062
compress	99.7%	99.5%	49.3%	11.4%	1.8%	1995067192	318765365	182481314
db	63.7%	49.2%	53.9%	55.2%	0.2%	229082873	36830775	17634446
mpegaudio	7.9%	4.6%	62.9%	24.4%	52.0%	3798725510	860533959	164886641

Table 4. All benchmarks were run with -s10, which is the middle sized spec input. Jinst is the percentage of instructions executed in Java code. The remainder executed in the runtime. Jrefs is the percentage of memory references issued in Java. JavaSP is the percentage of Java memory references that are to the stack. RunSP is the percentage of memory references made to the stack by the Java runtime. mSP is the maximum possible contribution to the tag unchecked references if we converted every remaining stack access–namely spill code and parameter access. # inst/ld/st are the numbers of instructions, loads and stores from the Java code, not including the runtime.

7 Comparison with hardware tag-check elimination schemes

In this section, we compare our direct addressing scheme for eliminating tag checks at compile time with dynamic hardware alternatives that are invisible to software. One approach is for the hardware to remember the tag of the last cache line that was accessed and to compare this against the tag of the next memory access before enabling the tag search [2]. The main disadvantage of this scheme is that it adds a wide tag compare into the critical path of every cache access, adding several gate delays to this latency-sensitive path. A variant of this scheme is to remember the last line accessed within each cache subbank, and only power up cache tags if a different line is accessed within each subbank.

Table 5 compares results for the C and Java benchmarks using these two schemes. Using 8 DARs usually removes more tag checks than a hardware single last line buffer without the additional access latency, although with `pgp` the hardware scheme is significantly better. The hardware and software techniques can be combined, with the last line buffer used in cases where the DARs were not specified or unsuccessful. In this case, accesses will incur the additional cache access latency of the hardware scheme. The results in the fourth column of Table 5 show that combining the techniques usually does better than using each alone, indicating that they are capturing different types of cache line reuse.

The fifth column in Table 5 shows the results for the per-subbank last line buffer (16 subbanks). This removes many more tag checks than the previous schemes, but requires an extra tag comparator in each subbank and incurs the additional memory access latency. Finally, the sixth column shows the effect of adding 8 DARs to the per-subbank last line buffers. Here, there is little additional benefit (except for `mipmap`) as the hardware scheme has captured most of the available cache line reference locality.

The results for the Java benchmarks are similar, with the hardware last line scheme eliminating roughly the same number of tag checks as the 8 DAR scheme, but with the additional memory access latency. There is a smaller benefit to combining the hardware and software schemes for the Java programs, because the DARs only give benefit to the hardware schemes where the analysis was successful, as in jess and jack. Again, the per-subbank last line scheme performs well, removing 80–90% of all tag checks.

Program	8 DAR	last ln	last ln + 8 DAR	ll-sub	ll-sub + 8 DAR
C Benchmarks					
g721_de	76.5%	73.5%	82.1 +08.6%	98.4%	98.4 +00.0%
g721_en	76.3%	73.2%	81.7 +08.5%	98.4%	98.4 +00.0%
untoast	75.0%	39.6%	82.3 +42.7%	97.3%	97.5 +00.2%
osdemo	65.7%	47.8%	75.6 +27.8%	86.4%	88.4 +02.0%
mipmap	57.2%	22.5%	64.6 +42.1%	60.1%	85.7 +25.6%
toast	52.9%	15.0%	87.7 +72.7%	91.4%	98.5 +07.1%
unepic	46.2%	57.0%	71.2 +14.2%	81.0%	83.9 +02.9%
peg_en	43.2%	27.6%	46.6 +19.0%	65.5%	67.6 +02.1%
cjpeg	41.5%	17.5%	50.0 +32.5%	74.5%	79.6 +05.1%
texgen	39.3%	36.4%	56.2 +19.8%	78.4%	83.7 +05.3%
peg_de	37.5%	19.4%	41.2 +21.8%	59.0%	62.0 +03.0%
mpeg2de	37.1%	7.7%	40.4 +32.7%	84.9%	86.1 +01.2%
rasta	33.7%	19.1%	43.9 +24.8%	81.0%	85.8 +04.8%
pgp_de	29.2%	46.1%	57.2 +11.1%	89.8%	91.4 +01.6%
c_audio	25.1%	0.1%	25.1 +25.0%	65.1%	68.8 +03.7%
d_audio	25.1%	0.1%	25.1 +25.0%	58.1%	61.7 +03.6%
djpeg	24.0%	17.2%	30.5 +13.3%	64.5%	67.5 +03.0%
pgp_en	23.6%	54.5%	67.6 +13.1%	95.4%	96.8 +01.4%
mpeg2en	16.8%	7.9%	22.1 +14.2%	88.7%	89.6 +00.9%
epic	16.5%	8.9%	19.1 +10.2%	70.7%	72.4 +01.7%
Java Benchmarks					
jack	58.2%	54.6%	66.0 +11.4%	88.9%	90.7 +01.8%
raytrace	56.7%	66.9%	68.2 +01.3%	90.3%	90.7 +00.4%
compress	53.4%	54.8%	61.6 +06.8%	80.9%	82.3 +01.4%
jess	62.8%	58.2%	73.6 +15.4%	84.2%	87.7 +03.5%
db	51.8%	50.0%	62.5 +12.5%	81.2%	83.9 +02.7%
mpgaudio	18.0%	27.8%	36.5 +08.7%	81.4%	83.5 +02.1%

Table 5. Tag checks eliminated by 8 direct address registers (DARs), by a last line hardware tag compare (last ln), by adding 8 DARs to a single line buffer, by per-subbank last line buffers (ll-sub) with 16 subbanks, and by adding 8 DARs to the subbank last line buffers. The hardware last line schemes add the latency of an additional tag compare to all memory operations.

8 Related work

The ARM instruction set includes load/store multiple instructions that can be used to avoid tag checks for sequential accesses to the same cache line [18]. These instructions are typically only used for procedure call/return, whereas our model allows significantly greater flexibility. For example, the results we presented for the C Mediabench code were for non-stack accesses which are much less amenable to load/store multiple.

Some researchers [2, 14] have described hardware L0 caches designed for low power access. These schemes have performance impacts, whereas the direct addressing scheme does not affect performance. Direct addressing can also be combined with some of these hardware schemes to save further power.

Other researchers [4, 9, 17] have developed software caching schemes that use compile-time information to reduce software tag checks. FlexCache [17] adds HotPage registers, which are similar to DARs except they also hold a tag along with the direct address. They are used as a small compiler-managed hardware tag array for a software associative cache. The HotPage-likely compiler analysis implements static software way-prediction to index the likely HotPage register holding the translation for a given memory access. The speculation is checked by a hardware compare of the virtual address with the HotPage tag. The authors mention that an additional optimization, HotPage-predictable analysis [4], could avoid this tag check but do not include compiler algorithms or results. In contrast, our work removes tag checks from a hardware associative cache scheme with no performance penalty, and our compiler analysis avoids tag checks by statically guaranteeing two accesses are to the same line.

Fisher [12] and Ellis [3] were the first to use loop unrolling to improve the alignment of memory references in a loop body. Their work was done in the context of a clustered VLIW in which main memory was divided among separate banks. Their architecture supported a fast path to memory when data were located on a cluster's local memory bank. Alignment of memory operations was therefore an important factor in machine performance.

Barua et al. expanded on these ideas and introduced Modulo Unrolling [1]. This work introduced precise equations for determining the unroll factors for loop nests. In Modulo Unrolling, outer loops may be unrolled to create aligned references outside the inner loop. This work was done in the conext of the RAW machine [5] in which processor memory is distributed across processing tiles. As is the case with the clustered VLIW, access to a local bank is faster than access to a remote bank.

9 Conclusions

Direct addressed caches provide a new hardware-software interface to use energy of cache accesses. Direct addressing uses compile-time information plus a minimal amount of hardware to remove data cache tag checks, thus saving energy. Our implementations of direct addressing in a C and Java compiler resulted in data cache energy savings from 9–40% for C and 9–31% for Java. In contrast to other cache energy saving techniques, direct addressing does not change the performance of the processor, it just reduces the amount of microarchitectural work the processor performs.

10 Acknowledgements

This work was funded by DARPA PAC/C award F30602-00-2-0562, NSF Grant CCR-0073510, DARPA/AFRL Contract F33615-00-C-1692, NSF Grant CCR00-86154, NSF Grant CCR00-63513, and NSF CAREER Grant CCR-0093354.

References

1. Rajeev Barua, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Memory Bank Disambiguation using Modulo Unrolling for RAW Machines. In *Proceedings of the Fifth International Conference on High Performance Computing*, Chennai, India, Dec 1998.
2. T. Burd. *Energy-Efficient Processor System Design.* PhD thesis, University of California at Berkeley, 2001.
3. J. R. Ellis. *Bulldog: A Compiler for VLIW Architectures.* The MIT Press, Cambridge, Massachussetts, 1985.
4. C. A. Moritz *et al.* Hot pages: Software caching for RAW microprocessors. *MIT-LCS Technical Memo LCS-TM-599*, August 1999.
5. Elliot Waingold *et al.* Baring It All to Software: RAW Machines. 30(9):86–93, Sep 1997.
6. J. Montanaro *et al.* A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *IEEE JSSC*, 31(11):1703–1714, November 1996.
7. M. Rinard *et al.* The FLEX compiler infrastructure. 1999–2001. `http://www.flex-compiler.lcs.mit.edu`.
8. M. S. Lam *et al.* The SUIF compiler system. 1992–2001. `http://www-suif.stanford.edu`.

9. O.S. Unsal *et al.* Cool-Cache for hot multimedia. In *MICRO–34*, 2001.
10. S. B. Furber *et al.* ARM3 - 32b RISC processor with 4kbyte on-chip cache. In G. Musgrave and U. Lauther, editors, *Proceedings IFIP TC 10/WG 10.5 Int. Conf. on VLSI (VLSI'89)*, pages 35–44. Elsevier (North Holland), 1989.
11. W. Lee *et al.* Space-time scheduling of instruction-level parallelism on a RAW machine. *ACM SIGPLAN Notices*, 33(11):46–57, 1998.
12. Joseph A. Fisher, John R. Ellis, John C. Ruttenberg, and Alexandru Nicolau. Parallel Processing: A Smart Compiler and a Dumb Machine. In *ACM SIGPLAN '84 Symposium on Compiler Construction*, pages 37–47, June 1984.
13. Intel Corp. *Intel Xscale core developers manual*, order no. 273473-001 edition, December 2000.
14. J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *MICRO–30*, pages 184–193, 1997.
15. R. Krashinsky. Microprocessor energy characterization and optimization through fast, accurate, and flexible simulation. Master's thesis, Massachusetts Institute of Technology, May 2001.
16. A. Ma, M. Zhang, and K. Asanović. Way memoization to reduce fetch energy in instruction caches. *ISCA Workshop on Complexity Effective Design*, July 2001.
17. C. A. Moritz, M. Frank, and S. Amarasinghe. Flexcache: A framework for flexible compiler generated data caching. *Lecture Notes of Computer Science, Springer–Verlag*, 2001.
18. M. Muller. Power efficiency & low cost: The ARM6 family. In *Hot Chips IV*, August 1992.
19. R. Panwar and D. Rennels. Reducing the frequency of tag compares for low power I-cache design. In *SLPE*, pages 57–62, October 1995.
20. M. Zhang and K. Asanović. Highly-associative caches for low-power processors. *Kool Chips Workshop, MICRO-33*, 2000.

Network 21

Proc. of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM 2000), Boston MA, pp. 120–130, August 2000.

A Scalable Location Service for Geographic Ad Hoc Routing

Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger and Robert Morris [⋆]

MIT Laboratory for Computer Science
`{jinyang, jj, decouto, karger, rtm}@lcs.mit.edu`

Abstract. GLS is a new distributed location service which tracks mobile node locations. GLS combined with geographic forwarding allows the construction of ad hoc mobile networks that scale to a larger number of nodes than possible with previous work. GLS is decentralized and runs on the mobile nodes themselves, requiring no fixed infrastructure. Each mobile node periodically updates a small set of other nodes (its location servers) with its current location. A node sends its position updates to its location servers without knowing their actual identities, assisted by a predefined ordering of node identifiers and a predefined geographic hierarchy. Queries for a mobile node's location also use the predefined identifier ordering and spatial hierarchy to find a location server for that node.

Experiments using the *ns* simulator for up to 600 mobile nodes show that the storage and bandwidth requirements of GLS grow slowly with the size of the network. Furthermore, GLS tolerates node failures well: each failure has only a limited effect and query performance degrades gracefully as nodes fail and restart. The query performance of GLS is also relatively insensitive to node speeds. Simple geographic forwarding combined with GLS compares favorably with Dynamic Source Routing (DSR): in larger networks (over 200 nodes) our approach delivers more packets, but consumes fewer network resources.

[⋆] Jinyang Li and John Jannotti are supported by DARPA contract N66001-99-2-8917. David Karger is supported by NSF contract CCR-9624239, an Alfred P. Sloane Foundation Fellowship, and a David and Lucille Packard Foundations Fellowship.

1 Introduction

This paper considers the problem of routing in large ad hoc networks of mobile hosts. Such networks are of interest because they do not require any prior investment in fixed infrastructure. Instead, the network nodes agree to relay each other's packets toward their ultimate destinations, and the nodes automatically form their own cooperative infrastructure. We describe a system, *Grid*, that combines a cooperative infrastructure with location information to implement routing in a large ad hoc network. We analyze Grid's location service (GLS), show that it is correct and efficient, and present simulation results supporting our analysis.

It is possible to construct large networks of fixed nodes today. Prominent examples include the telephone system and the Internet. The cellular telephone network shows how these wired networks can be extended to include large numbers of mobile nodes. However, these networks require a large up-front investment in fixed infrastructure before they are useful—central offices, trunks, and local loops in the case of the telephone system, radio towers for the cellular network. Furthermore, upgrading these networks to meet increasing bandwidth requirements has proven expensive and slow.

The fact that large fixed communication infrastructures already exist might seem to limit the usefulness of any competing approach. There are, however, a number of situations in which ad hoc networks are desirable. Users may be so sparse or dense that the appropriate level of fixed infrastructure is not an economical investment. Sometimes fixed infrastructure exists but cannot be relied upon, such as during disaster recovery. Finally, existing services may not provide adequate service, or may be too expensive.

Though ad hoc networks are attractive, they are more difficult to implement than fixed networks. Fixed networks take advantage of their static nature in two ways. First, they proactively distribute network topology information among the nodes, and each node pre-computes routes through that topology using relatively inexpensive algorithms. Second, fixed networks embed routing hints in node addresses because the complete topology of a large network is too unwieldy to process or distribute globally. Neither of these techniques works well for networks with mobile nodes because movement invalidates topology information and permanent node addresses cannot include dynamic location information. However, there is a topological assumption that works well for radio-based ad hoc networks: nodes that are physically close are likely to be close in the network topology; that is, they will be connected by a small number of radio hops.

Grid uses geographical forwarding to take advantage of the similarity between physical and network proximity. A source must know the geographical positions of any destination to which it wishes to send, and must label packets for that destination with its position. An intermediate node only needs to know its own position and the positions of nearby nodes; that is enough information to relay each packet through the neighbor that is geographically closest to the ultimate destination. Although Grid forwards packets based purely upon local geographic information, it is highly likely that packets are also approaching their destination as measured by the number of remaining hops to the destination. Because nodes only need local information, regardless of the total network size, geographic forwarding is attractive for large-scale networks.

However, to be useful in a larger context, a system based on geographic forwarding must also provide a mechanism for sources to learn the positions of destinations. To preserve scalability, this location service must allow queries and updates to be performed using only a handful of messages. Of course, the location service itself must operate using only geographic forwarding. It should also be scalable in the following senses:

1. No node should be a bottleneck—the work of maintaining the location service should be spread evenly over the nodes.
2. The failure of a node should not affect the reachability of many other nodes.
3. Queries for the locations of nearby hosts should be satisfied with correspondingly local communication. This would also allow operation in the face of network partitions.
4. The per-node storage and communication cost of the location service should grow as a small function of the total number of nodes.

The Grid location service (GLS) presented in this paper satisfies all of these requirements.

The rest of the paper describes the design and simulated performance of Grid. Section 2 reviews existing work in scalable ad hoc networking. Section 3 describes the characteristics of geographic forwarding. Section 4 describes Grid's distributed location service algorithm. Section 5 describes our implementation of geographic forwarding and the GLS in detail. Section 6 analyzes Grid's routing performance and scalability using simulations. Section 7 suggests areas for future improvements. Section 8 summarizes the paper's contributions.

2 Related work

Most existing ad hoc routing systems distribute either topology information or queries to all nodes in the network. Some, such as DSDV [17], are *proactive*; they continuously maintain route entries for all destinations. Other techniques are *reactive*, and construct routes to destinations as they are required. This includes systems such as DSR [11], AODV [16], and TORA[15]. Broch et al. [5] and Johansson et al. [10] each provide overviews of these ad hoc routing techniques, along with comparative measurements using small (30–50 node) simulations. Grid's main contribution compared to these works is increased scalability.

More closely related to Grid are protocols that use geographic positions. Finn's Cartesian routing [8] addresses each node with a geographic location as well as a unique identifier. Packets are routed by sending them to the neighbor closest to the packet's ultimate destination. Dead ends are handled by scoped flooding. However, Finn gives no detailed explanation of how node locations are found or how mobility is handled.

More recent work on geographic approaches to routing includes the DREAM [3] and LAR [14] systems. Both systems route packets geographically, in a manner similar to Finn's Cartesian system. They differ in how a node acquires the geographic position of a destination. DREAM nodes proactively flood position updates over the whole network, allowing other nodes to build complete position databases. LAR nodes reactively flood position queries over the entire network when they wish to find the position of a destination. Because they both involve global flooding, neither system seems suited to large networks.

The Landmark system [18, 19] actively maintains a hierarchy to provide routing in a changing network. Nodes in a Landmark network have unique permanent IDs that are not directly useful for routing. Each node also has a changeable Landmark address, which consists of a list of IDs of nodes along the path from a well-known root to the node's current location. A Landmark address can be used directly for routing, since it is similar to a source route. The Landmark system provides a location service that maps IDs to current addresses. Each node **X** sends updates containing its current Landmark address to a node that acts as its address server, chosen by hashing **X**'s ID to produce a Landmark address A. If a node **Y** exists with that address, **Y** acts as **X**'s location server. Otherwise the node with Landmark address closest to A is used. Anyone looking for **X** can use the same algorithm to find **X**'s location server, which can be queried to find **X**'s current Landmark address. This combination of location servers and addresses that encode routing information is similar to the architecture described in this paper. Grid, however, avoids building hierarchies, as they are vulnerable to the movement of nodes near the top of the hierarchy.

3 Geographic forwarding

We use a simple scheme for geographic forwarding that is similar to Cartesian routing [8]. Each node determines its own geographic position using a mechanism such as GPS [2]; positions consist of latitude and longitude. A node announces its presence, position, and velocity to its neighbors (other nodes within radio range) by broadcasting periodic HELLO packets. Each node maintains a table of its current neighbors' identities and geographic positions. The header of a packet destined for a particular node contains the destination's identity as well as its geographic position. When node needs to forward a packet toward location P, the node consults its neighbor table and chooses the neighbor closest to P. It then forwards the packet to that neighbor, which itself applies the same forwarding algorithm. The packet stops when it reaches the destination.

A packet may also reach a node that does not know about any nodes closer than itself to the ultimate destination. This dead-end indicates that there is a "hole" in the geographic distribution of nodes. In that case, the implementation described in this paper gives up and sends an error message to the packet's source node.

Recovering from dead-ends is possible using the same neighbor position table used in geographic forwarding. Karp and Kung propose GPSR [13], a geographic routing system that uses a planar subgraph of the wireless network's graph to route around holes. They simulate GPSR on mobile networks with 50–200 nodes, and show that it delivers more packets successfully with lower routing protocol overhead than DSR on networks with more than 50 nodes. Bose et al. independently demonstrate a loop-free method for routing packets around holes using only information local to each node. The method works only for *unit graphs*, in which two nodes can communicate directly in exactly the cases in which they are within some fixed distance of each other.

3.1 Effect of density

Geographic forwarding works best when nodes are dense enough that dead ends are not common. We present a simple evaluation of the effects of node density using the *ns* [7] network simulator. The simulated nodes have 2 Megabit per second IEEE 802.11 radios [6] with ranges of about 250 meters; each node transmits HELLO messages at 2 second intervals, and routing table entries expire after 4 seconds. Nodes move continuously at 10 m/s; each node moves by selecting a random destination, moving toward it, and selecting a new destination when it reaches the old one. Each node sends packets to three destination nodes selected at random; each conversation starts at a time selected randomly over the 300 second life of the simulation. A conversation involves sending 6 packets of 128 bytes each at quarter second intervals. Senders know the correct geographic positions of destinations.

Figure 1 is the result of simulations over a range of node densities. In each simulation, the nodes are placed at random in a 1 km^2 square. The graph reports the fraction of packets that were not delivered for each node density. In this scenario, geographic forwarding works well for more than 50 nodes per square kilometer. If 50 nodes are evenly placed in a 1 km^2 square, the inter-node spacing is $141 = 1000/\sqrt{50}$ meters, which is within radio range. More generally, the simulation results agree with a mathematical analysis of random nodes distributed throughout the unit square: one can prove that if the communication radius is r and the number of points exceeds $(6/r^2)\ln(6/r^2)$ per km^2, then dead ends are extremely unlikely to occur.

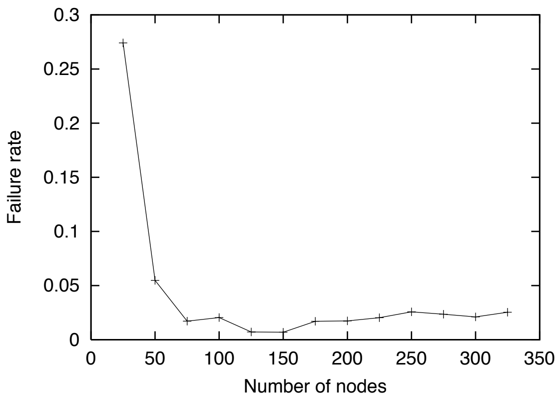


Fig. 1. Fraction of data packets unable to be delivered using geographic forwarding with a perfect location service, as a function of node density. The simulation area is 1 km^2.

4 The grid location service

Combining geographic forwarding with a mechanism for determining the location of a node implements the traditional network layer: any node can send packets to any other node. A trivial location service might consist of a statically positioned location server. Nodes would periodically update this server (using geographic forwarding to the server's well-known coordinates) with their current location. For a node **A** to contact node **B**, **A** queries the location server for **B**'s current location before using geographic forwarding to contact **B**.

Using a single location server has a number of problems. The centralized server is a single point of failure; it is unlikely to scale to a large number of mobile nodes; it can not allow multiple network partitions to each function normally in their own partition; and nodes near to each other gain no advantages—they must contact a potentially distant location server in order to communicate locally.

We introduce a distributed location service (GLS) that is designed to address these problems. GLS is fault-tolerant; there is no dependence on specially designated nodes. GLS scales to large numbers of nodes; our goal is to provide a service that scales to at least the size of a large metropolitan area. Finally, GLS operates effectively even for isolated pockets of nodes. A node should be able to determine the location of any node that it can reach with geographic forwarding. That is, a location lookup should not involve nodes that are too far "out of the way" of a straight line trip from the node performing the lookup to the node being looked up.

GLS is based on the idea that a node maintains its current location in a number of *location servers* distributed throughout the network. These location servers are not specially designated; each node acts as a location server on behalf of some other nodes. The location servers for a node are relatively dense near the node but sparse farther from node; this ensures that anyone near a destination can use a nearby location server to find the destination, while also limiting the number of location servers for each node. On the other hand long distance queries are not disproportionally penalized: query path lengths are proportional to data path lengths.

In order to spread uniformly the work of acting as location servers, GLS avoids techniques such as leader election or hierarchy to determine location server responsibility. These schemes place undue stress on the nodes unlucky enough to be elected as a leader or placed at higher levels in the hierarchy. Instead GLS allows a node **X** to select a set of location servers that, probabilistically, is unlike the set of servers selected by other nodes and does not change drastically as nodes enter or leave the network. Nodes searching for **X** are able to find **X**'s location servers using no prior knowledge beyond node **X**'s ID. This is accomplished by carrying out much the same protocol that **X** used to select its servers in the first place.

Our approach draws its intuition from *Consistent Hashing*, a technique developed to support hierarchical caching of web pages [12]. To avoid making a single node into the bottleneck of the hierarchical cache, that paper used a hash function to build a distinct hierarchy for each page, much as we use a distinct location service hierarchy for each target. Also like our paper, that paper used nested query radii to ensure that queries for a given page did not go to caches much farther away than the page itself.

GLS balances the location server work evenly across all the nodes if there is a random distribution of node IDs across the network. GLS ensures that nodes are allocated unique, random IDs by using a strong hash function to obtain an ID from a node's unique name. The name could be any uniquely allocated name, such as Internet host names, IP addresses, or MAC addresses. For purposes of discussing the GLS, a node's ID is more interesting than its original name, therefore when we refer to a node **A**, we are referring to the node whose name hashes to **A**.

4.1 Selecting and querying location servers

GLS provides for distributed location lookups by replicating the knowledge of a node's current location at a small subset of the network's nodes. This set of nodes is referred to as the node's location servers. A node **A** hoping to contact node **B** can query one of a number of other nodes that know **B**'s location. Of course, **A** must be able to contact the nodes that know **B**'s location. This means that **A**'s search for **B**'s location servers and **B**'s original recruitment of location servers ought to lead to the same servers. When **B** recruits location servers it uses the same information that **A** will have when searching for **B**'s location servers: **B**'s name and certain information that all nodes have at startup.

At startup, all nodes know the same global partitioning of the world into a hierarchy of grids with squares of increasing size, as shown in Figure 2. The smallest square is referred to as an order-1 square. Four order-1 squares make up an order-2 square, and so on. It is important that not every square made up of four order-n squares is also an order-$(n+1)$ square. Rather, to avoid overlap, a particular order-n square

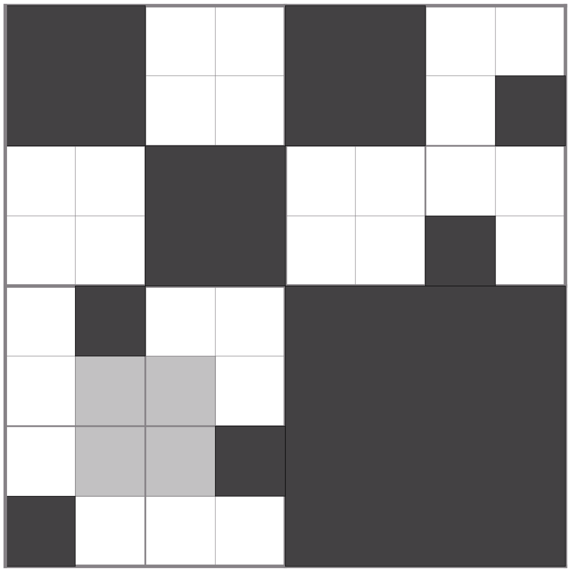


Fig. 2. A piece of the global partitioning of the world. A few example squares of various orders are shown with dark shading. The lightly shaded square is shown as an example of a 2x2 square which is *not* an order-2 square because of its location. An order-n square's lower left corner's coordinates must be of the form $(a2^{n-1}, b2^{n-1})$ for integers a,b.

is part of only one order-$(n+1)$ square, not four. This maintains an important invariant: a node is located in exactly one square of each size. This system of increasing square sizes provides a context in which a node selects fewer and fewer location servers at greater distances. Our choice of a grid-based partition is somewhat arbitrary; any other balanced hierarchical partition of the space can be used instead.

Consider how **B** determines which nodes to update with its changing location, using its ID and the predetermined grid hierarchy. **B** knows that other nodes will want to locate it, but that they will have little knowledge beyond **B**'s ID. **B**'s strategy is to recruit nodes with IDs "close" to its own ID to serve as its location servers. We define the node *closest* to **B** in ID space to be the node with the *least ID greater than* **B**. The ID space is considered to be circular, 2 is closer to 17 than 7 is to 17.

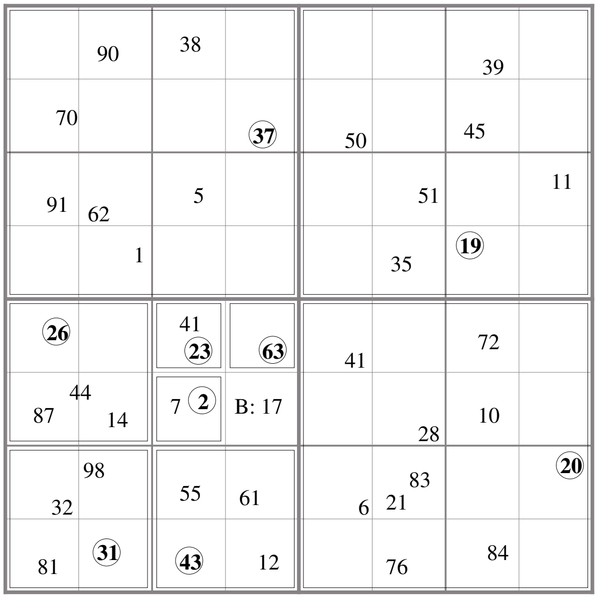


Fig. 3. The inset squares are regions in which **B** will seek a location server. The nodes that become **B**'s location servers are circled and shown in bold.

If we consider the tree corresponding to the grid decomposition, a node selects location servers in each *sibling* of a square that contains the node. The exact details of the selection are best understood with an example (see Figure 3). A node chooses three location servers for each level of the grid hierarchy. For example, in the figure, **B** recruits three servers in order-1 squares, three servers in order-2 squares, and three servers in order-3 squares. In each of the three order-1 squares that, along with **B**'s own order-1 square, make up an order-2 square, **B** chooses the node closest to itself in ID space as a server. The same location server selection process occurs in higher order squares. In the three order-2 squares that combine with **B**'s order-2 square to make an order-3 square, **B** selects 26, 31, and 43 as location servers.

Figure 4 shows the state of a Grid network once all nodes have provided their coordinates to the nodes that will act as their location servers. With the complete network state as reference, we can return to the problem of how **A** finds the location of **B**.

To perform a location query, **A** sends a request (using geographic forwarding) to the least node greater than or equal to **B** for which **A** has location information. That node forwards the query in the same way, and so on. Eventually, the query will reach a location server of **B** which will foward the query to **B** itself. Since the query contains **A**'s location, **B** can respond directly using geographic forwarding. The location query is forwarded all the way to **B** so that **B** can respond with its latest location.

For illustrative purposes we have ignored an important bootstrapping issue. We have assumed that nodes select their location servers appropriately and send their coordinates to them. This appears to assume that a node can scan an entire square (of arbitrary size) and choose the appropriate node to act as its server. In fact, nodes route update packets to their location servers without knowing their identities. Assume that a node **B** wishes to recruit a location server in some order-n square. **B** sends a packet, using geographic forwarding, to that square. The first node **L** in the square that receives the packet begins a location update process that closely resembles a query for **B**'s location; but this update will actually carry the current location of **B** along with it. As we will demonstrate below, the update will arrive at the least node greater than **B** before leaving the order-n square containing **L**. This is exactly the appropriate destination for the location update to go to; the final destination node simply records **B**'s current location and becomes a location server for **B**.

The only requirement for **B** to distribute its location to the appropriate server in an order-n square is that the nodes contained in the square have already distributed their locations *throughout that square.* If we imagine an entire Grid system being turned on at the same time, order-1 squares would exchange information using the local routing protocol, then nodes could recruit their order-2 location servers, then order-3, etc. Once the order-n location servers are operating, there is sufficient routing capability to set up the order-$(n+1)$ location servers.

4.2 Efficiency analysis

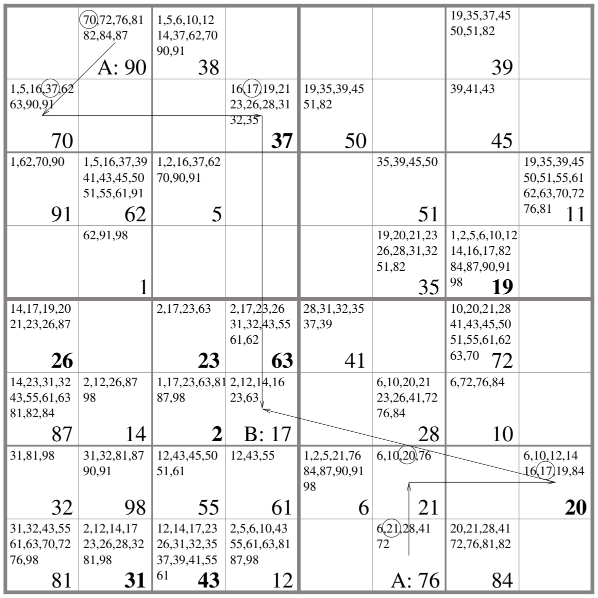


Fig. 4. An entire network's location server organization. Each node is shown with the list of nodes for which it has up to date location information; **B**'s location servers are shown in bold. Two possible queries by **A** for **B**'s location are shown.

When nodes are not moving, the number of steps taken by a location query from **A** to **B** is no more than the order of the smallest square in which **A** and **B** are colocated. A location query *step* is distinct from a single *hop* in the geographic forwarding layer; indeed, each location query step is likely to require several geographic forwarding hops. In Figure 4, the entire diagram is an order-4 square. Therefore all queries can be performed in no more than four location query steps.

At each step, a query makes its way to the best (closest in ID space to the destination) node at successively higher levels in the grid hierarchy. At the start, the query is forwarded to the best node in the local order-1 square using the local routing protocol. From this point on, each step moves the query to the best node in the next larger containing square; when that next larger square contains the destination node, the best node (closest to the destination ID) must be the destination itself. Thus the query's next step is to the destination. This behavior not only limits the number of steps needed to satisfy a query, it also bounds the geographic region in which the query will propagate. Because the query proceeds into larger and larger squares *that still contain the source*, the query will stay inside the smallest square containing the source and the destination.

To understand why each step brings the query to the best node in a larger square, we will first consider the query from node **A** (76) for the address of **B** (17), shown starting in the lower right of Figure 4. Our abbreviated topology has no more than one node per square, so the query trivially begins at the best node, itself, in its order-1 square. The query moves to the best node (21) in **A**'s order-2 square, because 76 happens to know the positions for all the nodes in its order-2 square. This is an artifact of our sparse layout, so the next step tells the important story: why 21 knows the location of the best node in the next higher order square.

Recall that 21 is the best node in its order-2 square. This guarantees that no nodes in that square have IDs between 17 and 21. Now, consider a node **X** somewhere in node 21's order-3 square, but not in 21's order-2 square. Recall that **X** had to choose a location server in node 21's order-2 square. If **X**'s ID is between 17 and 21 then **X** *must* have chosen node 21 as a location server since there are no better nodes in node 21's order-2 square. Thus, node 21 knows about *all* nodes in its order-3 square that lie between 17 and itself, including the minimum such node. In this case, that node is 20. At the next step, node 20 must know about all nodes in the order-4 square between 17 and itself. Since nodes 20 and 17 share the same order-4 square (the entire figure), node 20 knows about node 17, and the query is finished.

The above example demonstrates why node 21 knew node 20's location and was therefore able to move the query from the best node in its order-2 square to the best node in its order-3 square. One may wonder, however, why node 21 does not know about some other node whose ID is between 17 and 20, and which lies at a distant location. This would be undesirable as node 21 would then forward the packet far away simply because, for example, it might know the location of node 19. But this cannot happen because node 20 acts as a shield for node 21 during location server selection. That is, for any node outside of the lower right quadrant of figure 4, node 21 is guaranteed *not* to be the best choice for location server; node 20 will always be preferable. In addition, because every location query is labelled with its source, intermediate query steps know what level of the hierarchy the query is currently in, and can refrain from sending queries too far away.

Having built an intuition, we now give an inductive proof that a query needs no more than n location query steps to reach its destination when the source and destination are colocated in an order-n square. Furthermore, the query never leaves the order-n square in which it starts. We assume, without loss of generality, that the destination node's ID is 0. We then proceed inductively to prove the following equivalent claim: in n or fewer location query steps, a query reaches the node with the lowest ID (i.e closest to 0) in the order-n square containing the source. Since the destination is node 0, when the query reaches the order-n square that contains both the source and the destination nodes, it must reach the destination.

Base case (order-1 square): The query begins at a node **X**. Node **X** may or may not be the node with the lowest ID in its order-1 square. If so, the query trivially reaches the lowest node in the order-1 square after zero location query steps. If **X** is not the node with the lowest ID, then **X** will know the location of the node with the lowest ID in the order-1 square, **Y**, through the local routing protocol. Node **X** will not know of any other nodes with IDs lower than **Y**. Any such node would not have selected **X** as a location server as **Y** would always have been the better choice. Therefore the lowest node that **X** is aware of is **Y** and the query will be forwarded there in one location query step.

Inductive step (order-$(n+1)$ *square):* We claim that if a query is at the node **X** with the lowest ID in its order-n square, then **X** will route the query to the node **Y** with the lowest ID in its order-$(n+1)$ square with one or zero location query steps. If **X** has the lowest node ID in the order-$(n+1)$ square, then our claim is trivially true. If not, **X** will know the coordinates of **Y** and *will not* know the coordinates of any node lower than **Y** outside the order-$(n+1)$ square. Node **X** will know the coordinates of **Y** because **Y** will have selected **X** as a location server. Node **Y** must have selected a location server in **X**'s order-n square because **Y**'s order-n square is a part of the same order-$(n+1)$ square as **X**'s. Node **Y** must have selected **X** because **X** is the lowest node in its square that is greater than **Y**. Node **X** will not know the location of any node lower than **Y** outside of its order-$(n+1)$ square because when any such node sought a location server in **X**'s order-$(n+1)$ square, Node **Y** was the better choice. Therefore the lowest node that **X** is aware of is **Y** and the query will be forwarded there in one location query step. □

It is important to remember however, that this proof applies only to a static network. Additional techniques, described in Section 5, help Grid to deal with the problems cereated by mobility. These sections describe Grid's approach to keeping location servers up to date in the face of node motion and Grid's recovery techniques when, despite updates, location information is found to be out of date.

5 Implementation

This section describes the details of the geographic forwarding and GLS protocols.

5.1 Geographic forwarding

The geographic forwarding layer uses a two hop distance vector protocol. This helps alleviate holes in the topology and ensures that each node knows the location of all nodes in its order-1 square. Each node maintains a table of immediate neighbors as well as each neighbor's neighbors. Each entry in the table includes the node's ID, location, speed, and a timestamp. Each node periodically broadcasts a list of all neighbors it can reach in one hop, using a HELLO message. When a node receives a HELLO message, it updates its local routing table with the HELLO message information. Using this protocol nodes may learn about two hop neighbors—nodes that cannot be reached directly, but can be reached in two hops via the neighbor that sent the HELLO message. The routing table is also updated every time a node receives a packet, using the packet's last hop information.

Each entry in the neighbor table expires after a fixed timeout. However, when an entry expires, the node estimates the neighbor's current position using its recorded speed. If it would likely still be in range, the entry may still be used for forwarding, but it is not reported as a neighbor in further HELLO messages. This special treatment is justified by two properties of the 802.11 MAC layer. First, broadcast packets are more likely to be lost in the face of congestion than unicast packets. Thus it is not unusual to miss HELLO messages from a node that is still nearby. Second, unicast transmissions are acknowledged. If the neighbor has actually moved away, the transmitting node will be notified when it attempts to forward packets through the missing node. The invalid neighbor entry is then removed immediately and a new forwarding path is chosen.

HELLO
Source ID
Source location
Source speed
Neighbor list: IDs and locations
Forwarding pointers

Fig. 5. HELLO packet fields.

To select a next hop, nodes first choose a set of nodes from all nodes in their neighbor table. This set consists of the best nodes to move the packet to, as defined by the shortest distance to the destination from the candidate nodes. All nodes whose distances to the destination are nearly equal are considered in this set. Call this set B. If B contains any single-hop neighbors, remove double-hop neighbors from B. A node, **X**, is then chosen at random from B. If **X** is a single-hop neighbor, the packet is forwarded to **X**, otherwise, since **X** may be reachable from any number of single hop neighbors, the best such neighbor is chosen and the packet is forwarded to that node. If the transmission fails, the chosen node is removed from consideration and the packet is reprocessed, starting with the original B (with **X** removed if it was a single-hop neighbor).

5.2 Updating location information

GLS maintains two tables in each node. The location *table* holds the node's portion of the distributed location database; each entry consists of a node ID and that node's geographic location. The location *cache* holds location information that the node learns by looking at update packets it forwards. A node only uses the cache when originating data packets. Because each node uses the neighbor table maintained by the geographic forwarding layer to learn about other nodes in its order-1 square, the node does not need to send normal GLS updates within its order-1 square.

As a node moves, it must update its location servers. Nodes avoid generating excessive amounts of update traffic by linking their location update rates to their distance traveled. A node updates its order-2 location servers every time it moves a particular threshold distance d since sending the last update; the node updates its order-3 servers after each movement of $2d$. In general, a node updates its order-i servers after each movement of $2^{i-2}d$. This means that a node sends out updates at a rate proportional to its speed and that updates are sent to distant servers less often than to local servers. In addition, nodes send location updates at a low rate even when stationary.

Location update packets (see Figure 6) include a timeout value that corresponds to the periodic update interval, allowing the servers to invalidate entries shortly after a new entry is expected. The time at which the location update packet is generated is also included in the update packet so that the freshness of location information obtained from different nodes for the same destination can be compared. GPS receivers can provide every node in the network with closely synchronized time.

LOCATION UPDATE
Source ID
Source location
Source timestamp
Update destination square
Update timeout
Next location server's ID
Next location server's location

Fig. 6. GLS update packet fields.

When forwarding an update, a node adds the update's contents to its location cache. The node associates a relatively short timeout value with the cached entries regardless of the recommended timeout value carried in the update packet.

Nodes piggyback their location information on data packets, so that two nodes who are communicating always know how to reach each other. In the case of one-way communication, nodes also periodically send their position information directly to nodes who are sending them data.

5.3 Performing queries

When a node **S** originates a data packet for destination **D**, it first checks its location cache and location table to find **D**'s location. If it finds an entry for **D**, it sends the packet to **D**'s recorded location using geographic forwarding. Otherwise, **S** initiates a location query for **D** using the GLS. GLS will eventually deliver the query packet (Figure 7) to **D**, which will geographically route a response to **S** that includes **D**'s current location.

If **S** had to initiate a GLS query, it stores the data packet in a send buffer while it waits for the reply from **D**. Node **S** reinitiates the query periodically if it gets no reply, using binary exponential backoff to increase the timeout intervals.

LOCATION QUERY
Source ID
Source location
Ultimate target ID
Next location server's ID
Next location server's location
Timestamp from previous server's database

Fig. 7. GLS query packet fields.

5.4 Location query failures

A location query may fail for two reasons. First, a node may receive a query packet for **D**, and not know the location of any node with an ID closer to **D** than itself. This type of failure is relatively uncommon. It occurs when a location server has not recently received a location update for a node it should know about. Because the server has timed out the node's previous update, it has no way to forward the query packet. There are ways to alleviate these failures, such as using stale location data in a last ditch effort to forward a query packet if the query would otherwise fail. The second type of query failure occurs when a location server forwards a packet to the next closest node's square, but the node is no longer in that square (that is, the location information at the previous location server is out of date). Because this failure mode is more common, Grid contains a specialized mechanism to alleviate the problem.

Consider a node **D** that has recently moved from the order-1 square s_1 to the order-1 square s_2. Node **D**'s location servers, particularly those that are far away, will think that **D** is in s_1 until **D**'s next updates reach them. To cope with this, **D** leaves a "forwarding pointer" in s_1 indicating that it has moved to s_2. When a packet arrives in s_1 for **D**, it can be correctly sent on by following the forwarding pointer. **D** broadcasts its forwarding pointer to all nodes in s_1 when leaving. Conceptually, we can think of the forwarding pointers as being located in the *square* s_1 rather than at any particular node. Therefore, all nodes that move into s_1 should pick up the forwarding pointers associated with s_1, and when nodes leave s_1, they should forget the corresponding forwarding pointers. To propagate forwarding pointers to all nodes in the order-1 square and keep all newcomers to the square updated, a randomly chosen subset of the forwarding pointers stored at a node (up to five in our simulation implementation) is piggybacked on the node's periodic HELLO messages. Upon hearing a HELLO message, a node adds each forwarding pointer in that message to its own collection of forwarding pointers, but only if the pointer's original broadcaster was in the same square as the node. In this way, forwarding pointer information is effectively and efficiently spread to every node in the square. With this propagation mechanism, even if all the nodes that originally received **D**'s forwarding pointer were to leave the square themselves, the information would still be available in the square.

6 Performance analysis

This section presents simulation results for GLS that show how well it scales. Good scaling means that the amount of work each node performs does not rise quickly as a function of the total number of nodes. We use two metrics for work: the number of location database entries each node must store, and the number of protocol packets each node must originate or forward in order to route a given workload. The simulations show that these costs scale well with the number of nodes.

Mobility increases the work required in two ways. First, a node that moves must update its location servers. Second, if a node has moved recently, some nodes may retain out-of-date location information for it; this will cause queries for the moved node to travel farther than necessary, or to fail and need to be resent. Handling mobility requires a tradeoff between the bandwidth used by location updates and the bandwidth available for data. If a moving node sends updates aggressively, other nodes are more likely to be able to find it. However, the updates consume bandwidth in competition with data. Worse, a very aggressive update policy may cause enough congestion that updates themselves are dropped. At the other extreme, a node could send updates infrequently even when moving quickly, increasing the amount of bandwidth available to data. However, that bandwidth is not useful if the success rate of location query becomes low because of inaccurate location information. The simulations show that Grid can achieve a reasonable tradeoff for the choice of update rate.

6.1 Simulation scenario

The simulations use CMU's wireless extensions [9] for the *ns* [7] simulator. The nodes use the IEEE 802.11 radio and MAC model provided by the CMU extensions; each radio's range is approximately a disc with a 250 meter radius. The simulations without data traffic use 1 Megabit per second radios; the simulations with data traffic use 2 Megabits per second radios. Each simulation runs for 300 simulated seconds. Each data point presented is an average of five simulation runs.

The nodes are placed at uniformly random locations in a square universe. The size of each simulation's universe is chosen to maintain an average node density of around 100 nodes per square kilometer. One reason for this choice is that we intend the system to be used over relatively large areas such as a campus or city, rather than in concentrated locations such as a conference hall. Another reason is that we expect any deployed system to use radios that allow the power level to be decreased in areas with high node density. The GLS order-1 square is 250 meters on a side. For a network of 600 nodes, which is the biggest simulation we have done, the grid hierarchy goes up to order-5 in a square universe 2900m on a side.

Each node moves using a "random waypoint" model [5]. The node chooses a random destination and moves toward it with a constant speed chosen uniformly between zero and a maximum speed (10 m/s unless noted otherwise). When the node reaches the destination, it chooses a new destination and begins moving toward it immediately. These simulations do not involve a pause time.

6.2 GLS results

The results in this section involve only GLS (and geographic forwarding), without any data traffic. The default simulation parameters for this section are an 802.11 radio bandwidth of 1 Megabit per second, and a communication model in which each node initiates an average of 15 location queries to random destinations over the course of the 300 second simulation, starting at 30 seconds. The location update threshold distance is an important parameter that may need to be tuned. For this reason we present results for three values of the threshold: 100, 150, and 200 meters.

Figure 8 shows the success rate for GLS location queries, as a function of the total number of nodes. Queries are not retransmitted, so a success means a success on the first try. As mentioned earlier, most failures are due to either location information invalidated by node motion or nodes not being correctly updated because of delayed or lost

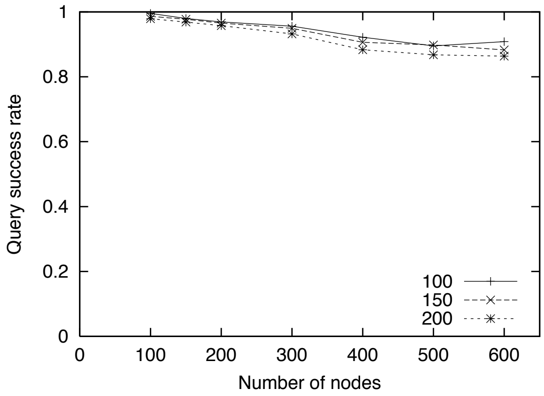


Fig. 8. GLS query success rate as a function of the total number of nodes. The nodes move at speeds up to 10 m/s (about 22 miles per hour). Each line corresponds to a different movement update threshold.

location updates. The success rate for data sent after a successful query would be much higher than indicated here because the endpoints of a connection directly inform each other of their movements.

Figure 9 shows the average number of Grid protocol packets forwarded and originated per second per node as a function of the total number of nodes. Grid generates three types of protocol packets: HELLO packets that are generated every two seconds but not forwarded, location update packets that are also periodic but require forwarding, and location query and reply packets that also require forwarding. As location updates are generated by nodes as they move, the results depend on node speeds; the simulated nodes move at speeds uniformly distributed between 0 and 10 m/s. Figure 9 is generated from the same simulations that produced Figure 8. The graph shows that Grid imposes a modest protocol traffic load as the network size grows.

Figure 10 shows how the distance that query packets travel compares with the actual distance in hops between the source and the destination. We record the total number of geographical forwarding hops (for all query steps) that each query takes, as well as how many hops the reply takes. Since query replies are sent directly to the query source using geographic forwarding, the reply return path indicates the geographical forwarding hop distance between the source and destination. We averaged the query hop lengths for all queries with a given response hop length. The graph shows that on average, query packets only travel about 6 hops more than the geographical forwarding

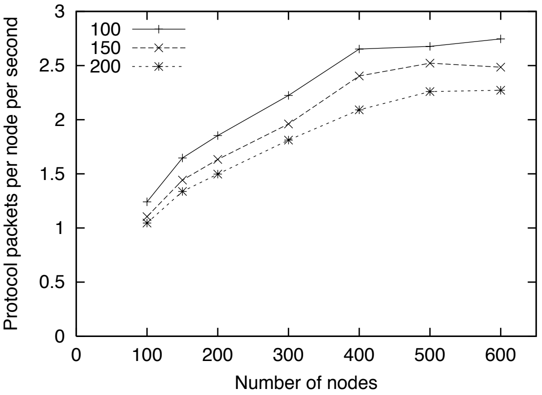


Fig. 9. Average number of Grid protocol packets forwarded and originated per second by each node as a function of the total number of nodes. Nodes move at speeds up to 10 m/s.

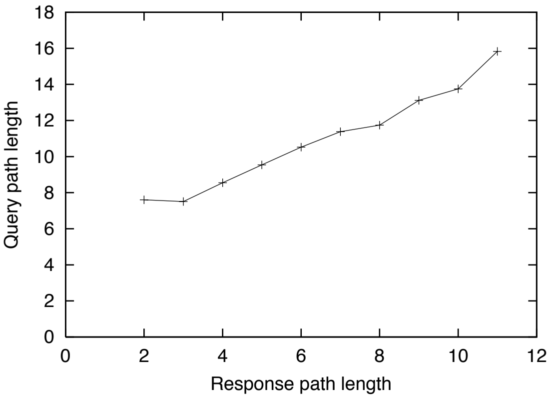


Fig. 10. Average query path length (in hops) as a function of the query reply path length, for 300 nodes moving up to 10 m/s.

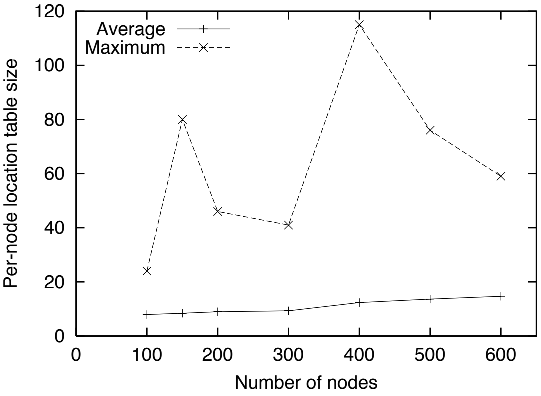


Fig. 11. Average and maximum per-node location database size (number of entries) as a function of the total number of nodes. The nodes move at speeds up to 10 m/s.

route between nodes. Also, the distance traveled by a query between two nodes is proportional to the actual distance between those nodes. Our simulation agrees with a theoretical analysis that proves that with a sufficiently dense uniform distribution, the number of hops traveled by the query is proportional to the distance to the destination. The simulation involves 300 nodes moving at speeds up to 10 m/s, with a location update threshold of 200 meters.

Figure 11 shows the effect of the total number of nodes on the size of each node's GLS location table. The plots include both the average and maximum location table size over all nodes. The spikes at 150 and 400 nodes occur because the simulated area does not exactly fill a hierarchy, causing the database load to be distributed unevenly. At these points, the maximum database size is larger because the squares that extend across the edge of the simulated area contain relatively few nodes; these nodes must store more than their fair share of location database entries. On the other hand, the average table size grows very slowly with the network size.

This highlights a problem that may arise in practice when nodes are not uniformly distributed. A small number of nodes in a high-level square may end up responsible for tracking the locations of a large number of nodes in sibling squares. This would require large amounts of space in these few nodes.

Figure 12 shows the effect of node movement speed on the GLS query success rate, for 100 nodes. As nodes move faster, their location

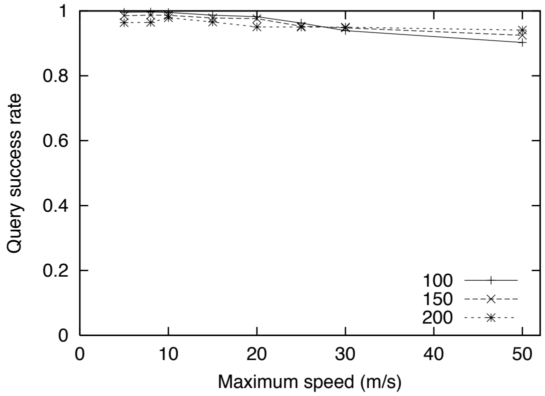


Fig. 12. GLS query success rate as a function of maximum node speed in a network of 100 nodes. 50 m/s is about 110 mph.

servers are more likely to be out of date. On the other hand, the nodes also generate updates faster. The net effect is that the query success rate is relatively insensitive to node speed, however, the update traffic grows as nodes move faster.

Figure 13 shows the effect of nodes turning on and off. Some nodes are always on, while the rest alternate being on and off for intervals uniformly distributed from 0 to 120 and 0 to 60 seconds, respectively. As we are simulating node crashes, nodes do not do anything special before turning off; they simply lose all their location table data. In practice, if a node was manually turned off, it would be appropriate to first redistribute its location table to get better performance. Each point in the graph represents a simulation in which a different fraction of nodes are always on. The simulations involve 100 nodes, each moving with a maximum speed of 10 m/s. The statistics are limited to queries addressed to nodes that are turned on; no queries are generated to nodes that are off as these queries will always fail. When a node turns off, a part of the distributed location database is lost; when a node turns on, it will not be able to participate correctly in the update and query protocol for a while. The graph shows that even a great deal of instability does not have a disastrous effect, and that the query success rate degrades gracefully as nodes turn on and off.

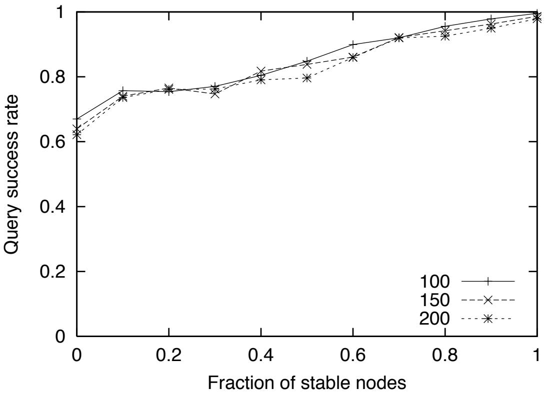


Fig. 13. The effect of turning off nodes on the query success rate. The X axis indicates the fraction of nodes that are always on; the remaining nodes cycle on and off for random periods up to 120 and 60 seconds, respectively. The simulations all involve 100 nodes moving at speeds up to 10 m/s.

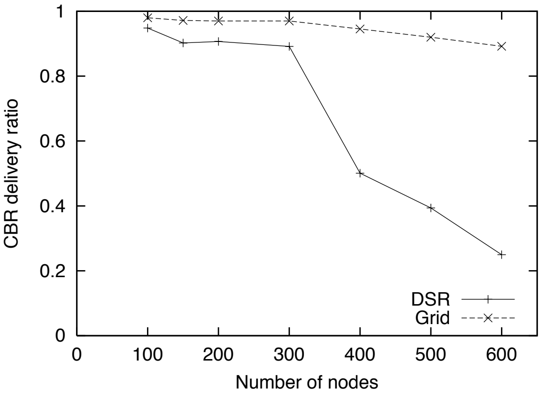


Fig. 14. The fraction of data packets that are successfully delivered in simulations for increasing numbers of nodes. The nodes move with a maximum speed of 10 m/s.

6.3 Data traffic

The simulations in this section measure Grid's behavior when forwarding data traffic. The 802.11 radio bandwidth is 2 Megabits per second, and the location update threshold distance is 200 meters. The data traffic is generated by a number of constant bit rate connections equal to half the number of nodes. No node is a source in more than one connection and no node is a destination in more than three connections. For each connection four 128-byte data packets are sent per second for 20 seconds. Connections are initiated at random times between 30 and 280 seconds into the simulation. For purposes of comparison we include results for the DSR [11] protocol. This may not be a fair comparison since DSR is optimized for relatively small networks [4].

Figure 14 shows the fraction of data packets successfully delivered. Most of the data packets that Grid fails to deliver are due to GLS query failures; these packets never leave the source. Once Grid finds the location of a destination, data losses are unlikely, since geographic forwarding adapts well to the motion of intermediate nodes. Below 400 nodes, most of the DSR losses are due to broken source routes; at 400 nodes and above, losses are mainly due to flooding-induced congestion. Grid does a better job than DSR over the whole range of numbers of nodes, especially for large networks.

Figure 15 shows the message overhead of the Grid and DSR protocols. Only protocol packets are included. In the case of Grid, these are HELLO, GLS update, and GLS query and reply packets. In the case of DSR, these are route request, reply, cached reply packets etc. DSR produces less protocol overhead for small networks, while Grid produces less overhead for large networks. At 400 nodes and above, DSR suffers from network congestion. Almost half of the route reply and cache reply messages are dropped due to congestion which causes DSR to inject even more route requests into the network. Also, as the network grows larger and congestion builds up, the source route is more vulnerable to failure which will also induce DSR source nodes to send more route request packets. DSR's overhead drops at 600 nodes because it could not send much more packets in the presence of congestion. We present overhead in terms of packets rather than bytes because medium acquisition overhead dominates actual packet transmission in 802.11, particularly for the small packets used by Grid.

7 Future work

One area of the GLS protocol that could be improved is the handling of node mobility. Accurate movement models may allow us to integrate movement prediction into the GLS protocol. Our current system makes little effort to predict the movement of nodes over long time periods because our movement model is randomized, but in the real world a node may not need to update a location server as often if its velocity is constant or predictable.

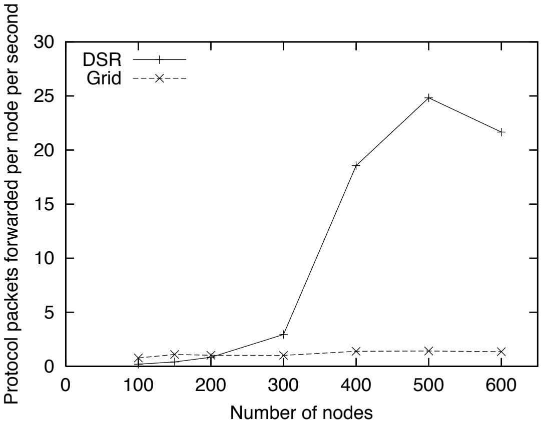


Fig. 15. The number of all protocol packets forwarded per node per second as a function of the total number of nodes. No data packets are included. The nodes move with a maximum speed of 10 m/s.

Currently the GLS protocol makes little effort to proactively correct out-of-date information when, for instance, a node crosses a grid boundary line. Proactive updates may reduce the incidence of query failures. However, the tradeoff is obvious—care must be taken not to consume too much bandwidth with the updates. An alternate strategy to address the same problem is to place less trust in locations obtained from distant location servers. Rather than trust a distant location server to pinpoint the order-1 square in which a node is located, a query could be moved to, for instance, the surrounding order-3 square. There the query can be restarted with the fresher information available in that square

Another potential area of improvement is adapting to node density. If an order-1 square becomes too crowded, each node will get less bandwidth from the shared radio spectrum, and each node will have to work harder to keep its neighbor table up to date. Radios with variable power levels would help alleviate this problem by changing the effective density of nodes within radio range. In addition, each square in the GLS may make a local decision about how finely to sub-divide itself; distant areas need not agree on the size of the order-1 square.

Finally, as we noted earlier, the choice of a grid based system is somewhat arbitrary. In fact, certain partitioning schemes offer the possibility of better scaling. The number of location servers that a node must recruit is equal to the number of neighbors per level in the geographic hierarchy multiplied by the number of levels in the hierarchy.

For a grid based system, this means that a node must maintain $3\log_4 n$ servers in a network that is n times the size of the coverage area of a single radio. It is possible, however, to split the world in half at each level, rather than in fourths, by using rectangles with an aspect ratio of $1/\sqrt{2}$. At successive levels, each such rectangle may be divided into two such rectangles. This leads to a network in which nodes must recruit only $\log_2 n$ location servers, or 2/3 the number of servers needed in a grid based approach.

8 Conclusions

Wireless technology has the potential to dramatically simplify the deployment of data networks. For the most part this potential has not been fulfilled: most wireless networks use costly wired infrastructure for all but the final hop. Ad hoc networks can fulfill this potential because they are easy to deploy: they require no infrastructure and configure themselves automatically. But previous ad hoc techniques do not usually scale well to large networks.

We have presented a mobile ad hoc networking protocol with significantly better scaling properties than previous protocols. Although somewhat complicated to understand, our protocol is very simple to implement. In many ways the two facets of our system, geographic forwarding and the GLS, operate in fundamentally similar ways. Geographic forwarding moves packets along paths that bring them closer to the destination in physical space, only reasoning about nodes with nearby locations at each step along the path. GLS moves packets along paths that bring them closer to the destination in ID space, using only information about nodes with nearby IDs at each step along the path. Both mechanisms are scalable because they only need local information in their respective spaces.

9 Acknowledgments

We would like to thank Frans Kaashoek for his valuable input which improved the paper. We would also like to thank Yan Zhang for her discussions and help with running simulations, and Brad Karp for reading a draft.

References

1. October 1998.
2. USCG Navigation Center GPS page, January 2000. `http://www.navcen.uscg.mil/gps/default.html`.
3. Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). In *Proc. ACM/IEEE MobiCom* [1], pages 76–84.
4. Josh Broch, David Johnson, and David Maltz. The Dynamic Source Routing protocol for mobile ad hoc networks. Internet draft (work in progress), Internet Engineering Task Force, October 1999. `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-03.txt`.
5. Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. ACM/IEEE MobiCom* [1], pages 85–97.
6. IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. New York, New York, 1997. IEEE Std. 802.11–1997.
7. Kevin Fall and Kannan Varadhan. *ns* notes and documentation. Technical report, UC Berkeley, LBL, USC/ISI, and Xerox PARC, November 1997. `http://www-mash.berkeley.edu/ns`.
8. Gregory G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. ISI/RR-87-180, ISI, March 1987.
9. CMU Monarch Group. CMU Monarch extensions to *ns*. `http://www.monarch.cs.cmu.edu/`.
10. Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proc. ACM/IEEE MobiCom*, pages 195–206, August 1999.
11. David B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 158–163, December 1994.
12. D. R. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. 29th ACM Symposium on Theory of Computing*, pages 654–663, May 1997.
13. Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. ACM/IEEE MobiCom*, August 2000.
14. Young-Bae Ko and Vaidya Nitin H. Location-Aided Routing (LAR) in mobile ad hoc networks. In *Proc. ACM/IEEE MobiCom* [1], pages 66–75.
15. Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. IEEE Infocom*, pages 1405–1413, April 1997.
16. Charles Perkins, Elizabeth Royer, and Samir R. Das. Ad hoc On demand Distance Vecor (AODV) routing. Internet draft

(work in progress), Internet Engineering Task Force, October 1999. `http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-04.txt.`

17. Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proc. ACM SIGCOMM Conference (SIGCOMM '94)*, pages 234–244, August 1993.
18. Paul F. Tsuchiya. The Landmark hierarchy: A new hierarchy for routing in very large networks. In *Proc. ACM SIGCOMM Conference (SIGCOMM '88)*, pages 35–42, August 1988.
19. Paul F. Tsuchiya. Landmark routing: Architecture, algorithms, and issues. MTR-87W00174, MITRE, May 1988.

20th ICDS, pp. 344–355, April 2000
(Full version: MIT LCS Technical
Report MIT-LCS-TR-794).

A Client-Server Approach to Virtually Synchronous Group Multicast: Specifications and Algorithms

Idit Keidar and Roger Khazan

MIT Laboratory for Computer Science
{`idish, roger`}`@theory.lcs.mit.edu`

Abstract. This paper presents a formal design for a novel group multicast service that provides virtually synchronous semantics in asynchronous fault-prone environments. The design employs a client-server architecture in which group membership is maintained not by every process but only by dedicated membership servers, while virtually synchronous group multicast is implemented by service end-points running at the clients. Specifically, the paper defines service semantics for the client-server interface, that is, for the group membership service. The paper then specifies virtually synchronous semantics for the new group multicast service, as a collection of commonly used safety and liveness properties. Finally, the paper presents new algorithms that use the defined group membership service to implement the specified properties. The algorithm that provides the complete virtually synchronous semantics executes in a single message round in parallel with the membership service's agreement on views, and is therefore more efficient than previously suggested algorithms providing such semantics.

1 Introduction

Group communication systems are powerful building blocks that facilitate the development of fault-tolerant distributed applications (see [1, 20, 5] for discussion of the utility of group communication systems). Group communication provides the notion of *group abstraction*, which allows processes to be easily organized in multicast groups. Group communication systems typically integrate two types of services: group membership and reliable group multicast. The membership service maintains a listing of the currently active and connected group members and delivers this information to its clients whenever it changes. The output of the membership service is called a *view*. Reliable multicast services that deliver messages to the current view members complement the membership service. In this paper, we present a novel group multicast service.

Group communication systems usually run in asynchronous fault-prone environments. In such environments, group communication systems generally provide some variant of *virtual synchrony* semantics which synchronize membership notifications with regular messages and thus simulate a "benign" world in which message delivery is reliable within the set of connected processes. Such semantics are especially useful for constructing fault-tolerant applications that maintain consistent replicated state of some sort (e.g., [7, 15]). The key aspect of virtual synchrony is the semantics of interleaving of message send and delivery events with view delivery events. In order to reason about this interleaving, we associate message send and delivery events with views: we say that an event `e` occurs at a process `p` *in view* `v` if `v` was the last view delivered to `p` before `e`, or a default initial view $\mathtt{v_p}$ if no such view was delivered.

Many variants of virtual synchrony semantics have been suggested (e.g., [17, 8, 20, 18, 7]). A key property specified by nearly all of these (e.g., [17, 8, 20, 18]) is that processes moving together from a view `v` to another view $\mathtt{v}'$ deliver the same messages in `v`. This property allows applications to avoid costly re-synchronization following certain view changes. Our service specification includes this property, as well as several additional safety and liveness properties. We present our specifications in Section 4.

During the period in which the group communication service is attempting to reach agreement on a view, processes may attempt to join/re-join. In such cases, previously suggested virtual synchrony algorithms, e.g., [8, 18], can have the current invocation of the membership and virtual synchrony proceed to termination without adding the joining processes, and then immediately start an attempt to add them. This strategy results in overhead (e.g., increased network load) because applications react to such outdated views just as they do to any other view, e.g., by re-synchronizing with the new members. Moreover, this strategy precludes situations when applications may rely on virtual synchrony to avoid the costly re-synchronizations all together. For example, consider a transient failure when a process `p` is unsuspected right after an attempt to remove `p` from the membership has started. Existing algorithms typically deliver a view excluding `p` and then re-invoke the algorithm to allow `p` to re-join. Since `p` does not move into the resulting view from the same view as the rest of the processes, these processes cannot rely on the key property of virtual synchrony to avoid re-synchronizing with `p`. In contrast, our algorithm never delivers views that reflect a membership that is already known to be out of date.

Traditionally, virtual synchrony semantics were implemented by algorithms that were integrated with group membership algorithms (e.g., in [8, 9, 2]). In contrast, our group multicast service is designed for a client-server architecture in which a small set of dedicated membership servers maintains client membership information (i.e., which clients are members of which group). This architecture was designed to provide scalable membership services in wide area networks (cf. [3]). Our virtual synchrony algorithm acts as the client of an *external* membership service.

Introducing the client-server design poses a major challenge: One has to define an interface by which a membership server interacts with its clients, in a way that would allow for simple and efficient implementations of both group membership (by the membership servers), and virtual synchrony (by service end-points at the clients). Such an interface has to provide sufficient level of synchronization to allow the virtual synchrony algorithm to reach agreement upon the set of messages delivered in the old view in *parallel* with the servers' agreement on views. At the same time, the virtual synchrony algorithm should avoid imposing limitations on the membership's choice of views (as explained above). In addition, one has to try to minimize the communication overhead induced by the client-server interaction.

We have designed an interface that addresses the challenges above. Our interface consists of two types of messages sent from membership servers to their clients: When a server engages in a view change, it sends its clients a `start_change` message. Each `start_change` message contains a locally unique identifier. This identifier is *not* globally agreed upon: `start_change` messages sent to different processes can contain different identifiers. Once the server agrees upon the new view with the other servers, it sends a `view` message to its clients. The view contains information that maps clients to the last `start_change` identifiers they received before receiving this view. A similar view structure is suggested in [18], for the purpose of not having concurrent views intersect. The servers do not need to hear from their clients in order to complete the algorithm.

Our interface allows for straightforward and efficient implementations of both membership and virtual synchrony. The algorithm we present in Section 5 exploits this interface to achieve virtual synchrony in a single round. We have implemented this algorithm (in C++) using the scalable one-round membership algorithm of [11]. The virtual synchrony round and the membership round are conducted in parallel: once the end-points receive the `start_change` notifications, they send each other special synchronization messages which allow them to agree upon the set of messages to be delivered before moving to the new view. We are not aware of any other algorithm that implements virtual synchrony in one communication round without pre-agreement upon a globally unique identifier while also not imposing restrictions on the membership's choice of the next view.

Throughout this paper we use the I/O automaton formalism (cf. [16], Ch. 8) to provide rigorous specifications and algorithm descriptions. Previously suggested I/O automaton-style specifications of group communication systems (e.g., [7, 10]) used a single abstract automaton to represent multiple properties of the same system component and presented a single algorithm automaton that implements all of these properties. Thus, no means were provided for reasoning about a subset of the properties, and it was often difficult to follow which part of the algorithm implements which part of the specification. We address this shortcoming by specifying separate properties as separate abstract automata, and by incrementally constructing the algorithm that implements them – in each step adding support for an additional property – using a novel *inheritance*-based construct, recently introduced to the I/O automaton model [14, 13]. This paper informally argues the algorithm's correctness; a formal correctness proof by simulation is included in the full paper [12].

2 Formal model and notation

In the I/O automaton model (cf. [16], Ch. 8), a system component is described as a state-machine, called an *I/O automaton*. The transitions of this state-machine are associated with named actions, which are classified as either *input*, *output*, or *internal*. Input and output actions model the component's interaction with other components, while internal actions are externally-unobservable.

Formally, an I/O automaton is defined as the following five-tuple: a signature (input, output and internal actions), a set of states, a set of start states, a state-transition relation (a cross-product between states, actions, and states), and a partition of output and internal actions into *tasks*. Tasks are used for defining fairness conditions.

An action π is said to be *enabled* in a state $\mathtt{s}$ if the automaton has a transition of the form $(\mathtt{s}, \pi, \mathtt{s}')$; input actions are enabled in every state. An *execution* of an automaton is an alternating sequence of states and actions that begins with its start state and in which every action is enabled in the preceding state. An infinite execution is *fair* if, for each task, it either contains infinitely many actions from this task or infinitely many occurrences of states in which no action from this task is enabled; a finite execution is *fair* if no action is enabled in its final state. A *trace* is a subsequence of an execution consisting solely of the automaton's external actions. A *fair trace* is a trace of a fair execution.

When reasoning about an automaton, we are only interested in its externally-observable behavior as reflected in its traces. There are two types of trace properties: *safety* and *liveness.* Safety properties usually specify that some particular bad thing never happens. In this paper we specify safety properties using centralized (global) I/O automata that generate the legal sets of traces; for such automata we do not specify task partitions. Each external action in such a centralized automaton is tagged with a subscript which denotes the process at which this action occurs. An algorithm automaton *satisfies* a specification if all of its traces are also traces of the specification automaton. Liveness properties usually specify that some good thing eventually happens. An implementation automaton satisfies a liveness property if the property holds in all of its *fair* traces.

The *composition operation* defines how automata interact via their input and output actions: It matches output and input actions with the same name in different component automata; when a component automaton performs a step involving an output action, so do all components that have this action as an input one. When reasoning about a certain system component, we compose it with abstract specification automata that specify the behavior of its environment.

I/O automata are conveniently presented using the *precondition-effect* style: In this style, typed state variables with initial values specify the set of states and the start states. A variable type is a set (if S is a set, the notation $S_\perp$ refers to the set $S \cup \{\perp\}$). Transitions are grouped by action name, and are specified as a list of triples consisting of an action name (possibly with parameters), a `pre` : block with preconditions on the states in which the action is enabled, and an `eff` : block which specifies how the pre-state is modified *atomically* to yield the post-state.

We use a novel *inheritance*-based formal concept, recently introduced into the I/O automaton model [14, 13]. A *child* automaton is specified as a modification of the parent automaton's code. When presenting a child we first specify a *signature extension* which consists of new actions (labeled new) and modified actions (a modified action is labeled with the name of the action which it modifies as follows:

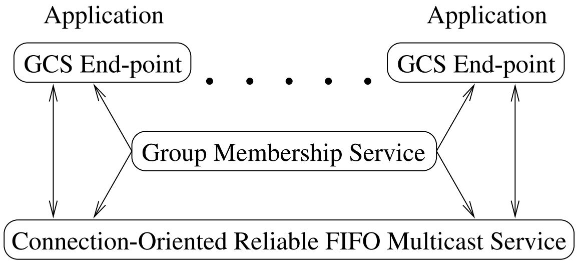


Fig. 1. The client-server architecture.

modifies `parent.action(parameters)`). We next specify the *state extension* consisting of new state variables added by the child. Finally, we describe the *transition restriction* which consists of new preconditions and effects added by the child to both new and modified actions. For modified actions, the preconditions and effects of the parent are appended to those added by the child. New effects added by the child are performed before the effects of the parent, all of them in a single atomic step. The child's effects are not allowed to modify state variables of the parent, to ensure that the set of traces of the child, when projected onto the parent's signature, is a subset of the parent's set of traces.

3 Environment specification

Our service is implemented in an asynchronous message-passing environment. Processes can crash, communication links may fail and may later recover, possibly causing network partitions and merges. In [12], we also model process recovery.

The service is implemented by *group communication service (GCS) end-points* running as clients of an external membership service whose specification appears in Section 3.1. The end-points communicate with each other using a reliable FIFO multicast service which we describe in Section 3.2, as depicted in Figure 1. We use the words "process" and "end-point" interchangeably.

3.1 The membership service

In Figure 2 we specify an external membership service whose interface consists of two output events:

`start_change`$_{\texttt{p}}$(`cid`, `set`) notifies process `p` that the membership service is attempting to form a new view with the members of `set`; `cid` is a local identifier.

`view`$_{\texttt{p}}$(`v`) notifies process `p` of the new view `v`. A view `v` is a triple consisting of an identifier `v.id`, a set of members `v.set`, and a function `v.startId` that maps members of `v` to start_change identifiers. Two views are the same if they consist of identical triples.

The membership specification captures two basic membership properties, which are fulfilled by virtually all group membership services (e.g., [2, 8, 4, 11, 18]): *Self Inclusion* requires every view delivered to an end-point `p` to include `p` as a member, and *Local Monotonicity* requires that view identifiers delivered to `p` be monotonically increasing.

In addition, the MBRSHP automaton specifies, using the `mode`[`p`] variable, that the membership service must precede every view `v` sent to end-point `p` with at least one `start_change` notification to `p`. It also requires that, for every view `v` sent to `p`, the start_change identifier `v.startId`(`p`) be the same as the `cid` of the latest `start_change` issued to `p` before the view, and that `v.set` be a subset of the `set` suggested in that `start_change`. Note that this specification does allow the membership service to add new processes while it is reconfiguring, as long as a new `start_change` is sent to the clients. Also note that the specified service is *partitionable* [20, 4], i.e., allows several disjoint views to exist concurrently.

The specification allows for simple and efficient distributed implementations, e.g., [11], as well as many other existing membership algorithms (e.g., [2]) which could be easily extended to provide the specified interface and semantics. In a possible implementation, a small number of servers could support a large number of clients, communicating with them asynchronously via FIFO ordered channels. Fault-tolerant implementations that support client migration are also possible if the server name is included in the start_change identifier to guarantee its local uniqueness.

3.2 The reliable FIFO multicast service

The group communication end-points communicate with each other using an underlying multicast service that provides reliable FIFO communication between every pair of connected processes. Many existing group communication systems (e.g., [9, 4]) implement virtual synchrony over similar underlying reliable communication substrates. In our implementation, we currently use the service of [19].

```
AUTOMATON MBRSHP

Type StartChangeId: Total-order; cid$_0$ is smallest.
    ViewId: Partial-order; vid$_0$ is smallest.
    View: ViewId x SetOf(Proc) x
                      x (Proc $\rightarrow$ StartChangeId)
Def  v$_p$ = $\langle$vid$_0$, {p}, {(p $\rightarrow$cid$_0$)}$\rangle$

Signature:
 Output: start_change$_p$(cid, set), Proc p,
              StartChangeId cid, SetOf(Proc) set
         view$_p$(v), Proc p, View v

State:
 ($\forall$ Proc p) View  mbrshp_view[p], initially v$_p$
 ($\forall$ Proc p) (StartChangeId x SetOf(Proc))
           start_change[p], initially $\langle$cid$_0$, {}$\rangle$
 ($\forall$ Proc p) mode[p] $\in$ {normal, change_started},
           initially normal

Transitions:
 OUTPUT  start_change$_p$(cid, set)
 pre: cid $>$ start_change[p].id
      p $\in$ set
 eff: start_change[p] $\leftarrow$ $\langle$cid, set$\rangle$
      mode[p] $\leftarrow$ change_started


 OUTPUT  view$_p$(v)
 pre: p $\in$ v.set $\wedge$ v.id $>$ mbrshp_view[p].id
      v.set $\subseteq$ start_change[p].set
      v.startId(p) = start_change[p].id
      mode[p] = change_started
 eff: mbrshp_view[p] $\leftarrow$ v
      mode[p] $\leftarrow$ normal
```

Fig. 2. Membership service safety spec.

Figure 3 presents a centralized automaton CO_RFIFO which specifies a multicast service appropriate for our group communication algorithm. CO_RFIFO maintains a FIFO queue `channel`[p][q] for every pair of end-points. An input action $\mathtt{send}_\mathtt{p}(\mathtt{set}, \mathtt{m})$ models the multicast of message `m` from end-point `p` to the end-points listed in the `set` by appending `m` to the queues `channel`[p][q] for every end-point `q` in `set`.

The $\texttt{deliver}_{\texttt{p,q}}(\texttt{m})$ action removes the first message from `channel`[`p`][`q`] and delivers it to `q`.

An end-point `p` may use the action $\texttt{reliable}_{\texttt{p}}(\texttt{set})$ to require CO_RFIFO to maintain reliable (gap-free) FIFO connections to the end-points in `set`. For every process `q` not in this `set`, CO_RFIFO may lose an arbitrary suffix of the messages sent from `p` to `q`, as modeled by the action `lose`(`p`, `q`).

In specifying liveness of CO_RFIFO, we require that messages sent to live and connected processes eventually reach their destinations. We formulate this property by defining every $\texttt{deliver}_{\texttt{p,q}}(\texttt{m})$ to be a task if and only if `q` is a member of `live_set`[`p`], a special variable periodically set by input actions $\texttt{live}_{\texttt{p}}(\texttt{set})$. The $\texttt{live}_{\texttt{p}}(\texttt{set})$ inputs are assumed to reflect the real state of the network, that is, the set of processes which are really alive and connected to `p`. Notice that we could not use the variable `reliable_set`[`p`] in this formulation because it is controlled by the client and thus does not necessarily reflect the *real* network situation.

4 GCS Specifications

We present the safety and liveness properties satisfied by our group communication service in Sections 4.1 and 4.2 respectively. These properties have been proven to be useful for many distributed applications (see [20]).

4.1 Safety properties

We present our safety specifications in four steps, as four automata: In Section 4.1 we specify a simple group communication service that provides reliable FIFO multicast within views. In Section 4.1 we extend the specification of Section 4.1 to also require that processes moving together from view `v` to view `v′` deliver the same messages in view `v`. In Section 4.1 we specify a service which provides *transitional sets* (first presented as part of *Extended Virtual Synchrony (EVS)* [17]). In Section 4.1 we specify the Self Delivery property which requires processes to deliver their own messages. The specified services are partitionable.

Within-view reliable FIFO multicast In Figure 4 we present the within-view reliable FIFO (WV_RFIFO) service specification. The specification uses centralized queues `msgs`[`p`][`v`] of application messages for each sender `p` and view `v`. The action $\texttt{send}_{\texttt{p}}(\texttt{m})$ models the multicast of message `m` from process `p` to the members of `p`'s current view by appending `m` to `msgs[p][current_view[p]]`. The $\texttt{deliver}_{\texttt{p}}(\texttt{q},\texttt{m})$ action models the delivery to process `p` of message `m` sent by process `q` while in `p`'s current view. The specification enforces gap-free FIFO ordered delivery of messages by using the variable `last_dlvrd[q][p]` to index the last message from `q` delivered to `p` in `p`'s current view.

This specification captures the following properties:

- Views delivered to the application satisfy Local Monotonicity and Self Inclusion (cf. Sec. 3.1).
- Messages are delivered in the same view in which they were sent. This property is useful for many applications (cf. [8, 20]) and appears in several systems and specifications (e.g., see [1, 2, 17, 7]). A weaker property that requires each message to be delivered in the same view at every process that delivers it, but not necessarily the view in which it was sent, is typically implemented on top of an implementation of within-view delivery (see [20]).
- Messages are delivered in gap-free FIFO order (within views). This is a basic property upon which one can build services with stronger ordering guarantees (e.g., causally or totally ordered multicast).

Virtual synchrony In this section we specify a virtually synchronous reliable FIFO multicast service, VS_RFIFO, as a child of the presented above WV_RFIFO automaton. The VS_RFIFO specification consists of the code given in both Figures 4 and 5.

In addition to the properties inherited from WV_RFIFO, the VS_RFIFO specification also requires that processes moving together from view `v` to view $\texttt{v}'$ deliver the same set of messages in `v`. To enforce this property, the specification introduces internal actions $\texttt{set\_cut}(\texttt{v},\texttt{v}',\texttt{c})$ that non-deterministically fix the set of messages to be delivered in view `v` by every process that moves from `v` to $\texttt{v}'$. This set is represented by the index of the last message to be delivered in `v` from each sender. Note that a process that delivers messages beyond an already established cut is not allowed to move into the view associated with the cut.

This property is commonly provided (e.g., [17, 8, 20, 18, 10]) and is often called *Virtual Synchrony* by itself. It is especially useful for applications that implement data replication using the state machine approach (e.g., [7, 15]). Such applications may exploit Virtual Synchrony to avoid sending costly synchronization messages among processes that continue together from one view to the next.

```
AUTOMATON CO_RFIFO

Signature:
Input:
  $\text{send}_{p}$(set,m), Proc p, SetOf(Proc) set, Msg m
  $\text{reliable}_{p}$(set), Proc p, SetOf(Proc) set
  $\text{live}_{p}$(set), Proc p, SetOf(Proc) set
Output:   $\text{deliver}_{p,q}$(m), Proc p, Proc q, Msg m
Internal: lose(p,q), Proc p, Proc q

State:
 ($\forall$ Proc p, Proc q) SeqOf(Msg) channel[p][q],
                           initially empty
 ($\forall$ Proc p) SetOf(Proc) reliable_set[p], init. $\{p\}$
 ($\forall$ Proc p) SetOf(Proc) live_set[p], init. $\{p\}$

Transitions:
 INPUT  $\textbf{send}_{p}$(set, m)
 eff: ($\forall$ q $\in$ set) append m to channel[p][q]

 OUTPUT  $\textbf{deliver}_{p,q}$(m) hidden parameter live_set[p]
 pre: m $=$ First(channel[p][q])
 eff: dequeue m from channel[p][q]


 INPUT  $\textbf{reliable}_{p}$(set)
 eff: reliable_set[p] $\leftarrow$  set

 INTERNAL  $\textbf{lose}$(p, q)
 pre: q $\notin$ reliable_set[p]
 eff: dequeue last message from channel[p][q]

 INPUT  $\textbf{live}_{p}$(set)
 eff: live_set[p] $\leftarrow$ set

Tasks:
 1. ($\forall$ p)($\forall$ q $\in$ live_set[p]) $\{$$\text{deliver}_{p,q}$(m)$\}$
 2. $\{$dummy()$\}$ $\cup$ $\{$$\text{deliver}_{p,q}$(m) | q $\notin$ live_set[p]$\}$
    $\cup$ $\{$lose(p,q)$\}$
```

Fig. 3. CO_RFIFO service specification.

```
AUTOMATON WV_RFIFO : SPEC

Signature:
Input:  send$_p$(m), Proc p, AppMsg m
Output: deliver$_p$(q, m), Proc p, Proc q, AppMsg m
        view$_p$(v), Proc p, View v

State:
($\forall$ Proc p, View v) SeqOf(AppMsg) msgs[p][v],
                   initially empty
($\forall$ Proc p, Proc q) Int last_dlvrd[p][q], init. 0
($\forall$ Proc p) View current_view[p], init. v$_p$

Transitions:
INPUT  send$_p$(m)
eff: append m to msgs[p][current_view[p]]

OUTPUT  deliver$_p$(q, m)
pre: m=msgs[q][current_view[p]][last_dlvrd[q][p]+1]
eff: last_dlvrd[q][p] $\leftarrow$ last_dlvrd[q][p]+1

OUTPUT  view$_p$(v)
pre: p $\in$ v.set $\wedge$ v.id $>$ current_view[p].id
eff: ($\forall$ q) last_dlvrd[q][p] $\leftarrow$ 0
     current_view[p] $\leftarrow$ v
```

Fig. 4. WV_RFIFO service specification.

```
AUTOMATON VS_RFIFO : SPEC    MODIFIES  WV_RFIFO : SPEC

Signature Extension:
 Output:   view_p(v) modifies wv_rfifo.view_p(v)
 Internal: set_cut(v, v′, c), View v, View v′,
                          (Proc → Int)⊥  c new

State Extension:
 (∀ View v, v′) (Proc→Int)⊥  cut[v][v′], init. ⊥

Transition Restriction:
 OUTPUT  view_p(v)
 pre: cut[current_view[p]][v] ≠ ⊥
 (∀ q) last_dlvrd[q][p] = cut[current_view[p]][v](q)

 INTERNAL  set_cut(v, v′, c)
 pre: cut[v][v′] = ⊥
 eff: cut[v][v′] ← c
```

Fig. 5. VS_RFIFO service specification.

Transitional set While Virtual Synchrony is a useful property, a process that moves from view $\mathtt{v}$ to view $\mathtt{v}'$ cannot locally tell which of the processes in $\mathtt{v.set} \cap \mathtt{v}'\mathtt{.set}$ move to view $\mathtt{v}'$ directly from view $\mathtt{v}$, and which move to $\mathtt{v}'$ from some other view. In order for the application to be able to exploit the Virtual Synchrony property, application processes need to be told which other processes move together with them from their old views in to their new views. The set of such processes is called a *transitional set*. The notion of a transitional set was first introduced as part of a special transitional view in the EVS [17] model. In our formulation (as in [20]), transitional sets are delivered to the applications together with (regular) views, as an additional parameter $\mathtt{T}$. The delivery of transitional sets satisfies the following property (cf. [20]):

Property 41 *The transitional set delivered by a process* $\mathtt{p}$ *when it moves from view* $\mathtt{v}$ *to view* $\mathtt{v}'$ *is a subset of* $\mathtt{v.set} \cap \mathtt{v}'\mathtt{.set}$ *that includes (a) all the processes (including* $\mathtt{p}$*) that move directly from* $\mathtt{v}$ *to* $\mathtt{v}'$ *and (b) no member of* $\mathtt{v}'\mathtt{.set}$ *that moves to* $\mathtt{v}'$ *from any view other than* $\mathtt{v}$*.*

Note that processes that move to the same view from different views deliver different transitional sets.

Figure 6 contains an automaton (without inheritance) specifying the Transitional Set property. Before $\mathtt{p}$ can move from view $\mathtt{v}$ to $\mathtt{v}'$, each member $\mathtt{q}$ of $\mathtt{v.set} \cap \mathtt{v}'\mathtt{.set}$ must execute $\mathtt{set\_prev\_view_q}(\mathtt{v}')$ to "declare" the view from which it intends to move to $\mathtt{v}'$; this action sets $\mathtt{prev\_view}[\mathtt{q}][\mathtt{v}']$ to $\mathtt{q}$'s current view. The transitional set delivered by $\mathtt{p}$ with $\mathtt{v}'$ is then computed to consist of those $\mathtt{q}$ in $\mathtt{v.set} \cap \mathtt{v}'\mathtt{.set}$ for which $\mathtt{prev\_view}[\mathtt{q}][\mathtt{v}'] = \mathtt{v}$.

Self delivery In Figure 7 we modify the WV_RFIFO specification automaton (Fig. 4) to capture the Self Delivery property by forbidding an end-point $\mathtt{p}$ to move from view $\mathtt{v}$ to $\mathtt{v}'$ before delivering all its own application messages sent in $\mathtt{v}$.

This *safety* property, when accompanied by the liveness property of Section 4.2, implies the Self Delivery *liveness* properties of [20] and [17], which require processes to *eventually* deliver their own messages.

4.2 Liveness property

In a fault-prone asynchronous model, it is not feasible to require that a group communication service be live in every execution. The only way to specify useful liveness properties without strengthening the communication model is to make these properties *conditional* on the underlying

```
AUTOMATON TRANS_SET : SPEC

Signature:
Output: $view_p$(v,T), Proc p, View v, SetOf(Proc) T
Internal: $set\_prev\_view_p$(v), Proc p, View v

State:
($\forall$ Proc p) View current_view[p], initially $v_p$
($\forall$ Proc p, View v) $View_\bot$ prev_view[p][v], init. $\bot$

Transitions:
 OUTPUT  $view_p$(v, T)
 pre: prev_view[p][v] = current_view[p]
      ($\forall$ q $\in$ v.set $\cap$ current_view[p].set)
             prev_view[q][v] $\neq$ $\bot$
      T = {q $\in$ v.set $\cap$ current_view[p].set |
             prev_view[q][v] = current_view[p]}
 eff: current_view[p] $\leftarrow$ v

 INTERNAL  $set\_prev\_view_p$(v)
 pre: p $\in$ v.set
      prev_view[p][v] = $\bot$
 eff: prev_view[p][v] $\leftarrow$ current_view[p]
```

Fig. 6. Transitional set specification.

```
AUTOMATON SELF : SPEC    MODIFIES  WV_RFIFO : SPEC

Signature Extension:
 Output: $view_p$(v) modifies wv_rfifo.$view_p$(v)

Transition Restriction:
 OUTPUT  $view_p$(v)
 pre: last_dlvrd[p][p] =
        = LastIndexOf(msgs[p][current_view[p]])
```

Fig. 7. Self Delivery property specification.

network behavior (as specified, e.g., in [7, 20]). Since our GCS uses an external membership service, we condition its liveness on the behavior of the membership service (which itself is assumed to satisfy some meaningful liveness properties, e.g., those of [11]). Provided the membership eventually delivers the same view to all the view end-points and does not deliver any subsequent views (i.e., stabilizes), we require the end-points to eventually deliver this view and all the messages sent in this view to their applications.

Property 42 *Let* $\mathtt{v}$ *be a view with* $\mathtt{v.set} = \mathtt{S}$. *Let* α *be a fair execution of a group communication service* GCS *in which, for every* $\mathtt{p} \in \mathtt{S}$, MBRSHP.$\mathtt{view}_\mathtt{p}(\mathtt{v})$ *action occurs and is followed by neither* MBRSHP.$\mathtt{view}_\mathtt{p}$ *nor* MBRSHP.$\mathtt{start\_change}_\mathtt{p}$. *Then at each* $\mathtt{p} \in \mathtt{S}$, GCS.$\mathtt{view}_\mathtt{p}(\mathtt{v})$ *eventually occurs. Furthermore, for every* GCS.$\mathtt{send}_\mathtt{p}(\mathtt{m})$ *that occurs after* GCS.$\mathtt{view}_\mathtt{p}(\mathtt{v})$, *and for every* $\mathtt{q} \in \mathtt{S}$, GCS.$\mathtt{deliver}_\mathtt{q}(\mathtt{p}, \mathtt{m})$ *also occurs.*

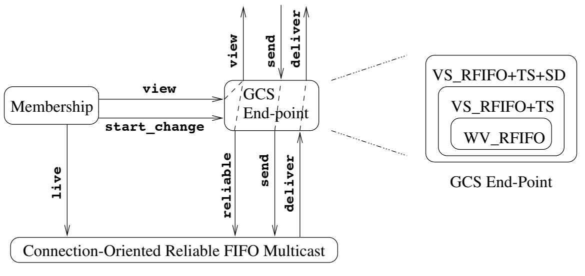
It is important to note that although our liveness property requires GCS to be live only in *certain* executions, any implementation which satisfies this property has to attempt to be live in *every* execution because of its inability to test the external condition of the membership becoming stable. Also note that, even though membership stability is formally required to last forever, in practice it only has to hold "long enough" for GCS to reconfigure.

5 The group multicast algorithm

Our group communication service is implemented by a collection of GCS end-points, each running the same algorithm. Figure 8 (a) shows the interaction of a GCS end-point with its environment, MBRSHP and CO_RFIFO (see Sec. 3). The end-point interacts with its application client by accepting the client's send-requests and by delivering application messages and views to the client. The end-point uses CO_RFIFO to send messages to other GCS end-points and to receive messages sent by other GCS end-points. When necessary, the end-point uses the action `reliable` to inform CO_RFIFO of the set of end-points to which CO_RFIFO must maintain reliable (gap-free) FIFO connections. The GCS end-point also receives `start_change` and `view` notifications from the membership service.

The algorithm running at each end-point is constructed in steps, at each step adding support for a new property:

- First, we present an algorithm $\text{WV\_RFIFO}_\mathtt{p}$ for an end-point of a within-view reliable FIFO multicast service.
- Then, in Section 5.2, we add support for the Virtual Synchrony and Transitional Set properties. We present a child $\text{VS\_RFIFO+TS}_\mathtt{p}$ of $\text{WV\_RFIFO}_\mathtt{p}$, and argue that the service built from $\text{VS\_RFIFO+TS}_\mathtt{p}$ end-points satisfies safety specifications VSRFIFO : SPEC and TS : SPEC, and liveness Property 42.



(a) Interaction with Membership and Multicast Services **(b)** Inheritance Hierarchy

Fig. 8. GCS end-point and environment.

- Finally, in Section 5.3, we add support for Self Delivery. The resulting automaton VS_RFIFO+TS+SD$_\mathtt{p}$ models a GCS end-point. Due to the use of inheritance, the service built from these end-points satisfies WV_RFIFO : SPEC, VSRFIFO : SPEC, and TS : SPEC. We argue that it also satisfies safety specification SELF : SPEC and liveness Property 42.

In the presented automata, each locally controlled action is defined to be a task by itself, which means that, if it becomes and stays enabled, it would eventually get executed.

When composing automata into a service, actions of the type MBRSHP.$\mathtt{start\_change}_\mathtt{p}(\mathtt{id}, \mathtt{set})$ are linked with CO_RFIFO.$\mathtt{live}_\mathtt{p}(\mathtt{set})$, and MBRSHP.$\mathtt{view}_\mathtt{p}(\mathtt{v})$ with CO_RFIFO.$\mathtt{live}_\mathtt{p}(\mathtt{v.set})$. This way, the `live_set`[`p`] at CO_RFIFO matches the MBRSHP's perception of which processes are alive and connected to `p`. (We assume that every permanently disconnected end-point is eventually excluded by either a `start_change` or a `view` notification.) Also, in the composed system, all output actions except the application interface are reclassified as internal.

We present our algorithm at a level that would be easy to follow and then supplement this presentation with a discussion of some important, practical optimizations.

5.1 Within-view reliable FIFO multicast algorithm

In this section we present algorithm WV_RFIFO$_\mathtt{p}$ for an end-point `p` of a service that interacts with MBRSHP and CO_RFIFO services and satisfies the WV_RFIFO : SPEC safety specification and liveness Property 42.

The MBRSHP and CO_RFIFO services by themselves already provide most of the properties required by the WV_RFIFO : SPEC specification:

```
AUTOMATON WV_RFIFO_p

Signature:
 Input:  send_p(m), AppMsg m
         co_rfifo.deliver_q,p(m), Proc q,
               (AppMsg + ViewMsg + FwdMsg) m
         mbrshp.view_p(v), View v

 Output: deliver_p(q, m), Proc q, AppMsg m
         co_rfifo.send_p(set, m), SetOf(Proc) set,
               (AppMsg + ViewMsg + FwdMsg) m
         view_p(v), View v

State:
 // Variables for handling application messages
 (∀ Proc q, View v) SeqOf(AppMsg_⊥) msgs[q][v],
                            initially empty
 Int last_sent, initially 0
 (∀ Proc q) Int last_rcvd[q],  initially 0
 (∀ Proc q) Int last_dlvrd[q], initially 0

 // Variables for handling views and view messages
 View current_view, initially v_p
 View mbrshp_view,  initially v_p
 For all Proc q: View view_msg[q], initially v_q

 SetOf(Proc) reliable_set, initially v_p.set

Transitions:
 INPUT  mbrshp.view_p(v)
 eff: mbrshp_view  ←  v

 OUTPUT  view_p(v)
 pre: v =  mbrshp_view ≠ current_view
 eff: current_view  ←  v
      last_sent   ←  0
      (∀ q) last_dlvrd[q]  ←  0

 OUTPUT  co_rfifo.reliable_p(set)
 pre: current_view.set ⊆ set
 eff: reliable_set  ←  set

 OUTPUT  co_rfifo.send_p(set, tag=view_msg, v)
 pre: view_msg[p] ≠ current_view
      current_view.set ⊆ reliable_set
      set =  current_view.set - {p}
      v = current_view
 eff: view_msg[p]  ←  current_view

 INPUT  co_rfifo.deliver_q, p(tag=view_msg, v)
 eff: view_msg[q]   ←  v
      last_rcvd[q]  ←  0

 INPUT  send_p(m)
 eff: append m to msgs[p][current_view]

 OUTPUT  deliver_p(q, m)
 pre: m = msgs[q][current_view][last_dlvrd[q]+1]
      (q = p)   ⇒   (last_dlvrd[q] < last_sent)
 eff: last_dlvrd[q]  ←  last_dlvrd[q] + 1

 OUTPUT  co_rfifo.send_p(set, tag=app_msg, m)
 pre: view_msg[p] = current_view
      set =  current_view.set - {p}
      m = msgs[p][current_view][last_sent + 1]
 eff: last_sent  ←  last_sent + 1

 INPUT  co_rfifo.deliver_q,p(tag=app_msg, m)
 eff: msgs[q][view_msg[q ]] [last_rcvd[q]+1] ← m
      last_rcvd[q]  ←  last_rcvd[q] + 1

 OUTPUT  co_rfifo.send_p(set,tag=fwd_msg,r,v,m,i)
 pre: m = msgs[r][v][i]

 INPUT  co_rfifo.deliver_q,p(tag=fwd_msg,r,v,m,i)
 eff:  msgs[r][v][i]  ←  m
```

Fig. 9. Within-view reliable FIFO multicast end-point automaton.

MBRSHP generates views that satisfy Local Monotonicity and Self Inclusion, and CO_RFIFO provides gap-free FIFO communication. Since WV_RFIFO$_p$ can just forward to its application the views generated by MBRSHP and can use CO_RFIFO to multicast application messages to other end-points, it only needs to ensure that messages are delivered in the same views in which they were sent. This can be done simply by tagging messages with the views in which they were sent and by allowing delivery of a message when its view tag matches the end-point's current view.

As an optimization of this idea, instead of tagging each message with a view in which it was sent, our algorithm sends a single, special `view_msg`(`v`) to all members of view `v` before sending them application messages in that view. An end-point can deduce the view in which an application message is sent from the latest `view_msg`(`v`) received from the application message sender.

The algorithm is captured in the WV_RFIFO$_{\mathtt{p}}$ automaton of Figure 9. Note that, instead of blindly relying on CO_RFIFO regarding which messages get delivered in a given view, WV_RFIFO allows processes to forward application messages on behalf of other processes. The code of WV_RFIFO$_{\mathtt{p}}$ does not specify a particular forwarding strategy – it allows for non-deterministic forwarding of messages. Without this, more refined versions and extensions of WV_RFIFO would not be able to introduce a specific forwarding strategy (as we do in VS_RFIFO+TS by adding a precondition on the action that sends forwarded messages).

There is also another place where the code leaves a non-deterministic choice: it is in handling of the `reliable_set` of CO_RFIFO. The code allows it to be an arbitrary superset of `current_view.set`. This set is further restricted in a child VS_RFIFO+TS$_{\mathtt{p}}$ of WV_RFIFO$_{\mathtt{p}}$.

The correctness of WV_RFIFO follows from the use of ordered message queues, the safety and liveness properties of CO_RFIFO, and the safety properties of MBRSHP. A formal proof is given in [12].

Also note that the presented code never removes messages from its buffers. An actual implementation can and should employ some sort of a garbage collection mechanism, for example discard messages sent in older views when moving in to a new view.

5.2 Virtual synchrony and transitional sets

The WV_RFIFO service presented above guarantees that in each view `v` every member delivers some prefix of the FIFO ordered messages sent by each end-point in `v`. The VS_RFIFO+TS service presented in this section extends WV_RFIFO to also guarantee that those end-points which transition directly from view `v` to the same view $\mathtt{v}'$ deliver not just "some" prefixes but "the same" prefixes of the FIFO ordered messages sent by each end-point in view `v` (cf. Sec. 4.1). Moreover, every view delivery is accompanied by a transitional set `T` that satisfies the Transitional Set property of Sec. 4.1.

In order to satisfy these two properties, an end-point moving from a view `v` to a view $\mathtt{v}'$ must first learn which other end-points may transition from `v` to $\mathtt{v}'$ and must agree with them on the lengths of the prefixes they need to deliver. In a nutshell, here is how the VS_RFIFO+TS service accomplishes this: Each time an end-point `p` is notified via MBRSHP.$\mathtt{start\_change}_{\mathtt{p}}(\mathtt{cid}, \mathtt{set})$ of the MBRSHP's attempt to form a new view, `p` reliably sends to `set` a synchronization message tagged with `cid`. When MBRSHP.$\mathtt{view}_{\mathtt{p}}(\mathtt{v}')$ is delivered to `p`, `p` uses the $\mathtt{v}'.\mathtt{startId}$ mapping to determine which synchronization message to use from each end-point `q` in $\mathtt{v.set} \cap \mathtt{v}'.\mathtt{set}$; it uses the one tagged with v′.`startId`(q). As a result, all end-points that move from view v to v′ use the same set of synchronization messages for computing the transitional set and the set of messages to be delivered to their application clients before v′. Notice that, by enriching views with the `startId` mapping, we eliminate the need to pre-agree on a common tag for identifying which synchronization messages to consider for a given view.

```
AUTOMATON VS_RFIFO+TS_p   MODIFIES   WV_RFIFO_p

Signature Extension:
 Input:  send_p(m)  modifies wv_rfifo.send_p(m)
         mbrshp.start_change_p(id, set), StartChangeId id, SetOf(Proc) set  new
         co_rfifo.deliver_q,p(m), Proc q, SyncMsg m   new

 Output: deliver_p(q, m)  modifies wv_rfifo.deliver_p(q, m)
         view_p(v, T), SetOf(Proc) T   modifies wv_rfifo.view_p(v)
         co_rfifo.reliable_p(set), SetOf(Proc) set  modifies wv_rfifo.co_rfifo.reliable_p(set)
         co_rfifo.send_p(set, m), SetOf(Proc) set, SyncMsg m  new
         co_rfifo.send_p(set, m)  modifies wv_rfifo.co_rfifo.send_p(set, m), FwdMsg m

State Extension:
 (StartChangeId × SetOf(Proc))_⊥ start_change, initially ⊥
 For all Proc q, ViewId id:  (View v, (Proc ⟶ Int) cut)_⊥ sync_msg[q][id], initially ⊥
 SetOf(FwdMsg) forwarded_set, initially empty

Transition Restriction:
 INPUT  mbrshp.start_change_p(id, set)
 eff: start_change  ⟵  ⟨ id, set ⟩

 OUTPUT  co_rfifo.reliable_p(set)
 pre: start_change = ⊥  ⇒  set = current_view.set
      start_change ≠ ⊥  ⇒  set = current_view.set ∪ start_change.set

 OUTPUT  co_rfifo.send_p(set, tag=sync_msg, cid, v, cut)
 pre: start_change ≠ ⊥
      start_change.set ⊆ reliable_set
      ⟨ cid, set ⟩  =  ⟨ start_change.id, start_change.set - {p} ⟩
      sync_msg[p][cid] = ⊥  ∧       v = current_view
      (∀ q ∈ current_view.set) cut(q) = LongestPrefixOf(msgs[q][v])
 eff: sync_msg[p][cid]  ⟵  ⟨ v, cut ⟩

 INPUT  co_rfifo.deliver_q,p(tag=sync_msg, cid, v, cut)
 eff: sync_msg[q][cid]  ⟵  ⟨ v, cut ⟩

 OUTPUT  deliver_p(q, m)
 pre:  if (start_change ≠ ⊥ ∧ sync_msg[p][start_change.id] ≠ ⊥) then
         if start_change.id ≠ mbrshp_view.startId(p) then
            last_dlvrd[q]+1 ≤ sync_msg[p][start_change.id].cut(q)
         else  let S = {r ∈ mbrshp_view.set ∩ current_view.set |
                          sync_msg[r][mbrshp_view.startId(r)].view = current_view}
              last_dlvrd[q]+1 ≤ max_{r ∈ S} sync_msg[r][mbrshp_view.startId(r)].cut(q)

 OUTPUT  view_p(v, T)
 pre: v.startId(p) = start_change.id    // to prevent delivery of obsolete views
      (∀ q ∈ v.set ∩ current_view.set) sync_msg[q][v.startId(q)] ≠ ⊥
      T = {q ∈ v.set ∩ current_view.set | sync_msg[q][v.startId(q)].view = current_view}
      (∀ q ∈ current_view.set) last_dlvrd[q] = max_{r ∈ T} sync_msg[r][v.startId(r)].cut(q)
 eff: start_change  ⟵  ⊥

 OUTPUT  co_rfifo.send_p(set,tag=fwd_msg,r,v,m,i)
 pre: (∀ q ∈ set) ⟨ q, r, v, i ⟩ ∉ forwarded_set
      ForwardStrategyPredicate( ⟨ set, r, v, i ⟩ , current_state)
 eff: (∀ q ∈ set) add ⟨ q, r, v, m, i ⟩ to forwarded_set
```

Fig. 10. Virtually synchronous reliable FIFO multicast and transitional set end-point automaton.

Algorithm details and safety argument Figure 10 presents the VS_RFIFO+TS$_{\mathtt{p}}$ automaton as a child of WV_RFIFO$_{\mathtt{p}}$. While there are no view changes, VS_RFIFO+TS$_{\mathtt{p}}$ does not modify the behavior of WV_RFIFO$_{\mathtt{p}}$. During a view change, VS_RFIFO+TS$_{\mathtt{p}}$ sends and handles synchronization messages, and also restricts the delivery of application messages according to the synchronization messages associated with the new view.

Upon receiving a $\mathtt{start\_change_p}(\mathtt{cid}, \mathtt{set})$ notification from MBRSHP, end-point `p` stores $\langle$`cid`, `set`$\rangle$ in the variable `start_change`, tells CO_RFIFO to maintain reliable communication to the end-points in $\mathtt{current\_view} \cup \mathtt{set}$, and then sends a synchronization message tagged with `cid` to every end-point in `set`. The synchronization message contains `p`'s current view `v` and a `cut`, which is a mapping from processes to indices; $\mathtt{cut}(\mathtt{q})$ is the index of the last message from `q` that `p` *commits* to deliver before delivering any view $\mathtt{v}'$ with $\mathtt{v}'.\mathtt{startId}(\mathtt{p}) = \mathtt{cid}$.

End-point `p` stores the synchronization message from `q` tagged with `cid` in $\mathtt{sync\_msg}[\mathtt{q}][\mathtt{cid}]$. Until `p` receives a view from MBRSHP, it does not know which synchronization messages from others to consider, so it restricts delivery of application messages to only those identified in its own latest `cut`. When a MBRSHP view $\mathtt{v}'$ is delivered to `p`, the $\mathtt{v}'.\mathtt{startId}$ mapping tells `p` to use the synchronization messages $\mathtt{sync\_msg}[\mathtt{q}][\mathtt{v}'.\mathtt{startId}(\mathtt{q})]$ from $\mathtt{q} \in \mathtt{v}'.\mathtt{set}$. The members of `p`'s transitional set for view $\mathtt{v}'$ are those end-points `q` whose $\mathtt{sync\_msg}[\mathtt{q}][\mathtt{v}'.\mathtt{startId}(\mathtt{q})].\mathtt{view}$ is the same as `p`'s current view `v`. After receiving view $\mathtt{v}'$ from MBRSHP, `p` allows delivery of application messages identified by `cuts` in the synchronization messages from the processes that are already known to be members of the transitional set. The delivery of $\mathtt{view_p}(\mathtt{v}', \mathtt{T})$ to `p`'s application is enabled only after `p` has received the synchronization messages from all the potential members of `T` and after it has delivered all application messages committed to by the `cuts` of the members of `T`. Since all the end-points that move from `v` to $\mathtt{v}'$ use the same set of synchronization messages, the Virtual Synchrony and Transitional Set safety properties are satisfied.

End-point `p` is guaranteed to eventually receive all the application messages sent by the members of its transitional set `T`. However, `p` may fail to receive some of the application messages sent by disconnected end-points (not in `T`) although certain cuts of members of `T` commit to deliver these messages. Such messages need to be forwarded to `p` by the members of `T` that have them. These members of `T` deduce from the `p`'s cut that `p` lacks these messages and use a `ForwardingStrategyPredicate` to compute which of them have to forward which missing messages to `p`. We describe some of the many possible such predicates in [12].

Liveness of VS_RFIFO+TS We show that, in a fair execution of VS_RFIFO+TS in which the same view `v'` is delivered to all its members as their last MBRSHP event, the three preconditions on the $\mathtt{view_p}(\mathtt{v'}, \mathtt{T_p})$ delivery are eventually satisfied for every $\mathtt{p} \in \mathtt{v'.set}$:

1. Condition `v'.startId(p)` = `start_change.id` remains true since by the assumption there are no subsequent `start_change` events at `p`.
2. End-point `p` eventually receives synchronization messages tagged with the "right" `cid` from everybody in $\mathtt{v.set} \cap \mathtt{v'.set}$ because they keep taking steps towards reliably sending these synchronization messages to `p` (by low-level fairness of the code) and because CO_RFIFO eventually delivers these messages to `p` (by the liveness assumption on CO_RFIFO).
3. End-point `p` eventually receives and delivers all the messages committed to in the cuts of the members of the transitional set $\mathtt{T_p}$ because for each such message there is at least one end-point in $\mathtt{T_p}$ that has the message in its `msgs` buffer and that would reliably forward it to `p` (according to the `ForwardingStrategyPredicate`) if so necessary. Also, `p` never delivers any messages beyond those committed to in the cuts of the members of $\mathtt{T_p}$ because of the precondition on application message delivery.

Optimizations Notice that end-point `p` does not need to send its current view and its cut to end-points which are not in `current_view.set` because `p` cannot be included in their transitional sets. Nevertheless, these end-points may wait to hear from `p` as `p` may still be in their current views. Therefore, in our algorithm, `p` sends synchronization messages to all the end-points in `start_change.set`. As an optimization, `p` could send a smaller synchronization message to processes in `start_change.set` − `current_view.set`, containing its `start_change.id` only (but neither a view nor a cut). The recipients of this message would know not to include `p` in their transitional sets for views `v'` with `v'.startId(p)` = `p`'s `start_change.id`. When using this optimization, `p` also does not need to include its current view in the synchronization messages sent to `current_view.set` − `start_change.set`, since the view information can be deduced from `p`'s `view_msg`.

Another optimization can be used to minimize synchronization message sizes if we strengthen the membership specification to require a MBRSHP.`start_change` to be sent every time the membership changes its mind about the next view. In this case, the latest MBRSHP.`start_change` has the same membership as the delivered MBRSHP.`view`, and therefore the synchronization messages do not need to include information about messages delivered from end-points in `start_change.set` ∩ `current_view.set` because the synchronization message from each of these end-points can terminate a stream of application messages that this end-point would deliver in its current view.

5.3 Self delivery

As a final step in constructing the automaton that models an end-point of our group communication service, GCS$_\texttt{p}$, we add support for Self Delivery to the VS_RFIFO+TS$_\texttt{p}$ automaton presented above. Self Delivery requires each end-point to deliver to its application all the messages the application sends in a view, before moving on to the next view.

In order to implement Self Delivery and Virtual Synchrony together in a live manner, each end-point must *block* its application from sending new messages while a view change is taking place (as proven in [8]). Therefore, we modify VS_RFIFO+TS$_\texttt{p}$ to have an output action `block` and an input action `block_ok`, and we assume that the application at end-point `p` has the matching actions and that it eventually responds to every `block` request with a `block_ok` response and subsequently refrains from sending messages until a `view` is delivered to it. In [12], we model this assumption with an abstract application automaton.

The GCS$_\texttt{p}$ automaton appears in Figure 11. After receiving the first `start_change` notification in a given view, end-point `p` issues a `block` request to the application and awaits receiving a `block_ok` response before sending a synchronization message to other members of `start_change.set`. The `cut` sent in the synchronization message commits to all the messages `p` received from its application in the current view.

Since the application is required to respond with `block_ok` and is then blocked from sending further messages, and since the `p`'s cut commits to all the messages the application has sent in the current view, the set of messages agreed upon based on the `cut`s includes all of `p`'s messages. Therefore, `p` delivers all these messages before moving on to a new view, and Self Delivery is satisfied. Since end-point `p` has its own messages on the `msgs`[p][p] queue, the modification does not affect the liveness property of VS_RFIFO+TS. Finally, we note that due to the use of inheritance, the GCS$_\texttt{p}$ automaton satisfies all the properties we have specified in Secion 4.

```
AUTOMATON GCS$_p$ = VS_RFIFO+TS+SD$_p$   MODIFIES  VS_RFIFO+TS$_p$

Signature Extension:
Input:  block_ok$_p$() new

Output: block$_p$() new
        view$_p$(v,T) modifies vs_rfifo+ts.view$_p$(v,T)
        co_rfifo.send$_p$(set, m) modifies
        vs_rfifo+ts.co_rfifo.send$_p$(set,SyncMsg m)

State Extension:
block_status $\in$ {unblocked, requested, blocked},
                     initially unblocked

Transition Restriction:
OUTPUT  block$_p$()
pre: start_change $\neq$ $\perp$
     block_status = unblocked
eff: block_status $\leftarrow$ requested

INPUT  block_ok$_p$()
eff: block_status $\leftarrow$ blocked

OUTPUT  co_rfifo.send$_p$(set, tag=sync_msg, cid, v, cut)
pre: block_status = blocked

OUTPUT  view$_p$(v,T)
eff: block_status $\leftarrow$ unblocked
```

Fig. 11. GCS$_p$ end-point automaton.

6 Conclusions

We have constructed a virtually synchronous group multicast algorithm which exchanges one round of synchronization messages during reconfiguration, in parallel with the execution of a group membership algorithm. In contrast to previously suggested virtual synchrony algorithms, our algorithm does not require processes to conduct an additional communication round in order to pre-agree upon a globally unique identifier and does not impose restrictions on membership service's choice of views. We are not aware of any other algorithm that has both of these features.

These features are achieved by virtue of a simple yet powerful idea: Membership service issues a *locally* unique start-change identifier every time it has new information about client membership. The inclusion of such identifiers in views eliminates the need to tag clients' messages with a common (globally unique) identifier.

The start-change interface is an important aspect of the design of a client-server oriented group communication service which decouples membership maintenance from group multicast in order to provide scalable group membership services in WANs. Maestro [6] and the service of [18] also separate the maintenance of membership from group multicast. Unlike Maestro [6], in our design, the client does not wait for the membership to agree upon a globally unique identifier before starting the virtual synchrony algorithm, and the membership service does not wait for responses from clients asserting that virtual synchrony was achieved before delivering views. Unlike [18], our service does not impose restrictions on the membership service's choice of views, thereby allowing its applications to benefit from Virtual Synchrony in more cases (as explained in Introduction).

In [12] we show that the service presented in Section 5 also provides meaningful and correct semantics in the environment where GCS end-points can crash and recover. In particular, it allows the recovered GCS end-points to continue running the algorithm under their original identity (in contrast e.g., to Isis [5] which requires recovered processes to assume new identities). Furthermore, GCS end-points do not need to store any information on stable storage.

Our service is implemented as part of a novel architecture for scalable group communication in WANs. After testing its current scalability limits, we intend to explore ways to improve the scallability further by incorporating a two-tier hierarchy into our algorithm, as suggested by Guo et al. [9]. With this approach, processes would send synchronization messages to their designated leaders, who would in turn exchange only the cumulative information among themselves. The framework in which we presented our algorithm allows us to incorporate extensions such as this one.

In [12] we formally prove the correctness of our algorithms. In particular, we prove the safety properties by defining simulation relations from the algorithm automata to the specification automata. The incremental way in which we have constructed our algorithms and specifications allows us to also construct the simulation proof incrementally. For example, in order to prove that VS_RFIFO+TS simulates VS_RFIFO+TS : SPEC we extend the simulation relation from WV_RFIFO to WV_RFIFO : SPEC and reason solely about the extension, without repeating the reasoning about the parent components. This reuse is justified by the Proof Extension theorem of [14, 13].

6.1 Acknowledgments

We thank Nancy Lynch, Alex Shvartsman and Jeremy Sussman for many discussions and helpful suggestions.

References

1. ACM. *Communications of the ACM 39(4), special issue on Group Communications Systems*, April 1996.
2. Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems*, 13(4), November 1995.
3. T. Anker, G. Chockler, D. Dolev, and I. Keidar. Scalable group membership services for novel applications. In Marios Mavronicolas, Michael Merritt, and Nir Shavit, editors, *Networks in Distributed Computing (DIMACS workshop)*, volume 45 of *DIMACS*, pages 23–42. American Mathematical Society, 1998.
4. Ö. Babaoğlu, R. Davoli, and A. Montresor. Partitionalbe group membership: Specification and algorithms. TR UBLCS97-1, Department of Computer Science, University of Bologna, January 1997.
5. K. Birman. *Building Secure and Reliable Network Applications.* Manning, 1996.
6. K. Birman, R. Friedman, M. Hayden, and I. Rhee. Middleware support for distributed multimedia and collaborative computing. In *Multimedia Computing and Networking (MMCN98)*, 1998.
7. A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. *ACM Transactions on Computer Systems*, 19(2):171–216, May 2001. Previous version appeared in PODC 1997.
8. Roy Friedman and Robbert van Renesse. Strong and Weak Virtual Synchrony in Horus. TR 95-1537, dept. of Computer Science, Cornell University, August 1995.
9. Katherine Guo, Werner Vogels, and Robbert van Renesse. Structured virtual synchrony: Exploring the bounds of virtual synchronous group communication. In *7th ACM SIGOPS European Workshop*, September 1996.
10. Jason Hickey, Nancy Lynch, and Robbert van Renesse. Specifications and proofs for ensemble layers. In *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS. Springer-Verlag, March 1999.
11. I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. A client-server oriented algorithm for virtually synchronous group membership in WANs. In *20th International Conference on Distributed Computing Systems (ICDCS)*, pages 356–365, April 2000. Full version: MIT Technical Memorandum MIT-LCS-TM-593a, June 1999, revised September 2000.

12. Idit Keidar and Roger Khazan. A client-server approach to virtually synchronous group multicast: Specifications, algorithms, and proofs. Technical Report MIT-LCS-TR-794, MIT Laboratory for Computer Science, November 1999.
13. Idit Keidar, Roger Khazan, Nancy Lynch, and Alex Shvartsman. An inheritance-based technique for building simulation proofs incrementally. To appear. Previous version in ICSE 2000, pp. 478–487.
14. Idit Keidar, Roger Khazan, Nancy Lynch, and Alex Shvartsman. An inheritance-based technique for building simulation proofs incrementally. In *22nd International Conference on Software Engineering (ICSE)*, pages 478–487, June 2000.
15. Roger Khazan, Alan Fekete, and Nancy Lynch. Multicast group communication as a base for a load-balancing replicated data service. In *12th International Symposium on DIStributed Computing (DISC)*, pages 258–272, Andros, Greece, September 1998.
16. N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, 1996.
17. L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *14th International Conference on Distributed Computing Systems (ICDCS)*, pages 56–65, June 1994. Full version: technical report ECE93-22, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA.
18. A. Schiper and A.M. Ricciardi. Virtually synchronous communication based on a weak failure suspector. In *23rd IEEE Fault-Tolerant Computing Symposium (FTCS)*, pages 534–543, June 1993.
19. Ilya Shnaiderman. Implementation of Reliable Datagram Service in the LAN environment. Lab project, The Hebrew University of Jerusalem, January 1999. http://www.cs.huji.ac.il/∼transis/publications.html.
20. R. Vitenberg, I. Keidar, G. V. Chockler, and D. Dolev. Group communication specifications: A comprehensive study. Technical Report CS99-31, Institute of Computer Science, Hebrew University, Jerusalem, Israel, September 1999. Also Technical Report MIT-LCS-TR-790, Massachusetts Institute of Technology, Laboratory for Computer Science and Technical Report CS0964, Computer Science Department, the Technion, Haifa, Israel.

Proceedings of the 3rd International Conference on Mobile Data Management, Singapore, January 2002, to appear.

An Approach to Disconnected Operation in an Object-Oriented Database

Sidney Chang and Dorothy Curtis

MIT Laboratory for Computer Science
`sidchang@alum.mit.edu, dcurtis@lcs.mit.edu`

Abstract. With the advent of mobile computers, new challenges arise for software designers. This paper focuses on disconnected operation: making mobile computers work well on shared data whether the network is available or not. Initially the shared data is cached on the mobile computer. Modifications and additions to this cached data will be reconciled with the shared data when the mobile computer is reconnected to the network. Conflict resolution will be used to reconcile conflicting changes. In this paper, we examine these issues by adding support for disconnected operation to Thor, an object-oriented database.

1 Introduction

As computers become more mobile, software needs to be adapted to work well, whether a network is available or not. The challenging aspect is to function without a network and still have local changes integrate easily with other data when the network is available. Initially the mobile computer needs to cache relevant data before disconnecting from the network. When network accessibility again becomes available, additions and modifications to the cached data need to be reconciled with the original data. Beyond this if two users have modified the same cached data while disconnected, some form of conflict resolution must be used to integrate these changes. This all must be done without violating serializability.

For example, a travelling salesperson commuting to the office would like to use his hand-held device to enter an appointment into his calendar which is stored in a central database at work. The hand-held device has a cached version of the salesperson's calendar which was downloaded the night before. Meanwhile back at the office, the salesperson's assistant is sitting in front of a desktop machine connected directly to the central database. The assistant enters an appointment into the salesperson's calendar for 2 PM to 4 PM, which is automatically updated in the central database. After the database is updated to reflect the assistant's change, the salesperson on his way to work now has stale data in his calendar. He thinks that he is free from 2 PM to 4 PM when he actually has an appointment for that time. To make things even worse, the salesperson, on his way to work, also enters an appointment into his calendar for 1 PM to 3 PM. This causes a conflict which must be resolved once the salesperson gets to work and reconnects his hand-held device to the database.

Several general properties become apparent from the example described. *Concurrent modifications to the same data may not always be undesirable.* In the case of the calendar, if the assistant had entered an appointment for 12 PM to 2 PM while the salesperson made an appointment for 3 PM to 5 PM, the same shared calendar data would be concurrently modified yet this would not violate the consistency of the data even though the salesperson modified his calendar while it contained stale data. *Conflicts are based on application semantics.* In the calendar, a conflict is an overlapping of appointments in time but could be completely different in another application. *From the user's perspective, automatic resolution of conflicts upon reconnection is desirable but may not always be possible; thus flexibility in resolving conflicts is important.* The salesperson may want all of the entries that he adds to the calendar to take precedence over others so his assistant's conflicting entry would have been deleted to make room for his entry. However there may also be special cases where the salesperson would not want this to apply.

1.1 Problem Statement

The problem of concurrency and shared data has been studied at length in the context of databases. The problem of shared data and disconnected operation has also been studied a great deal in the context of network partitions. But the problem of shared data and disconnected operation in the context of mobile devices changes because disconnections are more frequent and more predictable [12]. As a result, conflicts will be more likely. Therefore intelligent conflict resolution is necessary since the user of the mobile device will not want to lose all of the operations that he or she has performed while disconnected.

The problem that this paper addresses is how to build a system that manages shared data in the presence of disconnected operation. This system must address the issues of using stale data while disconnected and dealing with conflicting updates upon reconnection.

1.2 Achieving consistency

To achieve consistency a system can either use a *pessimistic* or an *optimistic* approach. In an optimistic scheme, users are allowed to modify shared data that may be concurrently accessed by other users. If an optimistic scheme is employed and users can be disconnected, some form of conflict resolution is required.

Pessimistic schemes prevent conflicts from occurring by permitting modifications only to that shared data for which the user has a lock. For the mobile computing setting, requiring the possession of locks to modify data limits the availability of data for other users.

Another aspect of achieving consistency with shared data, is maintaining serializability of data. Thus, a common way to handle concurrent accesses to shared data is to use transactions to aid in achieving both consistency and serializability of operations on shared data. Traditional properties of transactions are: atomicity, consistency, isolation, and durability (ACID) [10].

Various transaction models for mobile computing have been studied in [10, 5, 9]. Each is similar in that weaker forms of transactions (i.e. *weak*, *tentative*, or *second-class* transactions) are defined for transactions made on data local to a mobile device while disconnected. Using this weaker notion of transactions or *tentative transactions* allows for a system to have both consistency and an optimistic scheme in the presence of disconnections. While disconnected, tentative transactions operate on locally cached data. Each tentative transaction is logged at the disconnected device. Upon reconnection, each tentative transaction will either commit or abort as it is replayed against the shared data.

1.3 Flexible conflict resolution

Tentative transactions result in the need for intelligent conflict resolution. Since potentially all of the tentative transactions made while disconnected could be aborted, it is important that the system employ conflict resolution for those aborted transactions so that the disconnected user's operations are not lost.

Dealing with conflicts or aborts that occur upon reconnection is not simple. The problem is that resolution of conflicts is defined by application semantics and providing general support for a variety of applications is hard. Systems can automatically try to resolve conflicts. Another way to resolve the conflicts is to consult the user upon a failure. Conflict resolution can also be left up to the application since it is best aware of its own semantics. In the end, the most complete approach to resolving conflicts is a combination of system, application, and user support

Examples of such systems are Coda, Bayou, and Rover [7, 4, 6]. These systems will be discussed in greater detail in Section 5 and compared with the system presented here.

2 Thor overview

This project uses the Thor distributed object-oriented database system [8]. This paper proves the serializability of Thor and its ACID properties. In this section, an overview of the Thor architecture is presented.

2.1 Thor architecture

Thor provides a persistent store of objects where each object has a unique identity, set of methods on a per type basis, and state. The system has a client/server architecture where servers are repositories of persistent objects. The server or object repository (OR) consists of a root object plus all persistent objects that are reachable from the root object. The OR handles validation of transactions across multiple clients by using an optimistic concurrency control scheme described in [1]. Clients in Thor consist of a front end (FE) and an application. The FE handles caching of objects from the OR and transaction processing. Applications operate on cached objects at the FE inside transactions and commit transactions through the FE to the OR.

2.2 Objects in Thor

Each object is uniquely identified by an identifier known as an *oref.* Orefs are also used to locate an object within pages at the OR and FE. At the OR's objects are known only by their orefs. Objects at the FE are categorized as either persistent or non-persistent. Persistent objects are objects that the OR's are aware of and that are reachable from the persistent root object. Non-persistent objects are objects that are newly created by an application that have not yet been committed at the OR or objects that are used temporarily by the application and that will not need to persist across different runs of an application. Persistent objects at the FE are stored in the persistent cache which caches whole pages from the OR. An object in the persistent cache can be reached at the FE via its oref. Non-persistent objects are stored in the volatile heap and do not have orefs until they are committed and assigned oref's by the OR. To facilitate program access to objects at the FE in the persistent cache, orefs are mapped to local memory addresses.

2.3 FE transaction logging and committing transactions

Applications make high level operations on objects. These high-level operations on objects are reduced to reads, writes, and creations of objects. Each of these is logged by the FE in order to create the correct read, written, and created object sets to be sent to the OR in a commit request.

An application completes a transaction by making a request to the FE to commit the transaction. The FE processes this request by collecting all of the logged commit sets: the read object set (ROS), modified object set (MOS), and new object set (NOS). These sets are sent to the OR in the form of a commit request. The ROS and MOS will contain only persistent objects and the NOS will contain only those non-persistent objects created inside the transaction that are reachable from some persistent object. Before a NOS is sent to the OR it must contain orefs. The FE maintains some free pages for new orefs but in the event that there are no free orefs available, the OR is contacted to obtain new orefs.

To handle concurrency, an OR will validate a transaction based on whether or not that transaction read or wrote invalidated objects. The OR maintains a per-FE set of invalidated objects. These are objects whose state has become invalid since the time the FE cached them. An object at an FE is invalidated when another FE successfully commits a transaction modifying that object since the cached version is now stale. FE's are notified of invalidations and must acknowledge them by invalidating the objects in the persistent cache so that if those objects are ever accessed by the application, their new state will be fetched from the OR.

The OR can either commit or abort the FE's commit request. If the transaction is aborted by the OR, the FE must then roll back any of the changes made by the application. This includes reverting the state of modified objects back to the original state prior to the transaction and removing any newly created objects from the volatile heap. If the transaction is committed by the OR, the FE will move any newly created objects from the volatile heap to the persistent cache.

2.4 Summary of Thor

Thor provides transaction controlled access to shared data. Its optimistic concurrency scheme is appropriate for disconnected operation and its object-oriented nature should provide some benefits.

3 Disconnected operation in Thor

In the previous section, Thor was introduced. This section describes how we added disconnected operation to Thor [3].

The approach to disconnected operation in Thor is to use tentative transactions to manage shared data while disconnected and to provide a framework that enables the application to handle conflict resolution. The extension of Thor to support disconnected clients has two main aspects: extensions to the application and extensions to the FE (caching and per transaction processing).

3.1 FE support for disconnected operation

FE support for disconnected operation can be divided into three phases. The first is preparation for disconnection. The second is operating disconnected: handling transactions differently. The third is reconnecting with the OR: processing the pending transactions and the commits and aborts resulting from them.

Preparing for disconnect To prepare for disconnection from the OR, the FE needs to prefetch objects into the cache by processing queries specified by the application. The application may need a special prefetch query to ensure that all the relevant data is cached in the FE. This will be discussed in Section 3.2.

Operating disconnected Once the client has disconnected from the OR, an application will attempt to commit transactions as it normally would while connected. While disconnected, a commit becomes a *tentative commit* meaning that the commit could potentially be aborted by the OR upon reconnection. While disconnected, applications will operate the same as when they are connected by making operations on objects inside transactions. These operations will change the state of cached objects at the FE.

The FE logs tentative transactions in the tentative transaction log. This log saves enough state per tentative transaction in order to replay each tentative transaction once the FE is reconnected with the OR. The application is given an id for each tentative transaction that the application can associate with operations performed during that transaction. This information can be used later to assist the user in recovering from an abort. Figure 1 depicts an example tentative transaction log.

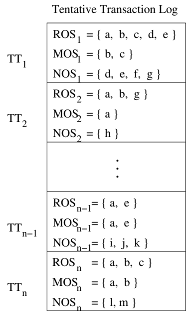


Fig. 1. Tentative Transaction Log

Tentative Transaction

The model used in connected Thor has the FE maintain transaction information on a per transaction basis. This information, also known as commit sets, consists of the read object set (ROS), the modified object set (MOS), and the new object set (NOS) created during a transaction. When the application commits the transaction, during connected operation, these sets are inserted directly into a commit request to the OR. But, while disconnected, these sets are maintained in the tentative transaction log.

The definitions of the commit sets change for tentative transactions. In a tentative transaction the ROS may consist of both persistent objects and objects that are *tentatively persistent.* An object is tentatively persistent if it was created by some tentative transaction that was tentatively committed but not yet committed at the OR. This also applies to the MOS in a tentative transaction: it can have both persistent and tentatively persistent objects. In Figure 1, TT_2, contains object g in ROS_2 since it is tentatively persistent from TT_1.

In a tentative transaction, commit sets must have all of their references to objects in oref format before they are sent to the OR. Temporary orefs are assigned to objects that are created by tentative transactions. In Figure 1, when TT_2 is stored into the tentative transaction log, references to g must be updated to the correct temporary oref assigned to it in TT_1.

In order to be able to handle the abort of a tentative transaction, each tentative transaction in the log must also save the state of each object in the MOS prior to its first modification. Objects in the MOS may be modified multiple times but only the initial state of the object before any modifications is saved in the log and only the state after the final modification in the duration of that transaction is saved in the MOS.

Reconnect When the FE reconnects with the OR, synchronization of the log occurs before the FE can proceed with any connected operations. Synchronization with the OR consists of replaying each tentative transaction in the order in which they were committed while disconnected, handling any aborts, and also handling invalidations.

Before sending a commit request to the OR, it is necessary to update temporary orefs in the NOS to permanent orefs. Permanent orefs are assigned either from free space in the current pages at the FE or by contacting the OR. In addition, the MOS and NOS may contain temporary orefs for tentatively persistent objects and these must be updated as well. Then the FE sends to the OR a commit request containing the commit sets stored for the tentative transaction.

The OR will then check if the commit request should be committed or aborted. The request will abort if an object in the read or write set of the transaction has been modified by another FE. When the FE receives the OR's response to the commit request it will process either a commit or an abort. On a commit the FE must install newly created objects from the tentative transaction into its persistent cache. On an abort, the FE uses the saved copies of modified objects to revert them back to their original state before the tentative transaction and then deletes newly created objects from the volatile heap.

When a tentative transaction is aborted, it is handled similarly to a connected abort. But, for a tentative transaction, in addition to reverting modified objects to their state before the tentative transaction, subsequent tentative transactions that depend on that tentative transaction must also be aborted. Tentative transaction TT_l depends on TT_k where $k < l$ if:

$$(ROS_l \cap MOS_k \neq \emptyset) \vee (MOS_l \cap MOS_k \neq \emptyset) \vee (ROS_l \cap NOS_k \neq \emptyset) \vee (MOS_l \cap NOS_k \neq \emptyset).$$

This defines dependency since TT_l can not be committed if it read or modified objects that were in an invalid state (as indicated by the abort of TT_k). Because the abort of TT_k causes the objects in NOS_k to be deleted, overall bookkeeping is simpler if transactions involving references to NOS_k are removed at the same time.

In checking for dependencies, if a subsequent tentative transaction in the log aborts due to its dependency on an aborted transaction, then any tentative transactions dependent on it must also abort. It is important to be careful about the order in which the state of objects in a tentative transaction's MOS are reverted to their saved state. For example, TT_k with $MOS_k = \{\text{a}\}$ aborts. TT_l with $MOS_l = \{\text{b}\}$ is dependent on TT_k because $ROS_l = \{\text{a,b}\}$. TT_m with $MOS_m = \{\text{b}\}$ is dependent on TT_l because $ROS_m = \{\text{b}\}$. So here is a chain of dependencies and after all of the dependent aborts have been processed, the state of object a should be as it was before TT_k and the state of object b should be as it was before TT_l. In the case of object b, it is important that its state be undone backwards, first to the saved state in TT_m and then to the state saved in TT_l. Therefore when aborts are processed, the dependencies are found in a forward scan but undoing the state of each is done in a backwards process through each of the dependent tentative transactions.

After the entire log of tentative transactions has been processed, invalidations are handled. In the process of replaying the tentative transactions, the FE may receive invalidation messages containing orefs of objects that have become stale. These stale objects must be marked invalid in the persistent cache.

After the FE processes all tentative transactions and invalidations, it must notify the application of any failures. It does this by returning to the application a set of tentative transaction id's containing the id of each tentative transaction that aborted.

Figure 2 depicts the reconnect process for a sample scenario. The tentative transaction log in this case contains three tentative transactions. TT_1 is aborted since some other FE made a modification to object a which this FE has not yet seen. Object a is in ROS_1 and MOS_1 so the OR must abort TT_1. The OR also sends an invalidation message for object a. Since $MOS_2 \cap NOS_1 \neq \emptyset$, the FE will automatically abort TT_2 without sending a commit request to the OR for it.

TT_3 is committed successfully. Then the FE processes the invalidation message and sends the acknowledgement to the OR. Finally the FE passes back to the application the list of tentative transaction id's that failed to commit.

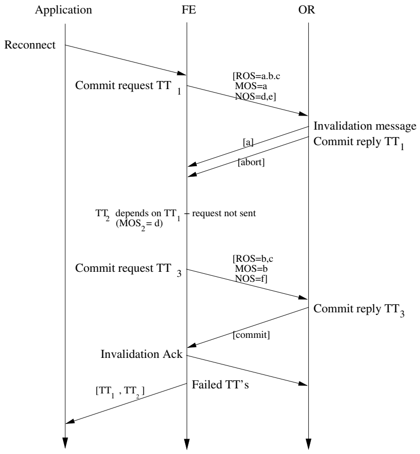


Fig. 2. Synchronizing the Log

3.2 Application support for disconnected operation

In addition to the support provided by the FE for disconnected operation, the application must also provide support for disconnected operation, namely support specific to application semantics. This support can be divided into the three components of preparing for disconnection, operating disconnected, and reconnecting. A specific example of an application and the support it provides will be discussed in Section 4.1.

Preparing for disconnect Application specific *hoarding queries* are used to prepare the client for disconnection from the OR. A hoarding query is an operation on the persistent objects in the database that causes objects to be fetched or hoarded from the OR into the FE cache prior to disconnecting from the OR.

Operating disconnected When disconnected, attempting to commit a transaction returns an id for the tentative commit. An application will use this id to identify data associated with the tentative transaction if it should abort. This contextual data can include operation type, parameters, or priority. Each operation type could also have an associated resolution function which attempts to use the saved parameters from the tentative transaction context to resolve a failure to commit. This extra support is necessary since Thor provides only a notification of conflicts and does no resolution itself.

Reconnect On reconnect, after the entire tentative transaction log has been replayed, the application receives a list of the id's of aborted transactions and then deals with their resolution. It does this by iterating through the list of failed transactions and calling their appropriate resolution functions. In the process of calling a resolution function, it is possible that the transaction will be aborted again and a series of nested calls to resolution functions and aborts may occur.

To resolve a conflict, the application has the flexibility to do a variety of resolutions since the application has control over where conflicts are detected and also has saved context for each transaction. While Thor provides conservative abort semantics that guarantee serializability of operations on shared objects, successful retries of a failed tentative transaction actually allow applications that do not require Thor's conservative abort semantics to achieve more relaxed semantics.

4 Evaluation

This section discusses how well disconnected operation in Thor achieves the goals of consistent shared data and flexible conflict resolution through the development of a sample application on top of the Thor framework. In addition, an analysis of performance is presented to discuss the overhead from disconnected operation.

4.1 Sample client program: a shared calendar

A shared calendar system was implemented as an application using the disconnected Thor framework described in Section 3. The calendar system maintains a database of calendars where each calendar is associated with a user but multiple users may modify a single calendar. Concurrent modifications to a single calendar are possible. A user can add and delete events to and from a calendar. Each event has an associated day and time.

The essential aspect of the design of the calendar application was the data modelling phase or development of the application's schema. The schema is the organization or structure of the data as represented in the database. In the data modelling phase it is important to consider the effect of concurrency on the data. It is especially important in Thor since conflicts are detected at the granularity of an object and therefore the design of the object-oriented schema will directly impact what conflicts are detected. In the calendar application, concurrent additions of events to a user's calendar are permissible so long as they do not overlap in time. The correct behavior is for a conflict to be detected only when concurrent updates to the calendar modify events that conflict in time. However, these conflicts are not always significant and in some situations, a user may want more relaxed semantics. These situations can be accommodated with the flexible resolution of conflicts.

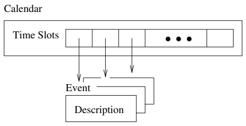


Fig. 3. Calendar Schema

To achieve the correct calendar conflict semantics in the calendar application, the schema was designed to detect conflicts at the granularity of time slot objects rather than the entire calendar object. This is depicted in Figure 3 where the calendar is a set of time slot objects and each time slot can be assigned to some event. With this design, if two users concurrently modify the same time slot object, then a conflict will be detected by Thor. This is the correct semantics for a conflict in a calendar, namely when two appointments are made for overlapping times. However since the user may want to allow this at times, it is important to consider the conflict resolution and the different properties that a user might want to be able to have in his calendar.

With the described design of the calendar, we maintain consistency in addition to getting the correct conflict semantics. Consistency is maintained since multiple users can concurrently add events to the calendar without having a conflict as long as a transaction does not read stale objects in the calendar. An additional factor to consider in the calendar application design is that a transaction that writes an event should be careful not to include reads of other time slots. This requires that the application developer be very careful in the organization of commit points in the application. For example, in a transaction that adds an event to the calendar, the application should not also read all of the objects in the calendar, since this will increase the likelihood of an abort.

Conflict resolution in the calendar application is flexible since it is possible for the application to have control over where conflicts are detected. In the case of the calendar, we know that conflicts are over time slots, so if a conflict occurs, we know it is because another user has already modified that same time slot. It is then up to the application to deal with this conflict. In order to be able to deal with a conflict the application needs to understand the context for a transaction. Therefore, as discussed in section 3.2, the calendar application saves the high-level operations made by the transaction and any arguments to the operations.

Using the saved context and having fine-grained control over conflict detection through the design of the application schema, any number of policies can be implemented to resolve conflicts.

4.2 Performance

The overall performance of the Thor system is discussed in [1]. Thor compares favorably with similar systems in terms of throughput and scalability. This section discusses the added overhead of supporting disconnected operation. First, the number of tentative transactions is limited by the amount of memory in the client. The overhead varies by the number of tentative transactions in the log, the read:write:new object ratio in the commit sets, and the level of contention or percentage of aborts for a given number of tentative transactions in the log. The remainder of this section will discuss the overhead of disconnected operation in Thor using experiments based on the OO7 benchmark which provides a comprehensive test of object-oriented database management system performance. The details of this benchmark are described in [2].

In comparison to connected operation in Thor, the design of disconnected operation in Thor has several differences that affect performance. These differences occur both while operating disconnected and upon reconnection. Experiments were conducted with a single FE and OR. The OR was run on a 400 Mhz dual Pentium II with 128 MB of RAM running the Linux Redhat distribution 6.2. The FE was run a 450 Mhz Pentium II with 128 MB of RAM running Linux Redhat distribution 6.2. The FE cache size for all experiments was 24 MB. All communication between the two machines was on an isolated network so that variations in network traffic would not affect the experimental results.

Each experiment uses an OO7 traversal to compare connected with disconnected operation. The difference between disconnected and connected operation occurs at commit points. During connected operation, the application simply waits for the response to a commit request from the OR. In disconnected operation, a commit has two parts. The first part is to tentatively commit the transaction while disconnected. This places the transaction in the tentative transaction log. The second part is to reconnect and send the tentative transaction from the log to the OR as a commit request.

Upon reconnection, replay of the log incurs two major costs that do not occur in connected commits. The first overhead incurred is, in preparation to send the tentative transaction as a commit request to the OR, newly created objects that have temporary orefs must be updated to have new permanent orefs. Getting permanent orefs is a cost that is also incurred during connected commits, however objects in the MOS and NOS need to be updated with these new orefs. This updating incurs the extra cost of a second traversal of the objects in the MOS and NOS for all tentatively committed transactions in the log.

We evaluated the average overhead to be 36.32% for updating temporary orefs on logs ranging from 10-100 tentative transactions with a workload of an OO7 insertion query with 5 new composite objects and 2 modified objects per transaction. The growth of the time to tentatively commit and reconnect is linear with respect to the number of tentative transactions. However it does grow at a faster rate than connected commits. This is due to an increasing number of permanent orefs to search through when replacing temporary orefs with permanent ones.

The second major source of overhead from disconnected operation comes from aborts. If a transaction is aborted, the log must be updated to abort any dependent transactions. This dependency check has the extra cost of scanning the log with a backwards undo (as described in Section 3.1) each time an abort happens.

Experiments were conducted in both low and moderate contention (abort rate) environments similar to experiments made by Adya for concurrency control studies in Thor [1]. The experiments make use of the OO7 T2a traversal rather than the Tnew since the Tnew traversal creates dependent tentative transactions. By using the T2a traversal

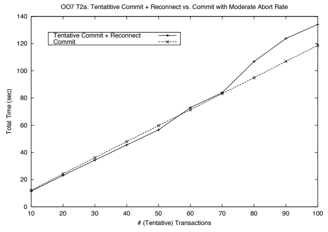


Fig. 4. Abort Overhead (20% abort rate)

both the low and moderate abort rate experiments have no dependencies between transactions to show the case of maximal scanning. Figure 4 shows the results for an OO7 T2a traversal with a 20% abort rate which is considered moderate contention. The results for a low 5% abort rate show similar trends.

5 Related work

Providing consistent shared data in the presence of disconnected operation is not a new problem. Researchers have analyzed the issues and systems have been implemented that support disconnected operation and the sharing of data.

The major relevant analytical work is [11]. Disconnected operation in Thor largely implements the behavior described in the caching example in this paper: the mobile computer performs weak transactions while disconnected. These transactions are committed only if they do not conflict with the strict transactions at the server. Beyond actually implementing this model, we have started to understand how applications can make use of this system.

Some implementations include Bayou, Rover, and Coda as mentioned in Section 1.3. Having discussed the design of disconnected operation in Thor and evaluated its effectiveness in achieving consistent shared data with flexible conflict resolution, this section visits each of the systems described in Section 1.3 to see how they compare. In general each system uses a similar notion of "tentative" data for data modified while disconnected but has different methods for handling concurrency and conflicts.

Coda supports disconnected operation but it is oriented around a file system. Conflicts are detected only at the granularity of files which gives an application much less control over the semantics of conflicts.

Thor on the other hand, can be used for a variety of applications where data easily fits into an object model where objects are small. However if an application is concerned over file-sharing such as in a collaborative document editing system, Coda may actually be a more suitable choice.

Both Bayou and Rover do not provide for any built-in notion of consistency. It is up to the application to define in its procedures, checks for conflicts and the procedures to resolve them. Thor takes some of the burden of this away from the application by having built-in conflicts detected on objects. While it is true that the application does play a role in defining conflicts since the application schema must be carefully designed to achieve the correct conflict semantics, Thor provides a framework with which the application can work. In addition this framework is a familiar one since it is essentially the framework of an object-oriented programming language.

Bayou and the approach to disconnected operation in Thor are similar in that application-specific conflict detection and resolution are facilitated. Bayou's dependency-check procedure is analogous to schema design in Thor since the manner in which the schema is designed, controls what conflicts are detected. Bayou's merge-proc function is analogous to application conflict resolution in Thor. The difference between the two is that there is no built in notion of consistency in Bayou. While Thor allows for an application to have control over where conflicts will be detected, the serializability of data will not be violated at any point. Thor could perhaps benefit from Bayou's notion of merge-procs. Since Thor applications must now include all conflict resolution code inside the application, it would be beneficial to add to Thor, a framework for applications to write resolution functions or perhaps even select from a set of common resolution functions.

6 Conclusion

This paper has described a system that can, with more experimentation, be extended to support a variety of applications. These applications will behave well using shared data whether the network is available or not.

Disconnected operation in Thor suits a variety of applications since it can provide strict consistency rules for applications that require them such as a banking system or airline reservation system. Yet, with the framework provided, it also allows applications with more relaxed consistency requirements to have enough control over conflicts and their resolution to achieve more flexible consistency semantics.

7 Acknowledgements

The authors are grateful for support from members of the MIT Project Oxygen partnership: Acer, Delta, Hewlett Packard, NTT, Nokia, and Philips and from DARPA through the Office of Naval Research contract number N66001-99-2-891702. We also wish to thank Chandra Boyapati, Liuba Shrira, Hong Tian, John Ankcorn, Steve Garland, Paul Kim, and Hari Balakrishnan for their thoughtful suggestions.

References

1. Atul Adya, Robert Gruber, Barbara Liskov, and Umesh Maheshwari. Efficient optimistic concurrency control using loosely synchronized clocks. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 23–34, San Jose, CA, US, 1995.
2. M.J. Carey, D.J. DeWitt, and J.F. Naughton. The oo7 benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 12–21, Washington D.C., May 1993.
3. Sidney Chang. Adapting an object-oriented database for disconnected operation. Master's thesis, Massachusetts Institute of Technology, 2001.
4. A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch. The bayou architecture: Support for data sharing among mobile users. In *Proceedings IEEE Workshop on Mobile Computing Systems & Applications*, pages 2–7, Santa Cruz, California, 8-9 1994.
5. Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *ACM SIGMOD International Conference on Management of Data*, pages 173–182, Montreal, Canada, 1996.
6. Anthony D. Joseph, Joshua A. Tauber, and M. Frans Kaashoek. Mobile computing with the rover toolkit. *IEEE Transactions on Computers: Special issue on Mobile Computing*, 1997.
7. J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *Thirteenth ACM Symposium on Operating Systems Principles*, pages 213–225, Asilomar Conference Center, Pacific Grove, U.S., 1991. ACM Press.
8. Barbara Liskov, Miguel Castro, Liuba Shrira, and Atul Adya. Providing persistent objects in distributed systems. In Rachid Guerraoui, editor, *ECOOP '99 — Object-Oriented Programming 13th European Conference, Lisbon Portugal*, volume 1628, pages 230–257. Springer-Verlag, New York, NY, 1999.
9. Q. Lu and M. Satyanaranyanan. Isolation-only transactions for mobile computing. *Operating Systems Review*, 28(2):81–87, 1994.
10. Evaggelia Pitoura and Bharat Bhargava. Revising transaction concepts for mobile computing. In *Proc. of the First IEEE Workshop on Mobile Computing Systems and Applications*, pages 164–168, Santa Cruz, CA, December 1994.

11. Evaggelia Pitoura and Bharat Bhargava. Data consistency in intermittently connected distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(6):896–915, 1999.
12. Evaggelia Pitoura and George Samaras. *Data Management for Mobile Computing*, chapter 3, pages 37–70. Kluwer Academic Publishers, 1998.

Proceedings of the 17th ACM SOSP,
Kiawah Island, SC, December 1999.

The Design and Implementation of an Intentional Naming System

William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan
and Jeremy Lilley

MIT Laboratory for Computer Science
{`wadjie, elliot, hari, jjlilley`}`@lcs.mit.edu`

Abstract. This paper presents the design and implementation of the Intentional Naming System (INS), a resource discovery and service location system for dynamic and mobile networks of devices and computers. Such environments require a naming system that is (i) expressive, to describe and make requests based on specific properties of services, (ii) responsive, to track changes due to mobility and performance, (iii) robust, to handle failures, and (iv) easily configurable. INS uses a simple language based on attributes and values for its names. Applications use the language to describe what they are looking for (i.e., their *intent*), not where to find things (i.e., not hostnames). INS implements a *late binding* mechanism that integrates name resolution and message routing, enabling clients to continue communicating with end-nodes even if the name-to-address mappings change while a session is in progress. INS resolvers self-configure to form an application-level overlay network, which they use to discover new services, perform late binding, and maintain weak consistency of names using soft-state name exchanges and updates. We analyze the performance of the INS algorithms and protocols, present measurements of a Java-based implementation, and describe three applications we have implemented that demonstrate the feasibility and utility of INS.

1 Introduction

Network environments of the future will be characterized by a variety of mobile and wireless devices in addition to general-purpose computers. Such environments display a degree of dynamism not usually seen in traditional wired networks due to mobility of nodes and services as well as rapid fluctuations in performance. There is usually no pre-configured support for describing, locating, and gaining access to available services in these heterogeneous, mobile networks. While the packet routing problem in mobile networks has been extensively researched [6, 32], the important functions of resource discovery and service location are only recently beginning to receive attention in the research community. We believe that this is an important problem to solve because the cost of deploying and running such a network infrastructure is dominated by software and service management, while diminishing hardware costs are making it cheap to network all sorts of devices. Examples of applications in such environments include sending a job to the closest (based on geographic location) and least-loaded printer, retrieving files from a mobile, replicated server based on network performance and server load, and retrieving the current image from all the mobile cameras in a particular section of a building.

Based on our target environment and applications, we identify the following important design goals for a naming system that enables dynamic resource discovery and service location:

- **Expressiveness.** The naming system must be flexible in order to handle a wide variety of devices and services. It must allow applications to express arbitrary service descriptions and queries.
- **Responsiveness.** The naming system must adapt quickly to end-node and service mobility, performance fluctuations, and other factors that can cause a change in the “best” network location of a service.
- **Robustness.** The naming system must be resilient to name resolver and service failures as well as inconsistencies in the internal state of the resolvers.
- **Easy configuration.** The name resolvers should configure themselves with minimal manual intervention and the system should not require manual registration of services. The resulting system should automatically distribute request resolution load among resolvers.

The main contribution of our work is the design and implementation of INS, an Intentional Naming System that meets the above goals. Because applications in our environment (as in many distributed environments) often do not know the best network location that satisfies their needs for information or functionality, we argue in favor of an *intentional* naming scheme and resolution architecture in which applications describe *what* they are looking for, not *where* to find it. Name resolvers in the network route requests to the appropriate locations by maintaining a mapping between service descriptions and their network locations.

INS achieves expressiveness by using an intentional name language based on a hierarchy of attributes and values, allowing nodes that provide a service to precisely describe what they provide and consumers to easily describe what they require. Names based on attributes and values have been suggested before in other contexts [5, 7, 13, 45] and we draw upon previous work in this area in designing our naming language. While several complex query languages based on attributes and values exist in the literature, ours is particularly simple and has a small set of commonly used operators, which makes it lightweight and easy to implement even on impoverished devices. We also design the INS resolution architecture to be independent of the specific language used to perform queries, so that it can also be used in the context of other service description languages.

An important characteristic of our target environment is mobility, where the network location (e.g., IP address) of an end-node changes. *Node mobility* may occur due to physical mobility or when a node changes the network used to communicate (e.g., changing from an wired Ethernet to a radio frequency network). Another form of mobility is *service mobility,* where the network addresses of a service does not change, but the end-nodes mapping to a service change because of a change in the values of the attributes sought by clients. In addition, our environment is dynamic because of performance fluctuations—as the load on service nodes and paths in the network changes, so does the location of the best node to serve each client request. Hence, INS should reflect performance changes in the results of name resolution.

In INS, clients use an intentional name to request a service without explicitly listing the end-node(s) that ultimately serve the request. This "level of indirection" provided by an intentional name allows applications to seamlessly continue communicating with end-nodes even though the mapping from name to end-node addresses may change during the session, transparent to the client. Thus, INS supports mobile applications, which use intentional names rather than IP addresses.

INS achieves responsiveness by *integrating* name resolution and message routing, operations that have traditionally been kept separate in network architectures. INS applications benefit from this abstraction using a *late binding* option, where the binding between the intentional name and network location(s) is made at message delivery time rather than at request resolution time. This binding is "best-effort" since INS provides no guarantees on reliable message delivery. Thus, INS uses an intentional name not only to locate services, but also to route messages to the appropriate end-points. This integration leads to a general method for performing application-level routing using names, effected by applications including data with the name resolution query.

Our integrated routing and resolution system provides two basic types of message delivery service using late binding. An application may request that a message be delivered to the "optimal" service node that satisfies a given intentional name, called *intentional anycast.* Here, the metric for optimality is application-controlled and reflects a property of the service node such as current load. A second type of message delivery, *intentional multicast,* is used to deliver data to *all* service nodes that satisfy a given name, for example, the group of sensors that have all recorded sub-zero temperatures. These two delivery services allow INS to achieve *application-level* anycast and multicast.

In keeping with the end-to-end principle [37], we leave the underlying network-layer addressing and routing of the IP architecture unchanged. Rather, our approach to providing these services is to layer them as an overlay network over unicast IP. The only network layer service that we rely upon is IP unicast, which is rapidly becoming ubiquitous in mobile and wireless environments[1].

Another reason for leaving the core infrastructure unmodified is that often, a network-layer service does not perfectly match the requirements of the application at hand. Indeed, performing anycast on a specific network-layer criterion such as hop-count, network latency or available bandwidth, is ineffective from the point of view of many applications because it does not optimize the precise metric that applications require. For example, a network-layer anycast [31] to find the "best" printer on a floor of a building cannot locate the *least-loaded* printers. To remedy this, INS allows intentional anycast based on *application-controlled* metrics, where resolvers select the least value according to a metric that is meaningful to and advertised by applications.

Despite allowing application-controlled routing metrics, INS presents a simple and well-defined service model for intentional anycast and multicast. In contrast to the active networks architecture [41, 46] and their naming counterpart, ActiveNames [43], where arbitrary code and services may be injected into the data path to customize the functions of an IP router or name resolver, INS resolvers do not run arbitrary code nor embed any application-specific semantics in the routing and resolution architecture. Instead, our system relies on structured names to express application parameters. This decision to leave IP unicast unmodified is based on the difficulties encountered in deploying other IP

---

[1] Note that we do not rely on Mobile IP [32] in INS.

extensions, for example, IP multicast [12], guaranteed services [10], and more recently, active IP networks [46]. In this sense, one may view the INS architecture as similar in philosophy to application-level anycast [3] and Web server selection, which have recently gained in popularity.

INS uses a decentralized network of resolvers to discover names and route messages. To ease configuration, INS resolvers self-configure into an application-level overlay network and clients can attach to any of them to resolve their requests and advertise services. These resolvers use *soft-state* [9] periodic advertisements from services to discover names and delete entries that have not been refreshed by services, eliminating the need to explicitly de-register a service. This design gracefully handles failures of end-nodes and services. They also implement load-balancing and load-shedding algorithms, which allows them to scale well to several thousand services.

The INS resolution architecture presented in this paper makes three key contributions: (i) it integrates resolution and routing, allowing applications to seamlessly handle node and service mobility, and provides flexible group communication using an intentional name as the group handle; (ii) its resolvers self-configure into an overlay network and incorporate load-balancing algorithms to perform well; and (iii) it maintains weak consistency amongst replicated resolvers using soft-state message exchanges. These features distinguish it from other service discovery proposals made in the recent past, including the IETF Service Location Protocol (SLP) [44, 33], Sun's Jini service discovery [21], the Simple Service Discovery Protocol [19], universal plug-and-play [42], and Berkeley's service discovery service [11].

An important feature of our architecture is its potential for incremental and easy deployment in the Internet, without changing or supplanting the existing Internet service model. INS is intended for dynamic networks on the order of several hundred to a few thousand nodes, many of which could be mobile (e.g., inside a single administrative domain, building, office or home network). We note, however, that the architecture presented in this paper is not directly applicable in the wide-area Internet. We are currently developing a wide-area architecture to complement this intra-domain INS architecture. However, despite this cautionary note, our performance results show that our resolution algorithms and load-balancing strategies permit a network of INS resolvers to scale to several thousand names and services. Our experimental results show that the time to discover new names is on the order of tens of milliseconds. We find that the time to process name updates is the performance bottleneck in many cases, and describe a technique to partition the namespace amongst resolvers to alleviate this problem.

To demonstrate the utility of INS, we describe its programming interface and the implementation of three applications: *Floorplan*, a map-based service discovery tool for location-dependent services, *Camera*, a mobile camera network for remote surveillance, and *Printer*, a load-balancing printer utility that sends user print requests to the best printer based on properties such as physical location and load. These applications use the INS API and support for mobility, group communication, and service location, gaining these benefits without any other pre-installed support (e.g., Mobile IP [32], IP multicast [12], SLP [44], etc.) for these features.

The rest of this paper is organized as follows. We describe the INS architecture in Section 2, the API and some applications in Section 3, our implementation in Section 4, its performance in Section 5, related work in Section 6, and our conclusions in Section 7.

2 System architecture

INS applications may be *services* or *clients*: services provide functionality or data and clients request and access these. *Intentional Name Resolvers (INRs)* route client requests to the appropriate services, implementing simple algorithms and protocols that may be implemented even on computationally impoverished devices. Any device or computer in an *ad hoc* network can potentially act as a resolver, and a network of cooperating resolvers provides a system-wide resource discovery service.

INRs form an application-level overlay network to exchange service descriptions and construct a local cache based on these advertisements. Each service attaches to an INR and advertises an attribute-value-based service description and an application-controlled metric. Each client communicates with an INR and requests a service using a query expression. Because service descriptions are disseminated through the INR network in a timely manner, a new service becomes known to the other resolvers and through them to the clients.

When a message arrives at an INR, it is resolved on the basis of the destination name. The INR makes a resolution/forwarding decision depending on the specific service requested by the client application. If the application has chosen early binding (selected using the *early-binding flag* in the request), the INR returns a list of IP addresses corresponding to the name. This is similar to the interface provided by the Internet Domain Name System (DNS) [27] and most other existing service discovery systems, and is useful when services are relatively static. When

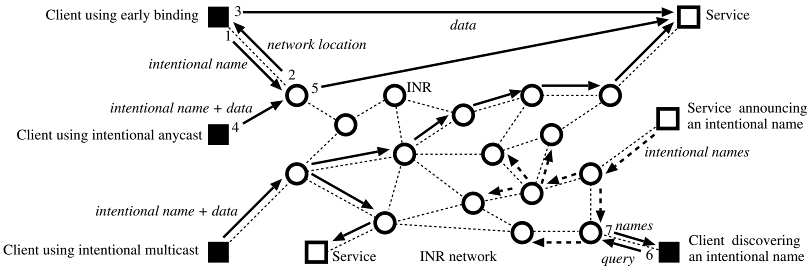


Fig. 1. The architecture of the Intentional Naming System. The upper-left corner shows an application using early binding: the application sends an intentional name to an INR to be resolved (1), receives the network location (2), and sends the data directly to the destination application (3). The middle-left shows an application using intentional anycast—the application sends an intentional name and the data to an INR (4), which tunnels to exactly one of the destinations that has the least metric (5). The lower-left corner shows an application using intentional multicast: the application sends an intentional name and the data to an INR, which forwards it through the INR network to all of the destination applications. The lower-right corner shows an application announcing intentional names to an INR. The intentional names are beginning to propagate throughout the INR network. An application discovering names sends a query to an INR (6), receives a set of names that match the name in query.

there are multiple IP addresses corresponding to a name, the client may select an end-node with the least metric, which is available from the result of the resolution request. This metric-based resolution is richer than round-robin DNS resolution.

INS applications use the two late binding options—intentional anycast and intentional multicast—to handle more dynamic situations. Here, the network addresses are not returned to the client, but instead, the INR forwards the name and the associated application payload directly to the end-nodes (e.g., services). If the application requests intentional anycast, the INR tunnels the message to exactly one of the end-nodes in its list that has the least metric. In INS, this metric does not reflect a network-layer metric such as hop-count used in network-layer anycast [31]; rather, INS allows applications to advertise arbitrary application-specific numeric metrics such as average load. In intentional multicast, the INR forwards each message to all next-hop INRs associated with the destination name. The message is forwarded along the INR overlay to all the final destination nodes that match the name.

INRs self-configure into a spanning-tree overlay network topology, optimizing the average delay between neighboring INRs. In constructing this overlay topology, we use measurements of INR-to-INR round-trip time. The spanning tree is used to disseminate service descriptions as well as receiver subscriptions. Unlike other overlay networks that maintain pre-configured, static neighbors such as the MBone [14] or the 6Bone [17], INRs can be spawned or terminated and automatically adjust their neighbor relationships based on network conditions. They also implement load-balancing algorithms to perform better, by spawning new resolvers on other nodes when the incoming request rate is high and delegating portions of the namespace to the newly spawned instances.

Figure 1 summarizes the architecture of INS, illustrating how applications and INRs interact.

2.1 Name-specifiers

INS implements intentional names using expressions called *name-specifiers*. Clients use name-specifiers in their message headers to identify the desired destinations (and sources) of messages. Name-specifiers are designed to be simple and easy to implement. The two main parts of the name-specifier are the *attribute* and the *value*. An attribute is a category in which an object can be classified, for example its 'color.' A value is the object's classification within that category, for example, 'red.' Attributes and values are free-form strings[2] that are defined by applications; name-specifiers do not restrict applications to using a fixed set of attributes and values. Together, an attribute and its associated value form an *attribute-value pair* or *av-pair*.

A name-specifier is a hierarchical arrangement of av-pairs such that an av-pair that is *dependent* on another is a descendant of it. For instance, in the example name-specifier shown in Figure 2, a building called the Whitehouse is meaningful only in the context of the city of Washington, so the av-pair `building=whitehouse` is dependent on the av-pair `city=washington`. Av-pairs that are *orthogonal* to each other but dependent on the same av-pair, are siblings in the tree. For example, a digital camera's data-type and resolution can be selected

---

[2] Attributes and values being free-form strings is not a fundamental property; fixed length integers could be used just as easily if the bandwidth or processing power required for handling names is a concern. Not having human readable strings makes debugging more difficult, but does not affect normal use of the system, since applications still need to understand the semantics of attribute and values to present them to users.

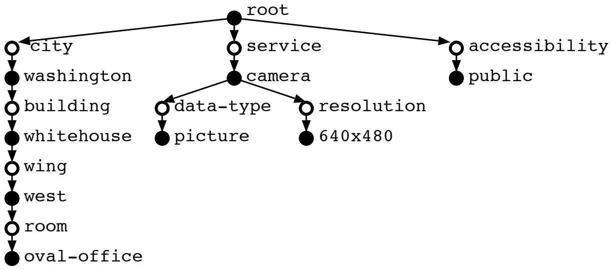


Fig. 2. A graphical view of an example name-specifier. The hollow circles are used to identify attributes; the filled circles identify values. The tree is arranged such that dependent attributes are descendants, and orthogonal attributes are siblings. This name-specifier describes a public-access camera in the Oval office.

independently of each other, and are meaningful only in the context of the camera service. Therefore, the av-pairs `data-type=picture` and `resolution=640x480` are orthogonal. This hierarchical arrangement narrows down the search space during name resolution, and makes name-specifiers easier to understand.

A simpler alternative would have been to construct a hierarchy of attributes, rather than one of av-pairs. This would result in `building` being directly dependent on `city`, rather than `city=washington`. However, it is also less flexible; our current hierarchy allows child attributes to vary according to their parent value. For example, `country=us` has a child that is `STATE=virginia`, while `country=canada` has a child that is `PROVINCE=ontario`.

Name-specifiers have a representation (Figure 3) that is used when they are included in a message header to describe the source and destination of the message. This string-based representation was chosen to be readable to assist with debugging, in the spirit of other string-based protocols like HTTP [16] and NNTP [22]. Levels of nesting are indicated by the use of brackets (`[` and `]`), and attributes and values are separated by an equals sign (`=`). The arbitrary use of whitespace is permitted anywhere within the name specifier, except in the middle of attribute and value tokens.

In addition to exact-value matches, name-specifiers also permit wild-card matching of values. To do this, the value is simply replaced by the wild-card token (`*`). Thus to construct a name-specifier that refers to *all* public cameras providing 640x480 pictures in the West Wing of the Whitehouse, not just the one in the Oval Office, an application replaces

```
[city = washington [building = whitehouse
                     [wing = west
                     [room = oval-office]]]]
[service = camera    [data-type = picture
                         [format = jpg]]
                     [resolution = 640x480]]
[accessibility = public]
```

Fig. 3. The wire representation of the example name-specifier shown in Figure 2, with line-breaks and extra spacing added to improve readability.

the value `oval-office` with '`*`' in the name-specifier shown in Figures 2 and 3. We are currently incorporating inequality operators ($<$, $>$, $\leq$, and $\geq$) to provide range selection operations in name-specifiers, to augment the matches described above.

2.2 Discovering names

Services periodically advertise their intentional names to the system to describe what they provide. Each INR listens to these periodic announcements on a well-known port to discover services running at different end-nodes. INRs replicate and form an overlay network among themselves, over which they send updates of valid names in the system.

The name discovery protocol treats name information as soft-state [9, 35], associated with a lifetime. Such state is kept alive or refreshed whenever newer information becomes available and is discarded when no refresh announcement is received within a lifetime. Rapid changes due to node mobility quickly propagate through the system and new information automatically replaces old, outdated information. Soft-state enables robust operation of the system since INRs can recover from errors and failures automatically without much disruption because incorrect information is updated by new messages. This choice allows a design where applications may join and leave the system without explicit registration and de-registration, because new names are automatically disseminated and expired names automatically eliminated after a timeout.

When clients make name resolution requests, INRs use the information obtained using service advertisements and INR updates to resolve them. In addition to sending resolution requests, clients can discover particular types of names or all known names in the system by sending a *name discovery* message to an INR, specifying an intentional name for the INR to match with all the names it knows about. This mechanism is useful for clients to bootstrap in a new environment.

INRs disseminate name information between each other using a routing protocol that includes *periodic* updates and *triggered* updates to their neighbor INRs. Each update contains the following information about a name-specifier:

- The IP addresses for this name-specifier and a set of `[port-number, transport-type]` pairs for each IP address. The port number and transport type (e.g., HTTP [2], RTP [38], TCP [34], etc.) are returned to the client to allow it to implement early binding.
- An application-advertised metric for intentional anycast and early binding that reflects any property that the service wants anycast routing on, such as current or average load.
- The next-hop INR and the metric, currently the INR-to-INR round-trip latency in the overlay network for the route, used for intentional multicast.
- A unique identifier for the announcer of the name called the *AnnouncerID*, used to differentiate identical names that originate from two different applications on the same node.

INRs use periodic updates to refresh entries in neighboring INRs and to reliably flood names. Triggered updates occur when an INR receives an update from one of its neighbors (either an INR or a client or service) that contains new information (e.g., a newly discovered name-specifier) or information that is different from the one previously known (e.g., better metric)[3].

For applications requiring intentional multicast, INRs forward the name and payload message through the overlay network to all of the network locations that announce a given name. In our current implementation, INRs use the distributed Bellman-Ford algorithm [1] to calculate shortest path trees to those end-nodes announcing the name. Unlike traditional routing protocols that use the algorithm [26], the INS architecture allows multiple identical names to exist in the system. The unique AnnouncerID ensures that routes to identical names can be differentiated. In our implementation, applications generate an AnnouncerID by concatenating their IP address with their startup time, allowing multiple instances to run on the same node.

---

[3] For inter-INR communications we could have had the INRs use reliable TCP connections and send updates only for entries that change, perhaps eliminating periodic updates at the expense of maintaining connection state in the INRs. We do not explore this option further in this paper, but intend to in the future.

2.3 Name lookup and extraction

The central activity of an INR is to resolve name-specifiers to their corresponding network locations. When a message arrives at an INR, the INR performs a lookup on the destination name-specifier in its name-tree. The lookup returns a *name-record*, which includes the IP addresses of the destinations advertising the name as well as a set of "routes" to next-hop INRs. The late binding process for anycast and multicast do not change the name-specifiers or data in any way. By integrating resolution with routing in the late binding process, INS enables seamless communication between clients and services even if the name-to-location mapping changes during the session.

The rest of this section describes how names are stored at an INR, lookups are performed on an incoming name, and how names are extracted for periodic or triggered updates in the name discovery protocol.

Name-trees Name-trees are a data structure used to store the correspondence between name-specifiers and name-records. The information that the name-records contain are the routes to the next-hop INRs, the IP addresses of potential final destinations, the metric for the routes, end-node metrics for intentional anycast, and the expiration time of the name-record.

The structure of a name-tree bears a close resemblance to a name-specifier. Like a name-specifier, it consists of alternating levels of attributes and values, but unlike a name-specifier there can be multiple values per attribute, since the name-tree is a superposition of all the name-specifiers the INR knows about. Each of these name-specifiers has a pointer from each of its leaf-values to a name-record. Figure 4 depicts an example name-tree, with the example name-specifier from Figure 2 in bold.

Name lookup The LOOKUP-NAME algorithm, shown in Figure 5, is used to retrieve the name-records for a particular name-specifier n from the name-tree T. The main idea behind the algorithm is that a series of recursive calls reduce the candidate name-record set S by intersecting it with the name-record set consisting of the records pointed to by each leaf-value-node. When the algorithm terminates, S contains only the relevant name-records.

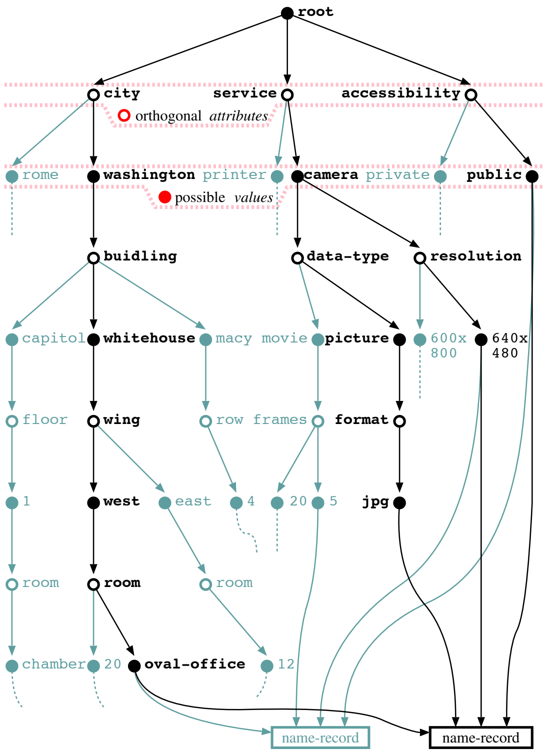


Fig. 4. A partial graphical view of an example INR name-tree. The name-tree consists of alternating layers of attribute-nodes, which contain orthogonal attributes, and value-nodes, which contain possible values. Value-nodes also contain pointers to all the name-records they correspond to. The part of the name-tree corresponding to the example name-specifier shown in Figure 2 is in bold.

The algorithm starts by initializing S to the set of all possible name-records. Then, for each av-pair of the name-specifier, it finds the corresponding attribute-node in the name-tree. If the value in the av-pair is a wild card, then it computes S' as the union of all name-records in the subtree rooted at the corresponding attribute-node, and intersects S with S'. If not, it finds the corresponding value-node in the name-tree. If it reaches a leaf of either the name-specifier or the name-tree, the algorithm intersects S with the name-records at the corresponding value-node. If not, it makes a recursive call to compute the relevant set from the subtree rooted at the corresponding value-node, and intersects that with S.

This algorithm uses the assumption that omitted attributes correspond to wild-cards; this is true for both queries and advertisements. A nice property of the algorithm is that it executes in a single pass without any backtracking. This also means that wild-cards should be used only on the leaf values (any av-pairs after a wild-card will be ignored).

Section 5.1 analyses this algorithm and discusses the experimental results of our implementation.

Name extraction To send updates to neighboring INRs, an INR needs to get name-specifiers from its name-tree to transmit. Since the name-tree is a superposition of all the name-specifiers the INR knows about, extracting a single name-specifier is non-trivial. The GET-NAME algorithm, shown in Figure 6, is used to retrieve the name-specifiers for a particular name-record r from the name-tree T. The main idea behind the algorithm is that a name-specifier can be reconstructed while tracing upwards to the root of the name-tree from parent of the name-record, and grafting on to parts of the name-specifier that have already been reconstructed.

All the value-nodes in the name-tree, T, are augmented with a "PTR" variable, which is a pointer to the corresponding av-pair in the name-specifier being extracted. Initially, all the PTRs are set to *null*, since they have no corresponding av-pairs; the root pointer (T.PTR) is set to point to a new, empty name-specifier. Then, for each parent value of r, the algorithm traces upwards through the name-tree. If it gets to part of the name-tree where there is a corresponding av-pair (v.PTR != null), and it has a name-specifier subtree to graft on to (s != null), it does so. If not, it creates the corresponding part of the name-specifier, sets v.PTR to it, grafts on s if applicable, and continues the trace with the parent value of v and the new subtree. Figure 7 illustrates an in-progress execution of the algorithm.

```
LOOKUP-NAME($T$,$n$)
  $S \leftarrow$ the set of all possible name-records
  for each av-pair $p := (n_a, n_v)$ in $n$
    $T_a \leftarrow$ the child of $T$ such that
              $T_a$'s attribute $= n_a$'s attribute
    if $T_a = null$
      continue

    if $n_v = *$                    ▷ wild card matching
      $S' \leftarrow \emptyset$
      for each $T_v$ which is a child of $T_a$
        $S' \leftarrow S' \cup$ all of the name-records in the
                          subtree rooted at $T_v$
      $S \leftarrow S \cap S'$
    else                            ▷ normal matching
      $T_v \leftarrow$ the child of $T_a$ such that
              $T_v$'s value $= n_v$'s value
      if $T_v$ is a leaf node or $p$ is a leaf node
        $S \leftarrow S \cap$ the name-records of $T_v$
      else
        $S \leftarrow S \cap$ LOOKUP-NAME($T_v$, $p$)
  return $S \cup$ the name-records of $T$
```

Fig. 5. The LOOKUP-NAME algorithm. This algorithm looks up the name-specifier n in the name-tree T and returns all appropriate name-records.

2.4 Resolver network

To propagate updates and forward data to services and clients, the INRs must be organized as a connected network. In our current design, this application-level overlay network is constructed in a distributed way by INRs self-configuring to form a spanning tree based on metrics that reflect INR-to-INR round-trip latency. The experiments conducted by the INRs to obtain this metric are called *INR-pings,* which consist of sending a small name between INRs and measuring the time it takes to process this message and get a response.

A list of active and candidate INRs is maintained by a well-known entity in the system, called the *Domain Space Resolver* (DSR). The DSR may be thought of as an extension to a DNS server for the administrative domain in which we currently are, and may be replicated for fault-tolerance. DSRs support queries to return the currently active and candidate INRs in a domain.

```
GET-NAME($T$,$r$)
  $n \leftarrow$ a new, empty name-specifier
  $T$.PTR $\leftarrow n$
  for each $T_v$ which is a parent value-node of $r$
    TRACE($T_v$, null)
  reset all PTRs that have been set to null
  return $n$

TRACE($T_v$,$n$)
  if $T_v$.PTR $\neq$ null          ▷ something to graft onto
    if $n \neq$ null                ▷ something to graft
      graft $n$ as a child of $T_v$.PTR
  else                              ▷ nothing to graft onto; make it
    $T_v$.PTR $\leftarrow$ a new av-pair consisting of
                   $T_v$'s value and its parent's attribute
    if $n \neq$ null                ▷ something to graft
      graft $n$ as a child of $T_v$.PTR
    TRACE(parent value-node of $T_v$, $T_v$.PTR)
```

Fig. 6. The GET-NAME algorithm. This algorithm extracts and returns the name-specifier for the name-record r in the name-tree T. TRACE implements most of the functionality, tracing up from a leaf-value until it can graft onto the existing name-specifier.

When a new INR comes up, it contacts the DSR to get a list of currently active INRs. The new INR then conducts a set of INR-pings to the currently active INRs and picks the one with the minimum value to establish a neighbor relationship (or *peer*) with. If each INR does this, the resulting topology is a spanning tree. Because the list of active INRs is maintained by the DSR, and all the other INRs obtain the same list, race conditions are avoided and one can impose a linear order amongst the active INRs based on the order in which they became active in the overlay network. Each INR on the active list, except the first one, has at least one neighbor ahead of it in the linear order, and the resulting graph is clearly connected by construction. Furthermore, each time a node arrives after the first one, it peers with exactly one node, so the number of links formed in an $n-$node network is $n-1$. Any connected graph with n nodes and $n-1$ links must be a tree.

Of course, despite each node making a local minimization decision from the INR-pings, the resulting spanning tree will not in general be the minimum one. We are currently working on improving this configuration algorithm by designing a relaxation phase that asynchronously changes neighbor relationships to eventually converge to an optimal

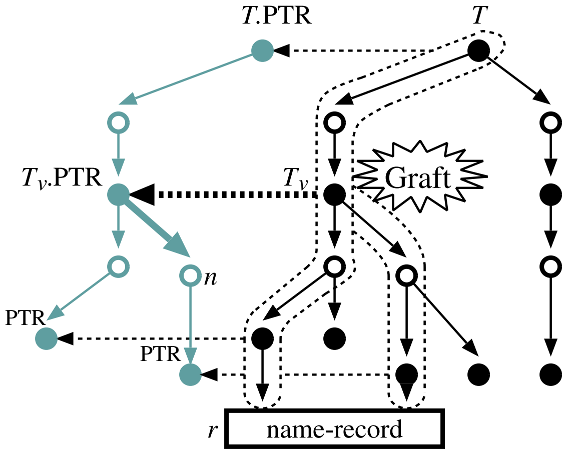


Fig. 7. An illustration of an in-progress execution of the Get-Name algorithm. The name-specifier being created is shown in gray on the left, while the name-tree it is being created from is shown in black on the right. The parts of the name-tree that are circled with dotted lines are the paths through the name-tree that have been traced. The dotted arrows are used to illustrate the assignments of the PTR variables. The thick arrows indicate the parts of the data structures that are currently being manipulated. In this example, the name-specifier fragment rooted at n is being grafted onto T_v.PTR, which is part of the main name-specifier.

tree in the absence of mobility. We also note that a spanning tree may not be a sufficiently robust overlay topology to exchange names and perform intentional multicast, because it has several single points of failure. We are currently exploring other algorithms for constructing more redundant overlay structures.

2.5 Load balancing and scaling

There are two potential performance and scaling bottlenecks in the system—lookups and name update processing. To handle excessive lookup loads, we allow INRs to spawn instances on other candidate (but currently inactive) resolvers, and kill themselves if they are not loaded. To spawn an INR on a candidate node, an INR obtains the candidate-node information from the DSR. An INR can also terminate itself if its load falls below a threshold, informing its peers and the DSR of this. The spanning tree overlay algorithm then adjusts to these changes in the active INR list.

Since INRs exchange name information with other resolvers on a periodic basis and also via triggered updates, update scalability is a serious concern. That is, after a point, the volume of name updates will start to saturate either the available bandwidth or processing capacity of a given resolver node. We conducted several experiments to understand the bottlenecks in our design. While the link bandwidth and processing time required for the name update protocol depends on the size of the name-specifiers and the complexity of the name tree, we found that the process was CPU-bound in all our experiments. On our Java implementation between various Pentium II machines running Linux RedHat 5.2 over 1-5 Mbps wireless links, we found that for a relatively rapid refresh interval of 15 seconds with randomly-generated 82-byte intentional names, the processor was saturated before the bandwidth consumption was 1 Mbps (Figure 8). We also found that the name processing in the name dissemination protocol dominated the lookup processing in most of our experiments. This occurs because in this design, all the resolvers need to be aware of all the names in the system and have to process them.

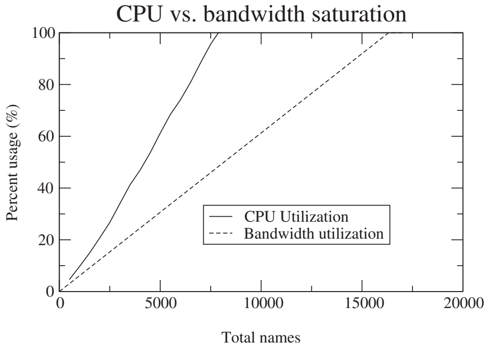


Fig. 8. An example of a CPU-bound configuration of INS. The Pentium II processor is saturated well before a 1mbit/s link. Numbers are shown with refreshes happening every 15 seconds.

Based on these experiments and a better understanding of the scaling bottleneck, we describe a solution that alleviates it. The idea is to partition the namespace into several *virtual spaces*, ensuring that each resolver only needs to route for a subset of all the active virtual spaces in the system. Conceptually, there is now one resolver overlay network per virtual space (however, the overlays for different virtual spaces may span the same resolver nodes).

More formally, we define a virtual space to be an application-specified set of names that share some attributes in common. For instance, all the cameras in building NE-43 at MIT could form the *camera-ne43* virtual space, and all the devices in the building NE43 could form the *building-NE43* virtual space. In the first case, the names (services) in the space might share the "service" (equal to "camera") and "location" (equal to NE-43 in MIT) attributes in common, while in the second case, they all share the same location. INS does not assume particular virtual space names in the system, but does require that each service name the virtual spaces it belongs to (it may belong to multiple virtual spaces too). Clients and applications may interact with services in any virtual space.

An INR knows which virtual space an advertisement or query belongs to because it standardizes a well-known attribute, "vspace" by which applications can express the name of their virtual space. The names of two virtual spaces for different sets of services must not collide, and we are currently exploring ways of handling this issue. Internally, an INR stores the names of different virtual spaces in separate, self-contained name trees.

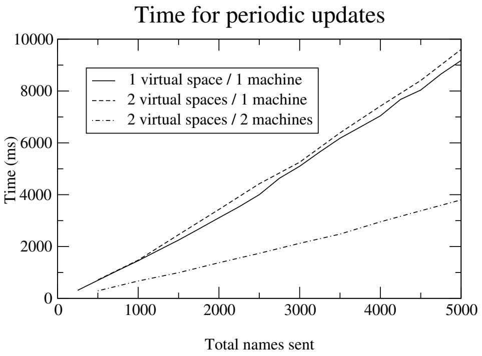


Fig. 9. Periodic update times when the names are divided into two equally-sized spaces.

Partitioning virtual spaces appears to be a promising way to shed load and significantly improve the scalability of INS, especially up to several thousand services. Based on several experiments, we found that the processing time required for periodic updates reduces proportionally when we partition the names into different virtual spaces and then distribute them on to separate resolvers, as shown in Figure 9. If an INR gets a request from a client to resolve for a virtual space it does not route for, it needs to forward the request to a resolver that does. This can be done by caching the resolvers for a small number of popular virtual spaces, and if a cache miss occurs, sending the request to the DSR to be resolved by an appropriate resolver.

In summary, two simple techniques hold promise for scaling the current performance of INS. If an INR is heavily loaded because of name lookups, it can obtain a candidate-INR list and spawn a new INR to handle some of its current load. The configuration protocol used by clients to pick a default INR will cause some of them to move to the newly spawned INR. If an INR is loaded because of update processing, it is likely that all the INRs in that virtual space are also loaded. Therefore, the solution is not to spawn another one for the same space, but to *delegate* one or more virtual spaces to a new INR network. Our experimental results indicate that this is a promising approach to take and we have started implementing this idea.

3 Applications

This section describes the INS API and three of the applications we have developed using it that leverage its support for resource discovery, mobility, and group communication. We describe *Floorplan*, a map-based discovery tool for location-dependent services, *Camera*, a mobile camera network, and *Printer*, a load-balancing printer utility.

An application uses the API to create a name-specifier for a service and to periodically advertise it to the INR network. To discover new services, an application uses the API to send a *discovery* message to an INR to find out what services matching a given name-specifier have been discovered by it. After discovering name-specifiers, the application communicates with the corresponding services by using the API functions to construct a message. Applications choose intentional anycast or intentional multicast by setting the *Delivery* bit-flag in the message header, and early or late binding by setting the *Binding* bit-flag.

3.1 *Floorplan*: a service discovery tool

Floorplan is a service discovery tool that shows how various location-based services can be discovered using the INS. As the user moves into a new region, a map of that region pops up on her display as a building floorplan. *Floorplan* learns about new services by sending a *discovery* message to an INR. This message contains a name-specifier that is used as a filter, and all the name-specifiers that match it are sent back to the application. Floorplan uses the location and service information contained in the returned name-specifiers to deduce the location and the type of each service and display the appropriate icon.

An important component of *Floorplan* is *Locator*, a location server. Rather than directly incorporate maps of regions, *Floorplan* retrieves them as needed from *Locator*. This retrieval is done by sending a request using a name-specifier such as:
`[service=locator[entity=server]][`*location*`].`
In response, *Locator* retrieves the desired map and sends it back to the requesting *Floorplan* instance, using the requestor's intentional name to route the message.

As services are announced or timed out, new icons are displayed or removed. Clicking on an icon invokes the appropriate application for the service the icon represents. The implementation of *Floorplan* deployed in our building allows users to discover a variety of services including networked cameras (Section 3.2), printers (Section 3.3), and device controllers for TV/MP3 players. These service providers advertise name-specifiers specifying several of their attributes, including their location in the building. For example, a camera in Room 510 advertises the following name-specifier:
`[service=camera[entity=transmitter][id=a]][room=510]`

3.2 *Camera*: a mobile camera service

We have implemented a mobile camera service, *Camera*, that uses INS. There are two types of entities in *Camera*: transmitters and receivers. A receiver requests images from the camera the user has chosen (in *Floorplan*) by sending requests to an intentional name that describes it. These requests are forwarded by INRs to a *Camera* transmitter, which sends back a response with the picture.

There are two possible modes of communication between camera transmitters and receivers. The first is a request-response mode, while the second is a subscription-style interaction that uses intentional multicast for group communication. In the request-response mode, a receiver sends an image request to the transmitter of interest by appropriately naming it; the corresponding transmitter, in turn, sends back the requested image to the receiver. To send the image back to only the requester, the transmitter uses the `id` field of the receiver that uniquely identifies it. *Camera* uses this to seamlessly continue in the presence of node or camera mobility.

For example, a user who wants to request an image from a camera in room 510 can send out a request to INRs with destination name-specifier:
`[service=camera[entity=transmitter]][room=510]`
and source name-specifier:
`[service=camera[entity=receiver][id=r]][room=510]`
The transmitter that receives this request will send back the image with the source and destination name-specifiers inverted from the above. The `room` attribute in the destination name-specifier refers to the *transmitter's* location; the `id` attribute allows the INRs to forward the reply to the interested receiver.

When a mobile camera moves to a different network location, it sends out an update to an INR announcing its name from the new location. The name discovery protocol ensures that outdated information is removed from the name-tree, and the new name information that reflects the new network location will soon come into effect. Thus, any changes in network location of a service is rapidly tracked and refreshed by INRs, allowing applications to continue.

In addition to such network mobility, INS also allows applications to handle service mobility. Here, a service such as a mobile camera moves from one location to another, and its network location does not (necessarily) change. However, its intentional name may change to reflect its new location or any new properties of the new environment it has observed, and it may now be in a position to provide the client with the information it seeks. With intentional names, such application-specific properties such as physical location can be expressed and tracked.

Camera uses intentional multicast to allow clients to communicate with groups of cameras, and cameras to communicate with groups of users. It takes advantage of the property that an intentional name can be used not only for rich service descriptions, but also to refer to a group of network nodes that share certain properties that are specified in their names.

To use this feature, the *Camera* transmitter sends out an image destined to all users subscribing to its images by setting the D bit-flag to *all*. When an INR looks up a name-specifier, it finds all of the network locations that match it. Rather than forwarding the data to just the best one of them, it sends the data to each next-hop INR for which there is an associated network location. Similarly, a user can also subscribe to *all* cameras in the building (or a subset of them named by certain criteria).

For example, a camera transmitter located in room 510 sends out its images to all of its subscribers at once using the following destination name-specifier:

```
[service=camera [entity=receiver][id=*]][room=510]
```

and set the *Delivery* bit-flag to *all*. The use of wild card `[id=*]` refers to all subscribers, regardless of their specific IDs.

When implementing *Camera*, we noticed that it would be useful to cache pictures at various places in the network, so that requests do not have to go back to the original server every time. To achieve this, we designed an *application-independent* extension to INS that allows INRs to cache data packets. Intentional names made the design of application-independent caching rather simple. With traditional naming schemes each application provides its own opaque names for its data units, and today's distributed caching schemes are tied to specific applications (e.g., Web caches). In contrast, intentional names give applications a rich vocabulary with which to name their data, while keeping the structure of these names understandable without any application-specific knowledge. Thus, the intentional names can be used as a handle for a cached object. Of course, it is still necessary to provide additional information to describe if or for how long the object should be cached; we therefore added a small number of additional fields to the INS message header to convey this information to the INRs.

3.3 *Printer*: a load-balancing printer utility

The printer client application starts when the user clicks on a printer icon on the floorplan display. The printer client application has several features. It can retrieve a list of jobs that are in the queue of the printer, remove a selected job from the queue provided the user has permission to do so, and allow the user to submit files to the printer. Job submissions to *Printer* can be done in two ways, one of which uses intentional anycast to discover the "best" printer according to location and load characteristics.

The first submission mode is the straightforward "submit job to *name*," where the *name* is the printer's intentional name. The second mode, which is one we find useful in day-to-day use, is to submit a job based on the user's location. The printer servers, which are proxies for the actual printers in our implementation, change the metrics that are periodically advertised to the INRs taking into account the error status, number of enqueued jobs, the length of each one, etc. The INRs forward jobs to the currently least-loaded printer based on these advertisements, and inform the user of the chosen printer. Advertising a smaller metric for a less loaded printer and using intentional anycast allows *Printer* to perform automatically balance their load.

For example, to submit a file to the least-loaded printer in room 517, the printer client sends the file with the following destination name-specifier:

`[service=printer [entity=spooler]][room=517]`

and sets the *Delivery* bit-flag to *any.* Note that the name of the printer is omitted on purpose. Using intentional anycast, INRs automatically pick the route that has the best metric for the specified printer name-specifier, which corresponds to the least-loaded printer in room 517.

4 Implementation

We have implemented INS and tested it using a number of applications, including those described in the previous section. Our INR implementation is in Java, to take advantage of its cross-platform portability; clients and services, however, are not constrained to be written in Java. In this section, we present the details of two aspects of INS: the architecture of an INR node, and the packet formats for intentional names.

INRs use UDP to communicate with each other. At an INR, the `Node` object manages all network resources. It maintains the `NameTree` that is used to resolve an intentional name to its corresponding name-record, a `NodeListener` that receives all incoming messages, and a `ForwardingAgent` to forward messages to INRs and applications. In addition, a `NameDiscovery` module implements the name discovery protocol, and a `NetworkManagement` application provides a graphical interface to monitor and debug the system, and view the name-tree. At the client, a `MobilityManager` detects network movement and rebinds the UDP socket if the IP address changes, transparent to applications.

The INR implementation consists of approximately 8500 lines of Java code, of which about 2500 lines are for the INS API. The API significantly eases application development—for instance, the *Floorplan* and *Camera* applications presented in Section 3 were each implemented in less than 400 lines of Java code (including both service and client code, but excluding the graphical user-interface), and the *Printer* application in less than 1000 lines.

Figure 10 shows the INS packet format for intentional names. The binding bit-flag (B) is used to determine whether early or late binding should be used, while the delivery bit-flag (D) is used to determine whether intentional anycast or multicast delivery should be used. Because name-specifiers are of variable length, the header contains pointers to the source name-specifier, destination name-specifier, and data, which give offsets from the beginning of the packet. This allows the forwarding agent of an INR to find the end of name-specifiers without having to parse them. INRs do not process application data. In addition, a *hop limit* field decrements at each hop and limits the number of hops a message can traverse in the overlay. The *cache lifetime* field gives the lifetime the data of this packet may be cached for, with a value of zero disallowing caching.

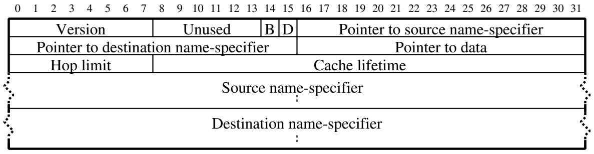


Fig. 10. The INS message header format.

5 Performance analysis and evaluation

In this section, we analyze the performance of the INS name lookup algorithm and present the results of our experiments with the lookup algorithm and name discovery protocol. These experiments were all conducted using off-the-shelf Intel Pentium II 450 MHz computers with a 512 kb cache and 128 Mb RAM, running either Red Hat Linux 5.2 or Windows NT Server 4.0, with our software built using Sun's Java version 1.1.7. The network nodes were connected over wireless RF links ranging between 1 and 5 Mbps.

5.1 Name lookup performance

Analysis To understand how INS scales with increasing lookup load, it is important to analyze the performance of the lookup algorithm. We analyze the worst-case run-time of the algorithm as a function of the complexity of the incoming name-specifier and the name-tree. To simplify the analysis, we assume that name-specifiers grow uniformly in the following dimensions (illustrated in Figure 11):

d	One-half the depth of name-specifiers
r_a	Range of possible attributes in name-specifiers
r_v	Range of possible values in name-specifiers
n_a	Actual number of attributes in name-specifiers

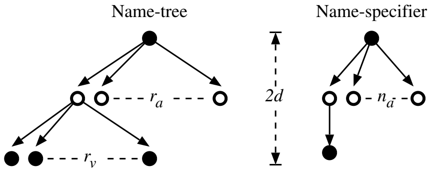


Fig. 11. A uniformly grown name-tree. Note that $d = (\text{tree depth})/2 = 1$ for this tree.

In each invocation, the algorithm iterates through the attributes in the name-specifier, finding the corresponding attribute and value in the name-tree and making a recursive call. Thus, the run-time is given by the recurrence:

$$T(d) = n_a \cdot (t_a + t_v + T(d-1)),$$

where t_a and t_v represent the time to find the attribute and value respectively. For now, assume that it takes time b for the base case:

$$T(0) = b$$

Setting $t = t_a + t_v$ and performing the algebra yields:

$$\begin{aligned} T(d) &= n_a \cdot (t + T(d-1)) \\ &= \frac{n_a^d - 1}{n_a - 1} \cdot t + n_a^{d-1} \cdot b \\ &= \Theta(n_a^d \cdot (t + b)) \end{aligned}$$

If linear search is used to find attributes and values, the running time would be:

$$T(d) = \Theta(n_a^d \cdot (r_a + r_v + b)),$$

because $t_a \propto r_a$ and $t_v \propto r_v$ in this case.

However, using a straightforward hash table to find these reduces the running time to:

$$T(d) = \Theta(n_a^d \cdot (1 + b))$$

From the above analysis, it seems that the n_a^d factor may suffer from scaling problems if d grows large. However, both n_a and d, scale up with *the complexity of a single application* associated with the name-specifier. There are only as many attributes or levels to a name-specifier as the application designer needs to describe the objects that are used by their application. Consequently, we expect that d will be near-constant and relatively small; indeed, all our current applications have this property in their name-specifiers.

The cost of the base case, b, is the cost of an intersection operation between the set of route entries at the leaf of the name-tree and the current target route set. Taking the intersection of the two sets of size s_1 and s_2 takes $\Theta(max(s_1, s_2))$ time, if the two sets are sorted (as in our implementation). In the worst case the value of b is of the order of the size of the universal set of route entries ($\Theta(|U|)$), but is usually significantly smaller. Unfortunately, an average case analysis of b is difficult to perform analytically since it depends on the number and distribution of names.

Experiments To experimentally determine the name lookup performance of our (untuned) Java implementation of an INR, we constructed a large, random name-tree, and timed how long it took to perform 1000 random lookup operations on the tree. The name-tree and name-specifiers were uniformly chosen with the same parameters as in the analysis in Section 5.1. We varied n, the number of distinct names in the tree, and measured lookup times. We limited the maximum heap size of the Java interpreter to 64 Mb and set the initial allocation to that amount to avoid artifacts from other memory allocation on the machine. The range of our experiments was limited by the memory required to store the distinct names to be looked up (part of the experimental apparatus) rather than the name-tree itself (which is much more compact).

We fixed the parameters at $r_a = 3$, $r_v = 3$, $n_a = 2$, and $d = 3$, and varied n from 100 to 14300 in increments of 100. Our results are shown in Figure 12. For this name-tree and name-specifier structure, our performance went from a maximum of about 900 lookups per second to a minimum of about 700 lookups per second. This experiment gave us a practical idea of how the base case b affects performance.

For the same experiment, we also recorded the amount of memory allocated by Java to the experiment; this amount should be greater than the actual name-tree size by only a constant amount. The strings we used for attribute and value names were only one (Unicode) character or 16 bits long, thus the memory is representative of what a more compact encoding of attributes and values would achieve. However the growth of the name-tree would remain the same, since after the first thousand names are in the name tree (where the graph curves up from zero) all of the attributes and values that exist are stored in the name-tree, and additional memory usage comes only from additional pointers and name-records. Our results are shown in Figure 13. The amount of memory allocated to the name-tree went from approximately 0.5 megabytes to 4 megabytes as the number of names was increased. We believe that this order-of-magnitude of lookup performance is adequate for intra-domain deployment, because of the load balancing provided by having multiple INRs and the parallelism inherent in independent name lookups.

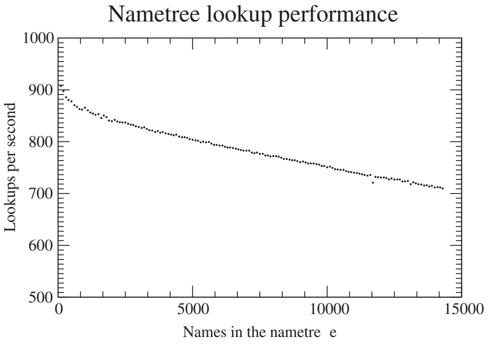


Fig. 12. Name-tree lookup performance. This graph shows how the name-tree lookup performance of an INR varies according to the number of names in its name-tree.

5.2 Name discovery performance

This section shows that INS is responsive to change and dynamism in services and nodes, by discussing the performance of the name discov-

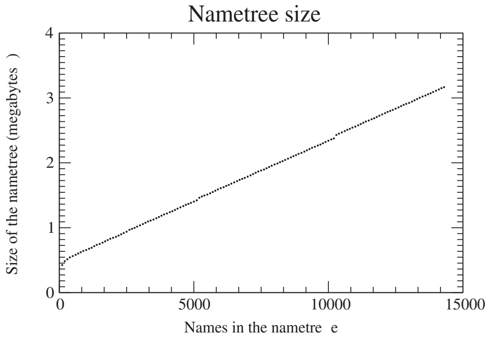


Fig. 13. Name-tree size. This graph shows how the name-tree size varies according to the number of names in its name-tree.

ery protocol. We measured the performance of INS in discovering *new* service providers, which advertise their existence via name-specifiers. Figure 14 shows the average discovery time of a new name-specifier as a function of n, the number of hops in the INR network from the new name.

When an INR observes a new name-specifier from a service advertisement, it processes the update message and performs a lookup operation on the name-tree to see if a name-specifier with the same AnnouncerID already exists. If it does not find it, it grafts the name-specifier on to its name-tree and propagates a triggered update to its neighbors. Thus, the name discovery time in a network of identical INRs and links, $T_d(n) = n(T_l + T_g + T_{up} + d)$, where T_l is the lookup time, T_g is the graft time, T_{up} is the update processing time, and d is the one-way network delay between any two nodes. That is, name discovery time should be linear in the number of hops. The experimental question is what the slope of the line is, because that determines how responsive INS is in tracking changes.

In our experiments the structure of the name-tree on each INR was relatively constant except for the new grafts, since we were not running any other applications in the system during the measurements. Thus, the lookup and graft times at one INR and the others were roughly the same. As shown in Figure 14, $T_d(n)$ is indeed linear in n, with a slope of less than 10 ms/hop. This implies that typical discovery times are only a few tens of milliseconds, and dominated by network transmission delays.

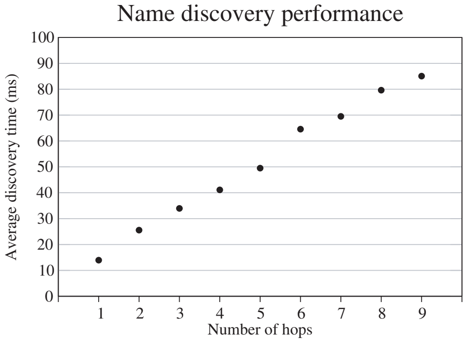


Fig. 14. Discovery time of a new network name. This graph shows that the time to discover a new network name is linear in the number of INR hops.

5.3 Routing performance

In addition to the lookup and discovery experiments, we also measured the performance of the overall system when both occur simultaneously. For these experiments, we sent a burst of one hundred 586-byte messages, gathered from the *Camera* application, between 15-second periodic update intervals. The name specifier source and destination addresses were randomly generated, on average 82 bytes long. The results are shown in Figure 15.

For the case in which the sender and receiver are on the same node, the processing and routing time varies somewhat with the name-tree size for the given virtual space, from 3.1 ms per packet with 250 names to 19 ms per packet with 5000 names. This is partially due to the speed of the name-tree lookups, but is also an artifact of the current end-application delivery code, which happens to vary linearly with the number of names. We observe a flatter line by when examining the data for packets destined to a remote INR in the name-tree of the same virtual space. For the most part, the next-hop processing time is about 9.8 ms per packet during the burst. In this case, name-tree lookups still occur, but the end-application delivery code is not invoked. This gives a better indication of the pure lookup and forwarding performance.

When the recipient resides in a different virtual space on another node, we observe a nearly constant time of 381 ms to resolve and route the burst of 100 messages. This steady time comes from the node having no knowledge of the end virtual space except for a next-hop INR address, which is requested and cached from the DSR on the first access, to which it can forward packets.

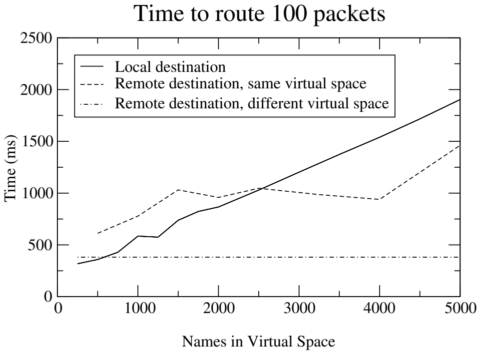


Fig. 15. Processing and routing time per INR for a 100-packet burst, in the intra-INR, inter-INR, and inter-virtual-space cases.

6 Related work

A flexible naming and resolution system for resource discovery, such as that provided by INS, is well-suited to dynamic network environments. INS uses a simple, expressive name language, late binding machinery that integrates resolution and routing for intentional anycast and multicast, soft-state name dissemination protocols for robustness, and a self-configuring resolver network.

INS is intended to *complement,* not replace the Internet DNS, which maps hostnames to IP addresses. DNS names are strictly hierarchical, whereas INS names use a more expressive attribute-based language. Unlike DNS, name propagation in INS resembles a routing protocol, tuned to perform rapid updates. In INS, names originate from and are refreshed by applications that advertise them. This enables *fate sharing* [9] between names and the corresponding services—if a node providing a service crashes, it will also cease to announce that service. In DNS, resolvers form a static overlay network consisting of the client's nameserver, the root server, and the owner domain's nameserver to route and resolve requests, unlike the INS self-configuring overlay.

There has been some recent activity in service discovery for heterogeneous networks of devices. Sun's Jini [21] provides a framework for spontaneous distributed computing by forming a "federation of networked devices" over Java's Remote Message Invocation (RMI). Jini does not address how resource discovery will work in a dynamic environment or when services fail, and can benefit from INS as its resource discovery system. Universal plug-and-play [42] uses a subset of XML to describe resources provided by devices and, like Jini, can benefit from INS as a discovery system. The Service Location Protocol (SLP) [44, 33] facilitates the discovery and use of heterogeneous network resources using centralized Directory Agents. The Berkeley Service Discovery Service (SDS) [11] extends this concept with secure, authenticated communications and a fixed hierarchical structure for wide-area operation. Unlike Jini, SLP, and SDS, INS handles dynamism via late binding, provides intentional anycast and multicast services, has self-configuring resolvers, and does not rely on IP multicast to perform discovery.

Numerous attribute-based directory services have been proposed in the past. The X.500 distributed directory [7, 36] by the CCITT (now the ITU-T) facilitates the discovery of resources by using a single global namespace with decentralized maintenance. INS differs from X.500 in its goals and mechanisms to achieve responsiveness and expressiveness; INS enables late binding and uses soft-state name dissemination. The INS resolver network is also different from the static X.500 hierarchy. These differences arise from differences in our environment, which is a dynamic and mobile network with little pre-configured infrastructure.

In addition to the wealth of classical literature on naming in distributed systems (e.g., Grapevine [4], Global Name Service [23], etc.), there has been some recent research in wide-area naming and resolution. For example, Vahdat *et al.* [43] present a scheme for *ActiveNames,* which allow applications to define arbitrary computation that executes on names at resolvers. INS and ActiveNames share some goals in common, but differ greatly in how they achieve them. In particular, INS does not require mobile code, relying instead on a simple but expressive naming scheme to enable applications to express intent, and late binding to be responsive to change. In addition, INS implements a self-configuring resolver network based on network performance.

An early proposal to decouple names from object locations was described in a paper by O'Toole and Gifford [28], where they describe a content naming scheme and its application to Semantic File Systems [18]. Their design and application of content names is very different from ours, but the underlying philosophy is similar. The *Discover* system [39] is an HTTP-based document discovery system that uses *query routing* to forward a query to the servers that contain the result. Discover is document-centric and uses parallel processes to search servers and merge the results.

Oki *et al.* introduce the *Information Bus* [30] to allow applications to communicate by describing the subject of the desired data, without knowing who the providers are. Other projects with a similar flavor include Malan *et al.'s* Salamander [25] and Talarian's SmartSockets [40]. These use a flat naming scheme, do not support late binding, and have statically configured resolvers. The idea of separating names from network locations was also proposed by Jacobson in the context of multicast-based self-configuring Web caches [20]. Estrin *et al.* build on this, exploring a diffusion-based approach to data dissemination in sensor networks using data attributes to instantiate forwarding state at sensor nodes [15]. Our intentional naming scheme has some features in common with these proposals, but differs in the details of the resolution, late binding and name dissemination processes, as well as the overall resolver architecture.

Cisco's DistributedDirector [8] resolves a URL to the IP address of the "closest" server, based on client proximity and client-to-server link latency. Unlike INS, DistributedDirector is not a general framework for naming and resolution and does not integrate resolution and routing. Furthermore, each resolver is independent in DistributedDirector, whereas they form a cooperating overlay network in INS.

IBM's "T Spaces" [24] enable communication between applications in a network by providing a lightweight database, over which network nodes can perform queries. However, this system has been optimized for relatively static client-server applications rather than for dynamic peer-to-peer communication, and uses a central database to maintain tuple mappings. Other architectures for object-oriented distributed computing are OMG's CORBA [29] and the ANSA Trading Service [13], where federated servers resolve client resolution requests.

Retaining network connectivity during mobility requires a level of indirection so that traffic to the mobile host can be redirected to its current location. Mobile IP [32] uses a Home Agent in the mobile host's home domain for this. With INS, the level of indirection to locate mobile services and users is obtained using the intentional naming system, since all traffic to the mobile service would go through the name resolution process. The tight integration of naming and forwarding enables continued network connectivity in the face of service mobility, and the decentralized INS architecture and name discovery protocols enhance robustness. A number of protocols for ad hoc or infrastructure-free routing have recently been proposed (e.g., [6]). These protocols, are essential to enable IP connectivity, but do not provide resource discovery functionality.

7 Conclusions

In this paper, we established the need for an intentional naming scheme, where applications describe *what* they are looking for, not *where* to find data. Our design goals were expressiveness, responsiveness, robustness and easy configuration. We presented the design, implementation and evaluation of an Intentional Naming System (INS) that meets these goals. INS uses a simple naming language based on attributes and values to achieve expressiveness, integrates name resolution and message routing to allow applications to be responsive to mobility and performance changes, uses periodic service advertisements and soft-state name dissemination protocols between replicated resolvers to achieve robustness, and deploys self-configuring name resolvers to ease configuration. The INS service model supports three types of resolution: early binding, where an application can obtain a list of IP addresses corresponding to a name, and two forms of late binding: intentional anycast and intentional multicast. Intentional anycast forwards a message to the "best" node satisfying a query while optimizing an application-controlled metric, and intentional multicast forwards a message to all names satisfying a query.

We presented the design and analysis of an efficient algorithm for name lookups and measurements of our implementation, which show that a Java implementation can perform between several hundred lookups per second (for complex name-specifiers) to a few thousand lookups per second. We evaluated the name discovery protocol and demonstrated that INS could disseminate information about new names in tens of milliseconds. We also measured the the processing time for name updates, analyzed the scaling bottlenecks, and found that namespace partitioning is a practical technique to improve the scalability of INS.

Our experience with INS has convinced us that using intentional names with late binding is a useful way of discovering resources in dynamic, mobile networks, and simplifies the implementation of applications. We emphasize that INS allows applications to efficiently track dynamic data attributes, because the choice of attributes to use in name-specifiers is completely under application-control. We therefore believe that INS has the potential to become an integral part of future device and sensor networks where decentralized, easily configurable resource discovery is essential.

There remain some important areas of research before the full benefits of INS can be realized. First, we need to carefully expand the set of supported operators in the resolution process, incorporating range matches in addition to exact matches of attributes and values. Second, the current INS architecture is intended for intra-domain deployment. We are actively developing a wide-area architecture to scale INS to wide-area networks. Third, the name discovery protocols need to be tuned to use bandwidth efficiently while disseminating names; some names are more ephemeral or more important than others, implying that all names must not be treated equally by the soft-state dissemination protocol [35]. And perhaps most importantly, we need to incorporate security mechanisms in the naming architecture before a more wide-scale deployment. Ultimately, the benefits of INS are in facilitating the development of useful applications and services, and we are implementing more applications to demonstrate the benefits of INS and to characterize the class of applications that INS facilitates.

8 Acknowledgments

This work was supported by a research grant from the NTT Corporation. We would like to thank Minoru Katayama and Ichizo Kogiku for their interest in this effort. We are grateful to John Guttag, Frans Kaashoek, and Suchitra Raman for useful suggestions on the design of INS, and to Dave Andersen, Frans Kaashoek, Barbara Liskov, Robert Morris, and Suchitra Raman for commenting on earlier drafts of this paper. This paper also benefited from the many comments of our “shepherd,” Jean Bacon, as well as the SOSP reviewers.

References

1. R. Bellman. On a routing problem. *Quarterly of Applied Mathematics, 16(1):87-90*, 1958.
2. T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol–HTTP/1.0.* Internet Engineering Task Force, May 1996. RFC 1945 (`http://www.ietf.org/rfc/rfc1945.txt`).
3. S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei. Application-Layer Anycasting. In *Proc. IEEE INFOCOM*, pages 1388–1396, March 1997.

4. A. Birrell, R. Levin, R. Needham, and M. Schroeder. Grapevine: An exercise in distributed computing. *Comm. of the ACM*, 25(4):260–274, April 1982.
5. T. Bray, D. Hollander, and A. Layman. Namespaces in XML. `http://www.w3.org/TR/1998/WD-xml-names-19980327`, March 1998. World Wide Web Consortium Working Draft.
6. J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proc. ACM/IEEE MOBICOM*, pages 85–97, October 1998.
7. CCITT. *The Directory—Overview of Concepts, Models and Services*, December 1988. X.500 series recommendations, Geneva, Switzerland.
8. Cisco—Web Scaling Products & Technologies: DistributedDirector. `http://www.cisco.com/warp/public/751/distdir/`, 1998.
9. D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. ACM SIGCOMM*, pages 106–114, August 1988.
10. D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms. In *Proc. ACM SIGCOMM*, pages 14–26, August 1992.
11. S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Proc. ACM/IEEE MOBICOM*, pages 24–35, August 1999.
12. S. Deering and D. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2):136–161, May 1990.
13. J. P. Deschrevel and A. Watson. A brief overview of the ANSA Trading Service. `http://www.omg.org/docs/1992/92-02-12.txt`, February 1992. APM/RC.324.00.
14. H. Eriksson. Mbone: The multicast backbone. *Communications of the ACM*, 37(8):54–60, 1994.
15. D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proc. ACM/IEEE MOBICOM*, pages 263–270, August 1999.
16. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, Jan 1997. RFC 2068 (`http://www.ietf.org/rfc/rfc2068.txt`).
17. B. Fink. 6bone Home Page. `http://www.6bone.net/`, January 1999.
18. D. Gifford, P. Jouvelot, M. Sheldon, and J. O'Toole. Semantic File Systems. In *13th ACM Symp. on Operating Systems Principles*, pages 16–25, October 1991.
19. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. Simple Service Discovery Protocol/1.0. `http://search.ietf.org/internet-drafts/draft-cai-ssdp-v1-02.txt`, June 1999. Internet Draft, expires December 1999.
20. V. Jacobson. How to Kill the Internet. Talk at the SIGCOMM 95 Middleware Workshop, available from `http://www-nrg.ee.lbl.gov/nrg-talks.html`, August 1995.

21. Jini (TM). `http://java.sun.com/products/jini/`, 1998.
22. B. Kantor and P. Lapsley. *Network News Transfer Protocol.* Internet Engineering Task Force, February 1986. RFC 977 (`http://www.ietf.org/rfc/rfc977.txt`).
23. B Lampson. Designing a Global Name Service. In *Proc. 5th ACM Principles of Dist. Comput.*, pages 1–10, August 1986.
24. T. Lehman, S. McLaughry, and P. Wyckoff. T Spaces: The Next Wave. `http://www.almaden.ibm.com/cs/TSpaces/`, 1998.
25. G. R. Malan, F. Jahanian, and S. Subramanian. Salamander: A Push-based Distribution Substrate for Internet Applications. In *Proc. USENIX Symposium on Internet Technologies and Systems*, pages 171–181, December 1997.
26. G. Malkin. *RIP Version 2.* Internet Engineering Task Force, November 1998. RFC 2453 (`http://www.ietf.org/rfc/rfc2453.txt`).
27. P. V. Mockapetris and K. Dunlap. Development of the Domain Name System. In *Proceedings of SIGCOMM '88 (Stanford, CA)*, pages 123–133, August 1988.
28. J. O' Toole and D. Gifford. Names should mean what, not where. In *5th ACM European Workshop on Distributed Systems*, September 1992. Paper No. 20.
29. Object Management Group CORBA/IIOP 2.3. `http://www.omg.org/corba/corbaiiop.html`, December 1998.
30. B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus (R) – An Architecture for Extensible Distributed Systems. In *Proc. ACM SOSP*, pages 58–78, 1993.
31. C. Partridge, T. Mendez, and W. Milliken. *Host Anycasting Service*, November 1993. RFC 1546 (`http://www.ietf.org/rfc/rfc1546.txt`).
32. C. Perkins. *IP Mobility Support*, October 1996. RFC 2002 (`http://www.ietf.org/rfc/rfc2002.txt`).
33. C. Perkins. Service Location Protocol White Paper. `http://playground.sun.com/srvloc/slp_white_paper.html`, May 1997.
34. J. B. Postel. *Transmission Control Protocol.* Internet Engineering Task Force, September 1981. RFC 793 (`http://www.ietf.org/rfc/rfc0793.txt`).
35. S. Raman and S. McCanne. A Model, Analysis, and Protocol Framework for Soft State-based Communication. In *Proc. ACM SIGCOMM*, pages 15–25, September 1999.
36. J. Reynolds. *Technical Overview of Directory Services Using the X.500 Protocol*, March 1992. RFC 1309 (`http://www.ietf.org/rfc/rfc1309.txt`).
37. J. Saltzer, D. Reed, and D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2:277–288, Nov 1984.
38. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications.* Internet Engineering Task Force, Jan 1996. RFC 1889 (`http://www.ietf.org/rfc/rfc1889.txt`).

39. M. Sheldon, A. Duda, R. Weiss, and D. Gifford. Discover: A Resource Discovery System based on Content Routing. In *Proc. 3rd Intl. World Wide Web Conf.*, 1995.
40. Rapid Infrastructure Development for Real-Time, Event-Driven Applications. `http://www.talarian.com/collateral/SmartSocketsWP-1.html`, 1998.
41. D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.
42. Universal Plug and Play: Background. `http://www.upnp.com/resources/UPnPbkgnd.htm`, 1999.
43. A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active Names: Flexible Location and Transport of Wide-Area Resources. In *Proc. USENIX Symp. on Internet Technologies & Systems*, October 1999.
44. J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. *Service Location Protocol*, June 1997. RFC 2165 (`http://www.ietf.org/rfc/rfc2165.txt`).
45. M. Wahl, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (Version 3).* Internet Engineering Task Force, December 1997. RFC 2251 (`http://www.ietf.org/rfc/rfc2251.txt`).
46. D. Wetherall. Active network vision and reality: lessons from a capsule-based system. In *Proc. 17th SOSP*, pages 64–79, December 1999.

Proc. of the 6th Annual International
Conference on Mobile Computing and
Networking (MOBICOM 2000),
Boston, MA, August 2000.

An End-to-End Approach to Host Mobility

Alex C. Snoeren and Hari Balakrishnan[⋆]

MIT Laboratory for Computer Science
{`snoeren, hari`}`@lcs.mit.edu`

Abstract. We present the design and implementation of an end-to-end architecture for Internet host mobility using dynamic updates to the Domain Name System (DNS) to track host location. Existing TCP connections are retained using secure and efficient connection migration, enabling established connections to seamlessly negotiate a change in endpoint IP addresses without the need for a third party. Our architecture is secure—name updates are effected via the secure DNS update protocol, while TCP connection migration uses a novel set of *Migrate* options—and provides a pure end-system alternative to routing-based approaches such as Mobile IP.

Mobile IP was designed under the principle that fixed Internet hosts and applications were to remain unmodified and only the underlying IP substrate should change. Our architecture requires no changes to the unicast IP substrate, instead modifying transport protocols and applications at the end hosts. We argue that this is not a hindrance to deployment; rather, in a significant number of cases, it allows for an easier deployment path than Mobile IP, while simultaneously giving better performance. We compare and contrast the strengths of end-to-end and network-layer mobility schemes, and argue that end-to-end schemes are better suited to many common mobile applications. Our performance experiments show that handoff times are governed by TCP migrate latencies, and are on the order of a round-trip time of the communicating peers.

[⋆] This research was supported in part by DARPA (Grant No. MDA972-99-1-0014), NTT Corporation, and IBM. Alex C. Snoeren is supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

1 Introduction

The proliferation of mobile computing devices and wireless networking products over the past decade has made host and service mobility on the Internet an important problem. Delivering data to a mobile host across a network address change without disrupting existing connections can be tackled by introducing a level of indirection in the routing system. This is the approach taken by Mobile IP [28, 30], which deploys a home agent that intercepts packets destined for a host currently away from its home network, and delivers it to the mobile host via a foreign agent in the foreign network. This approach does not require any changes to the fixed (correspondent) hosts in the Internet, but does require changing the underlying IP substrate to effect this *triangle* routing, and an authentication protocol to ensure that connections are not hijacked by a malicious party. Mobile IP is a "pure" routing solution, a network-layer scheme that requires no changes to any higher layer of the Internet protocol stack.

There are many classes of mobile applications [17]: those where other hosts originate connections to a mobile host and can benefit from both host location and handoff support (e.g., a mobile Web server, mobile telephony); those where the mobile host originates all connections, which do not require host location services but can benefit from handoff support (e.g., mail readers, Web browsers); and those where an application-level retry suffices if the network address changes unexpectedly during a short transaction, which need neither to work well (e.g., DNS resolution). We believe that a good end-to-end architecture for host mobility will support all these modes, and empower applications to make the choice best suited to their needs. Our architecture is motivated by, and meets, this goal. It is an end-to-end approach; no changes to the IP substrate are required.

In our mobility architecture, the decision of whether to support transparent connectivity across network address changes (especially useful for mobile servers) or not (not needed for short client-server transactions) is left to the application. While Mobile IP-style, fully-transparent mobility support is general and sufficient for mobile applications, this generality comes at significant cost, complexity, and performance degradation.

To locate mobile hosts as they change their network attachment point, we take advantage of the widely-deployed Domain Name System (DNS) [21] and its ability to support secure dynamic updates [9, 36]. Because most Internet applications resolve hostnames to an IP address at the beginning of a transaction or connection, this approach is viable for initiating new sessions with mobile hosts. When a host changes its network attachment point (IP address), it sends a secure DNS update to one of the name servers in its home domain updating its current location. The name-to-address mappings for these hosts are uncacheable by other domains, so stale bindings are eliminated.

The ability to support continuous communication during periods of mobility without modifying the IP substrate is a more challenging problem. Because TCP is a connection-oriented reliable protocol, many TCP applications reasonably expect this service model in the face of losses and transient link failures, route changes, or mobility. The two communicating peers must securely negotiate a change in the underlying network-layer IP address and then seamlessly continue communication. Furthermore, an approach that *requires* either communicating peer to learn about the new network-layer address before a move occurs is untenable because network-layer moves may be quite sudden and unpredictable.

We design a new end-to-end TCP option to support the secure migration of an established TCP connection across an IP address change. Using this option, a TCP peer can suspend an open connection and reactivate it from another IP address, transparent to an application that expects uninterrupted reliable communication with the peer. In this protocol, security is achieved through the use of a connection identifier, or *token*, which may be secured by a shared secret key negotiated through an Elliptic Curve Diffie-Hellman (ECDH) key exchange [37] during initial connection establishment. It requires no third party to authenticate migration requests, thereby allowing the end points to use whatever authentication mechanism they choose to establish a trust relationship. Although we only describe details for TCP migration, we find that this idea is general and can be implemented in a like manner for specific UDP-based protocols such as the Real-time Transport Protocol (RTP) to achieve seamless mobility for those protocols as well.

One way of thinking of our work is in the context of the end-to-end argument [33], which observes that functionality is often best implemented in a higher layer at an end system, where it can be done according to the application's specific requirements. We show that it is possible to implement mobility as an end-to-end service without network-layer support, while providing multiple mobility modes. In this sense, this is akin to applications deciding between UDP and TCP as a transport protocol; many opt for UDP's simplicity and timeliness over TCP's reliability. In the same fashion, applications should be able to select the mobility mode of their choice.

The other significant advantage of handling mobility on an end-to-end basis is that it enables higher layers like TCP and HTTP to learn about mobility and adapt to it. For example, it is a good idea after a network route change to restart TCP transmissions from slow start or a window-halving [14] since the bottleneck might have changed, or adapt the transmitted content to reflect new network conditions. These optimizations can be made naturally if mobility is handled end-to-end, since no extra signalling is needed. Indeed, the large body of work in mobile-aware applications [16, 23, 26] can benefit from our architecture.

Experience with previous end-to-end enhancements such as various TCP options (e.g., SACK [20]), path MTU discovery, HTTP/1.1, etc., has shown that such techniques often meet with less resistance to widespread deployment than changes to the IP substrate. This supports our belief that, in addition to the flexibility it offers, an end-to-end approach may be successfully deployed.

We have implemented this mobility architecture in Linux 2.2 and have conducted several experiments with it. We are encouraged by the ease with which seamless mobility can be achieved, the flexibility it provides, and the lack of performance degradation. Since our scheme does not impose any triangle routing anomalies, end-to-end latency for active connections is better than standard Mobile IP, and similar to Mobile IP with route optimization.

The rest of this paper describes the technical details of our approach. In Section 2, we survey related work in the area of mobility support. We describe our architecture in Section 3, and detail our new Migrate TCP option in Section 4. We discuss the security ramifications of our approach in Section 5 and our implementation and performance results in Section 6. We address some deployment issues in Section 7 and conclude in Section 8.

2 Related work

The problem of Internet host mobility has been approached from many angles in the literature, but they can be classified into two categories. Some techniques attempt to handle host relocation in a completely transparent fashion, hiding any changes in network structure from the end hosts. We term these techniques *network-layer* mobility. By contrast, many other approaches attempt to handle relocation at a higher level in the end host.

2.1 Network-layer mobility

Mobile IP (RFC 2002) [30] is the current IETF standard for supporting mobility on the Internet. It provides transparent support for host mobility by inserting a level of indirection into the routing architecture. By elevating the mobile host's *home address* from its function as an interface identifier to an *end-point identifier* (EID), Mobile IP ensures the delivery of packets destined to a mobile host's home address, independent of the host's physical point of attachment to the Internet, as reflected in its *care-of address*. Mobile IP does this by creating a routing tunnel between a mobile host's home network and its care-of address.

Such routing tunnels need to be implemented with care because advertising explicit host routes into the wide-area routing tables destroys routing scalability. Mobile IP uses a *home agent* physically attached to the mobile host's home network to intercept and tunnel packets to the mobile host. Hence, packets undergo *triangle* routing, which is often longer than the optimal unicast path.

Further compounding the problem is the widespread deployment of ingress filters [10], ratified in February 2000 by the IETF as a "Best Current Practice" to combat denial-of-service attacks. With this mechanism, a router does not forward packets with a source address foreign to the local network, which implies that a packet sent by a mobile host in a foreign network with its source address set to its home address will not be forwarded. The solution to this is to use *reverse tunneling*, which tunnels packets originating at the mobile host first to the host's home agent (using the host's care-of address as a source address), and then from there on to the destination using the home address as the source address. Thus, routing anomalies occur in both directions.

Perkins and Johnson present a route optimization option for Mobile IP to avoid triangle routing [29]. Here, correspondent hosts cache the care-of address of mobile hosts, allowing communication to proceed directly. It requires an authenticated message exchange from the home agent to the correspondent host [27]. The resulting Mobile IP scheme achieves performance almost equivalent to ours, but requires modifications to the end hosts' IP layer[1] as well as the infrastructure. In contrast, our approach achieves secure, seamless connection migration without a third-party home agent. It also provides a mobile host the ability to pick a mobility mode based on the needs of its applications.

---

[1] In fact, the draft allows on-path routers to cache the care-of addresses instead of the end host, but this requires modifying yet another level of infrastructure.

IPv6 provides native support for multiple simultaneous host addresses, and Mobile IPv6 provides mobility support for IPv6 in much the same fashion as Mobile IP for IPv4. IPv6 extensions allow for the specification of a care-of address, which explicitly separates the role of the EID (the host's canonical IP address) and routing location (the care-of address). Gupta and Reddy propose a similar redirection mechanism for IPv4 through the use of ICMP-like control messages which establish care-of bindings at the end hosts [11].

Mysore and Bharghavan propose an interesting approach to network-layer mobility [24], where each mobile host is issued a permanent Class D IP multicast address that can serve as a unique EID. If multicast were widely deployed, this is a promising approach; because a Class D EID has the benefit of being directly routable by the routing infrastructure, it removes the need for an explicit care-of address. However, this scheme requires a robust, scalable, and efficient multicast infrastructure for a large number of sparse groups.

2.2 Higher-layer methods

The home-agent-based approach has also been applied at the transport layer, as in MSOCKS [19], where connection redirection was achieved using a split-connection proxy.

The general idea of using names as a level-of-indirection to handle object and node mobility is part of computer systems folklore. For some years now, people have talked about using the DNS to effect the level-of-indirection needed to support host mobility, but to our knowledge ours is the first specific and complete architecture that uses the DNS to support Internet host mobility. Recently, Adjie-Winoto *et al.* proposed the integration of name resolution and message routing in an Intentional Naming System to implement a "late binding" option that tracks highly mobile services and nodes [2], and it seems possible to improve the performance of that scheme using our connection migration approach.

Our approach differs fundamentally from EID/locator techniques since it requires no additional level of global addressing or indirection, but only a (normally pre-existing) DNS entry and a shared connection key between the two end hosts. Furthermore, unlike previous connection-ID draft proposals such as Huitema's ETCP [12] for TCP connection re-addressing, it requires no modification to the TCP header, packet format, or semantics.[2] Instead, it uses an additional

---

[2] Special RST handling is required on some networks that may rapidly reassign IP addresses; Section 4.5 discusses this issue.

TCP option and the inserts an additional field into the Transmission Control Block (TCB).

There is a large body of work relating to improving TCP performance in wireless and mobile environments [6, 7]. While not the focus of our work, our adherence to standard TCP semantics allows these schemes to continue to work well in our architecture. Furthermore, since end hosts are explicitly notified of mobility, significant performance enhancements can be achieved at the application level [26].

3 An end-to-end architecture

In this section, we describe our end-system mobility architecture. There are three important components in this system: addressing, mobile host location, and connection migration. By giving the mobile host explicit control over its mobility mode, we remove the need for an additional (third-party) home-agent to broker packet routing. The DNS already provides a host location service, and any further control is managed by the communicating peers themselves, triggered by the mobile host when it changes network location.

We assume, like most mobility schemes, that mobile hosts do not change IP addresses more than a few times a minute. We believe this is a reasonable assumption for most common cases of mobility. We emphasize that this does not preclude physical mobility at rapid velocities across a homogeneous link technology, since that can be handled at the physical and link layers, e.g., via link-layer bridging [13].

The rest of this section discusses addressing in a foreign network and host location using the DNS. Section 4 is devoted to a detailed description of TCP connection migration.

3.1 Addressing

The key to the scalability of the Internet architecture is that the IP address serves as a routing locator, reflecting the addressee's point of attachment in the network topology. This enables aggregation based on address prefixes and allows routing to scale well. Our mobility architecture explicitly preserves this crucial property of Internet addressing.

Like Mobile IP, we separate the issues of obtaining an IP address in a foreign domain from locating and seamlessly communicating with mobile hosts. Any suitable mechanism for address allocation may be employed, such as manual assignment, the Dynamic Host Configuration Protocol (DHCP) [8], or an autoconfiguration protocol [35].

While IP addresses fundamentally denote a point of attachment in the Internet topology and say nothing about the identity of the host that may be connected to that attachment point, they have implicitly become associated with other properties as well. For example, they are often used to specify security and access policies as in the case of ingress filtering to alleviate denial-of-service attacks. Our architecture works without violating this trust model and does not require any form of forward or reverse tunneling to maintain seamless connectivity. In a foreign network, a mobile host uses a locally obtained interface address valid in the foreign domain as its source address while communicating with other Internet hosts.

3.2 Locating a mobile host

Once a mobile host obtains an IP address, there are two ways in which it can communicate with correspondent hosts. First, as a client, when it actively opens connections to the correspondent host. In this case, there is no special host location task to be performed in our architecture; using the DNS as before works. However, if the mobile host were to move to another network attachment point during a connection, a new address would be obtained as described in the previous section, and the current connection would continue seamlessly via a secure negotiation with the communicating peer as described in Section 4. If a mobile host were always a client (not an uncommon case today), then no updates need to be made to any third party such as a home agent or the DNS.

To support mobile servers and other applications where Internet hosts actively originate communication with a mobile host, we use the DNS to provide a level of indirection between a host's current location and an invariant end-point identifier. In Mobile IP, a host's home address is the invariant, and all routing (in the absence of route optimization) occurs via the home agent that intercepts packets destined to this invariant. Ours is not a network-layer solution and we can therefore avoid the indirection for every packet transmission. We take advantage of the fact that a hostname lookup is ubiquitously done by most applications that originate communication with a network host, and use the DNS name as the invariant. We believe that this is a good architectural model: a DNS name identifies a host and does not assume anything about the network attachment point to which it may currently be attached, and the indirection occurs only when the initial lookup is done via a control message (a DNS lookup).

This implies that when the mobile host changes its attachment point, it must detect this and change the hostname-to-address ("A-record") mapping in the DNS. Fortunately, both tasks are easy to accomplish, the former by using a user-level daemon as in Mobile IP, and the latter by using the well-understood and widely available secure DNS update protocol [9, 36]. We note that some DHCP servers today issue a DNS update at client boot time when handing out a new address to a known client based on a static MAC-to-DNS table. This augurs well for the incremental deployability of our architecture, since DNS update support is widely available.

The DNS provides a mechanism by which name resolvers can cache name mappings for some period of time, specified in the time-to-live (TTL) field of the A-record. To avoid a stale mapping from being used from the name cache, we set the time-to-live (TTL) field for the A-record of the name of the mobile host to *zero*, which prevents this from being cached.[3] Contrary to what some might expect, this does not cause a significant scaling problem; name lookups for an uncached A-record do not have to start from a root name server, because in general the "NS-record" (name server record) of the mobile host's DNS name is cacheable for a long period of time (many hours by default). This causes the name lookup to start at the name server of the mobile host's domain, which scales well because of administrative delegation of the namespace and DNS server replication in any domain. We note that some content distribution networks for Web server replication of popular sites use the same approach of small-to-zero TTL values to redirect client requests to appropriate servers (e.g., Akamai [3]). There is no central hot spot because the name server records for a domain are themselves cacheable for relatively long periods of time.

Even with uncacheable DNS entries there still exists a possible race condition where a mobile host moves between when a correspondent host receives the result of its DNS query and when it initiates a TCP connection. Assuming a mobile host updates its DNS entry immediately upon reconnection, the chances of such an occurrence are quite small, but non-zero, especially for a mobile host that makes frequent moves. In this case, the correspondent host will attempt to open a TCP connection to the mobile host's old address, and has no automatic fail-over mechanism.

In this case, the application must perform another DNS lookup to find the new location of the mobile host. We note that the trend towards dynamic DNS records has caused such application-level retries to find their way into applications already—for instance, current FreeBSD `telnet` and `rsh` applications try alternate addresses if an initial connec-

---

[3] Modern versions of BIND honor this correctly.

tion fails to a host that has multiple DNS A-records. It seems to be only a minor addition to refresh DNS bindings if connection establishment fails.

4 TCP connection migration

A TCP connection [32] is uniquely identified by a 4-tuple: ⟨*source address*, *source port*, *dest address*, *dest port*⟩. Packets addressed to a different address, even if successfully delivered to the TCP stack on the mobile host, must not be demultiplexed to a connection established from a different address. Similarly, packets from a new address are also not associated with connections established from a previous address. This is crucial to the proper operation of servers on well-known ports.

We propose a new *Migrate* TCP option, included in SYN segments, that identifies a SYN packet as part of a previously established connection, rather than a request for a new connection. This Migrate option contains a *token* that identifies a previously established connection on the same destination ⟨*address*, *port*⟩ pair. The token is negotiated during initial connection establishment through the use of a *Migrate-Permitted* option. After a successful token negotiation, TCP connections may be uniquely identified by either their traditional ⟨*source address*, *source port*, *dest address*, *dest port*⟩ 4-tuple, or a new ⟨*source address*, *source port*, *token*⟩ triple on each host.

A mobile host may restart a previously-established TCP connection from a new address by sending a special Migrate SYN packet that contains the token identifying the previous connection. The fixed host will than re-synchronize the connection with the mobile host at the new end point. A migrated connection maintains the same control block and state (with a different end point, of course), including the sequence number space, so any necessary retransmissions can be requested in the standard fashion. This also ensures that SACK and any similar options continue to operate properly. Furthermore, any options negotiated on the initial SYN exchange remain in effect after connection migration, and need not be resent in a Migrate SYN.[4]

Since SYN segments consume a byte in the TCP sequence number space, Migrate SYNs are issued with the same sequence number as the last transmitted byte of data. This results in two bytes of data in a migrated TCP connection with the same sequence number (the

---

[4] They can be, if needed. For example, it might be useful to renegotiate a new maximum segment size (MSS) reflecting the properties of the new path. We have not yet explored this in detail.

new SYN and the previously-transmitted actual data), but this is not a problem since the Migrate SYN segment need never be explicitly acknowledged. Any packet received from the fixed host by a migrating host at the mobile host's new address that has a sequence number in the appropriate window for the current connection implicitly acknowledges the Migrate SYN. Similarly, any further segments from the mobile host provide the fixed host an implicit acknowledgement of its SYN/ACK. Thus, there is exactly one byte in the sequence space that needs explicit acknowledgement even when the Migrate SYN is used.

4.1 An example

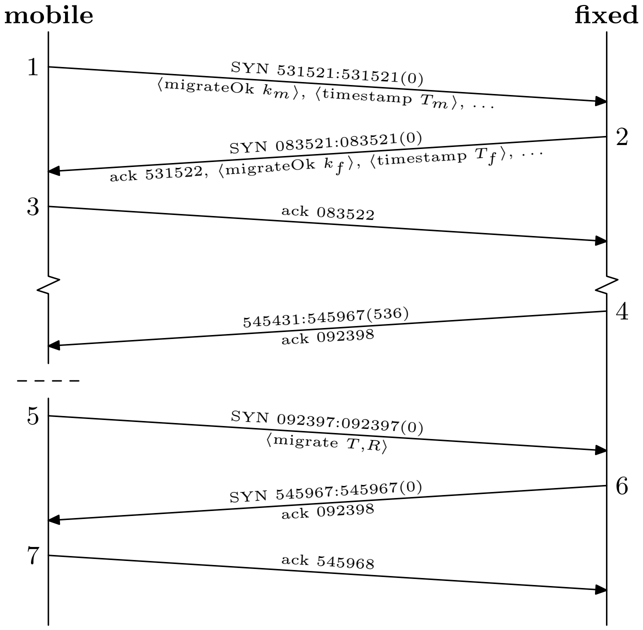


Fig. 1. TCP Connection Migration

Figure 1 shows a sample connection where a mobile client connects to a fixed host and later moves to a new address. The mobile client initiates the TCP connection in standard fashion in message 1, including a Migrate-Permitted option in the SYN packet. The values k_m and T_m are parameters used in the token negotiation, described in Section 4.3. The fixed server, with a migrate-compliant TCP stack, indicates its acceptance of the Migrate-Permitted option by including the Migrate-Permitted option in its response (message 2). The client completes the three-way handshake with message 3, an ACK. The connection then proceeds as any other TCP connection would, until message 4, the last packet from the fixed host to the mobile host at its current address.

At some time later the mobile host moves to a new address, and notifies the fixed server by sending a SYN packet from its new address in message 5. This SYN includes the Migrate option, which contains the previously computed connection token as part of a migration request. Note that the sequence number of this Migrate SYN segment is the same as the last byte of transmitted data. The server responds in kind in message 6, also using the sequence number of its last transmitted byte of data. The ACK, however, is from the same sequence space as the previous connection. While in this example it acknowledges the same sequence number as the SYN that generated it, it could be the case that segments were lost during a period of disconnect while the mobile host moves, and that the ACK will be a duplicate ACK for the last successfully received in-sequence byte. Since it is addressed to the mobile host's new location, however, it serves as an implicit ACK of the SYN as well. Upon receipt of this SYN/ACK, the mobile host similarly ACKs in the previous sequence space, and the connection resumes as before. All of the options negotiated on the initial SYN except the Migrate-Permitted option are still in effect, and need not be replicated in this or any subsequent migrations.

4.2 Securing the migration

It is possible to partially hijack TCP connections if an attacker can guess the sequence space being used by the connection [22]. With the Migrate options, an attacker who can guess both the sequence space and the connection token can hijack the connection completely. Furthermore, the ability to generate a Migrate SYN from anywhere greatly increases the connection's exposure. While ingress filtering can be used to prevent connection hijacking by attackers not on the path between the end hosts, such methods are ineffective in our case. We must therefore take care to secure the connection token.

The problem is relatively easy to solve if IP security (IPsec) [5] were deployed. While the spectrum of approaches that could be used is outside the scope of this paper, we note that IPsec provides sufficient mechanisms to secure migrateable connections. Currently, however, IPsec has not found wide-spread deployment. Hence, we provide a mechanism to self-secure the Migrate options. End hosts may elect to secretly negotiate an unguessable connection token, which then reduces the security of a migrateable TCP connection to that of a standard TCP connection, since no additional attacks are possible against a migrateable connection without guessing the token, and any attack against a standard TCP connection clearly remains feasible against a migrateable TCP connection.

An unguessable connection token is secured with a secret *connection key*. Since any host that obtains the connection key could fabricate the token and issue a Migrate request, we select the key with an Elliptic Curve Diffie-Hellman key exchange [37], as described below. Hosts using IPsec, or unconcerned with connection security, may choose to disable key negotiation to avoid excess computation.

4.3 Migrate-permitted option

Hosts wishing to initiate a migrateable TCP connection send a Migrate-Permitted option in the initial SYN segment. Similar to the SACK-Permitted option [20], it should only be sent on SYN segments, and not during an established connection. Additionally, hosts wishing to cryptographically secure the connection token may conduct an Elliptic Curve Diffie-Hellman (ECDH) key exchange through the option negotiation. (Elliptic Curve Diffie-Hellman is preferred to other methods of key establishment due to its high security-to-bit-length ratio. Readers unfamiliar with Elliptic Curve cryptography can find the necessary background material in [4].)

<table>
<tr><td>Kind: 15</td><td>Length = 3/20</td><td>Curve Name</td><td>ECDH PK</td></tr>
<tr><td colspan="4">ECDH Public Key (cont.)</td></tr>
<tr><td colspan="4">ECDH Public Key (cont.)</td></tr>
<tr><td colspan="4">ECDH Public Key (cont.)</td></tr>
<tr><td colspan="4">ECDH Public Key (cont.)</td></tr>
</table>

Fig. 2. TCP Migrate-Permitted option

As seen in figure 2, the Migrate-Permitted option comes in two variants—the insecure version, of length 3, and the secure version, with length 20. The secure version is used to negotiate a secret connection key, and contains an 8-bit *Curve Name* and a 136-bit *ECDH Public Key* fragment. The curve name field selects a particular set of domain parameters (the curve, underlying finite field, F, and its representation, the generating point, P, and the order of P, n), as specified in [4]. Use of the insecure version, which contains only a Curve Name field (which must be set to zero) allows the end host to skip the key negotiation process. In that case, the connection key is set to all zeros.

The secure variant of the Migrate-Permitted option also requires the use of the Timestamp [15] option in order to store up to 200 bits of ECDH keying material. The EDCH Public Key is encoded using the compressed conversion routine described in [4, Section 4.3.6]. The 136 least-significant bits are stored in the EDCH Public Key field of the Migrate-Permitted option, while the remaining 64 bits of the key are encoded in the Timestamp option. The timestamp option, while often included, is not used on SYN segments. The Protection Against Wrapped Sequence Numbers (PAWS) [15] check is only performed on synchronized connections, which by definition [32] includes only segments after the three-way handshake. Similarly, the Round-Trip Time Measurement (RTTM) [15] procedure only functions when a timestamp has been echoed—clearly this is never the case on an initial SYN segment. Hence the value of the Timestamp option on SYN segments is entirely irrelevant to current TCP stacks. Legacy TCP stacks will never receive a Migrate-Permitted option on a SYN/ACK, hence the Timestamp option will be processed normally. Special handling is only required for the SYN/ACK and following ACK segment on connections that have negotiated the Migrate-Permitted option, as Timestamp fields on these segments will not contain timestamps. Hence the RTTM algorithm must not be invoked for SYN/ACK or initial ACK segments of connections that have negotiated the Migrate-Permitted option.

The Timestamp *TSVal* field contains the 32 most-significant bits of the public key, while the *TSecr* field contains the next 32 most-significant bits. These two components, combined with the 136-bit EDCH Public Key field of the Migrate-Permitted option, constitute the host's public key, k. If the public key is less than 200 bits, it is left-padded with zeros. For any host, i, k_i is generated by selecting a random number, $X_i \in [1, n-1]$, where n is the order of P, and computing

$$k_i = X_i * P$$

The $*$ operation is the scalar multiplication operation over the field F. The security of the connection hinges on the secrecy of the negotiated key, hence X_i should be randomly generated and stored in the control block for each new connection. Any necessary retransmissions of the SYN or SYN/ACK must include identical values for the Migrate-Permitted and Timestamp option.

Upon receipt of an initial SYN with a Migrate-Permitted option, a host, j, with a compliant TCP stack must include a Migrate-Permitted option (and a Timestamp option if the secure variant is used) in its SYN/ACK segment. It similarly selects a random $X_j \in [1, n-1]$ which it uses to construct k_j, its public key, which it sends in the same fashion.

After the initiating host's reception of the SYN/ACK with the Migrate-Permitted and Timestamp options, both hosts can then compute a shared secret key, K, as specified in [37]:

$$K = k_i * X_j = k_j * X_i$$

This secret key is then used to compute a connection validation token. This token, T, is computed by hashing together the key and the initial sequence numbers N_i and N_j using the Secure Hash Algorithm (SHA-1) [25] in the following fashion (recall that host i initiated the connection with an active open, and host j is performing a passive open):

$$T = SHA1(N_i, N_j, K)$$

While SHA-1 produces a 160-bit hash, all but the 64 most-significant bits are discarded, resulting in a cryptographically-secure 64-bit token that is unique to the particular connection. Since SHA-1 is collision-resistant, the chance that another connection on the same ⟨*address*, *port*⟩ pair has an identical token is extremely unlikely. If a collision is detected, however, the connection must be aborted by sending a RST segment. (The host performing a passive open can check for collisions before issuing a SYN/ACK, and select a new random X_j until a unique token is obtained. Hence the only chance of collision occurs on the host performing the active open.)

4.4 Migrate option

The Migrate option is used to request the migration of a currently open TCP connection to a new address. It is sent in a SYN segment to a host with which a previously-established connection already exists (in the ESTABLISHED or FIN_WAIT states), over which the Migrate-Permitted option has been negotiated.

There are two 64-bit fields in a Migrate option: a *token*, and a *request.* In addition, there is an 8-bit sequence number field, *reqNo*, which

<table>
<tr><td></td><td>Kind: 16</td><td>Length = 19</td><td>ReqNo</td></tr>
<tr><td colspan="4">Token</td></tr>
<tr><td colspan="4">Token (cont.)</td></tr>
<tr><td colspan="4">Request</td></tr>
<tr><td colspan="4">Request (cont.)</td></tr>
</table>

Fig. 3. TCP Migrate option

must be monotonically increasing with each new migrate request issued by an end host for a connection. (The sequence number allows correspondent hosts to ensure Migrate SYNs were not reordered by the network. Sequence space wrap-around is dealt with in the standard fashion.) The token is simply the 64 most-significant bits of the connection's SHA-1 hash as computed in the Migrate-Permitted option exchange. The request, R, is similarly the 64 most-significant bits of a SHA-1 hash calculated from the sequence number of the connection initial sequence numbers N, Migrate SYN segment, S, the connection key, K, and the request sequence number, I.

$$R = SHA1(N_i, N_j, K, S, I)$$

SYN segments may now correctly arrive on a bound port not in the LISTEN state. They should be processed only if they contain the Migrate option as specified above. Otherwise, they should be treated as specified in [32]. Upon receipt of a SYN packet with the Migrate option, a TCP stack that supports migration attempts to locate the connection on the receiving port with the corresponding token. The token values for each connection were precomputed at connection establishment, reducing the search to a hash lookup.

If the token is valid, meaning an established connection on this ⟨*address*, *port*⟩ pair has the same token, and the *reqNo* is greater than any previously received migrate request, the fixed host then computes $R = SHA1(N_i, N_j, K, S, I)$ as described above, and compares it with the value of the request in the Migrate SYN. If the comparison fails, or the token was invalid, a RST is sent to the address and port issuing the Migrate SYN, and the SYN ignored. If, on the other hand, the token and request are valid, but the reqNo is smaller than a previously received request, the SYN is assumed to be out-of-order and silently discarded.

If the reqNo is identical to the most recently received migrate request this SYN is assumed to be a duplicate of the most recently received SYN, and processed accordingly.

Otherwise, the destination address and port[5] associated with the matching connection should be updated to reflect the source of the Migrate SYN, and a SYN/ACK packet generated, with the ACK field set to the last received contiguous byte of data, and the connection placed in the SYN_RCVD state. Upon receipt of an ACK, the connection continues as before.

4.5 MIGRATE_WAIT state

This section assumes that the reader is familiar with the TCP state machine and transitions [34, Chapter 18].

Special processing of TCP RST messages is required with migrateable connections, as a mobile host's old IP address may be reassigned before it has issued a migrate request to the fixed host. Figure 4 shows the modified TCP state transition diagram for connections that have successfully negotiated the Migrate-Permitted option. The receipt of a RST that passes the standard sequence number checks in the ESTABLISHED state does not immediately terminate the connection, as specified in [32]. Instead, the connection is placed into a new *MIGRATE_WAIT* state. (A similar, but far less likely situation can occur if the fixed host is in the FIN_WAIT1 state—the application on the fixed host has closed the connection, but there remains data in the connection buffer to be transmitted. For simplicity, these additional state transitions are not shown in figure 4.)

Connections in the MIGRATE_WAIT state function as if they were in the ESTABLISHED state, except that they do not emit any segments (data or ACKs), and are moved to CLOSED if they remain in MIGRATE_WAIT for over a specified period of time. We recommend using the *2MSL* ([32] specifies a Maximum Segment Lifetime (MSL) as 2 minutes, but common implementations also use values of 1 minute or 30 seconds for MSL [34]) period of time specified for the TIME_WAIT state.

Any segments received while in the MIGRATE_WAIT state should be processed as in the ESTABLISHED state, except that no ACKs should be generated. The only way a connection is removed from the MIGRATE_WAIT state is on the receipt of a Migrate SYN with the

---

[5] Migrated connections will generally originate from the same port as before. However, if the mobile host is behind a NAT, it is possible the connection has been mapped to a different port.

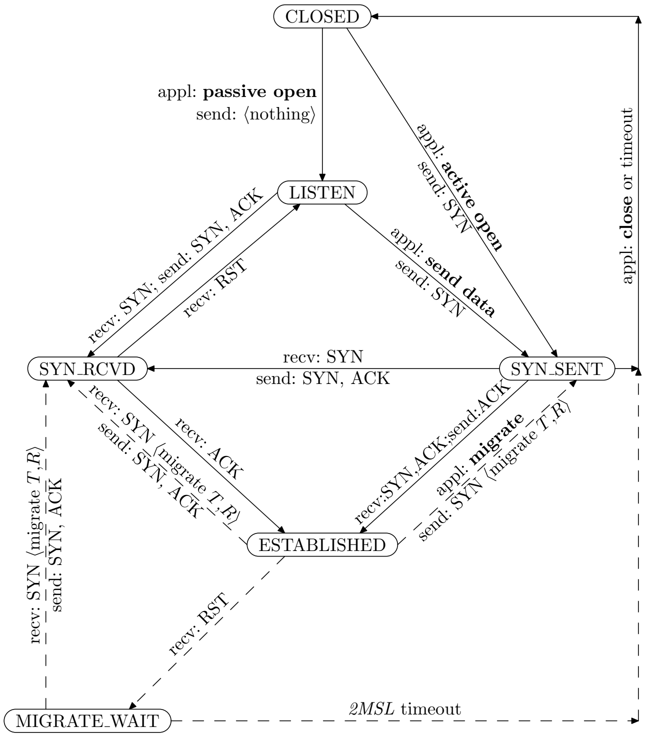


Fig. 4. Partial TCP state transition diagram with Migrate transitions (adapted from [34, figure 18.12])

corresponding connection key. The connection then responds in the same fashion as if it were in the ESTABLISHED state when it received the SYN.

The MIGRATE_WAIT state prevents connections from being inadvertently dropped if the address allocation policy on the mobile host's previous network reassigns the mobile host's old IP address before the mobile host has reconnected at a new location and had a chance to migrate the connection. It also prevents the continued retransmission of data to an unreachable host.

This passive approach to disconnection discovery is preferred over an active, mobile-initiated squelch message because any such message could be lost.[6] Furthermore, a mobile host may not have sufficient (if any) notice of address reassignment to issue such messages. As an added performance enhancement, however, mobile hosts aware of an impending migration may themselves emit a special RST to the peer, which will force the connection into MIGRATE_WAIT, preventing additional packet transmission until the mobile host has successfully relocated, although such action invokes the strict 2MSL time bound on the allowable delay for host relocation and connection migration.

5 Security issues

An end-to-end approach to mobility simplifies the trust relationships required to securely support end-host mobility compared to network-layer approaches such as Mobile IP. In addition to the relationship between a mobile host and any proxies or home agents, several Mobile IP-based proposals require that a correspondent host in communication with a mobile host assume the responsibility of authenticating communication with an arbitrary set of foreign agents. In their route optimization draft [29], Perkins and Johnson state:

> One of the most difficult aspects of Route Optimization for Mobile IP in the Internet today is that of providing authentication for all messages that affect the routing of datagrams to a mobile node.

Since no third parties are required or even authorized to speak on the mobile host's behalf in an end-to-end architecture, the only trust relationship required for secure relocation is between the mobile and correspondent host. Clearly they already must have a level of trust

---

[6] And any guaranteed-reliable transmission mechanism could take unbounded time.

commensurate with the nature of their communications since they chose to communicate in the first place.

Regardless of the simplicity of trust relationships, there remains the possibility that untrusted parties could launch attacks against the end hosts or connections between them utilizing either dynamic DNS updates or the Migrate and Migrate-Permitted options. The security of dynamic DNS updates is addressed in RFC 2137 [9], resting on the strength of the digital signature scheme used to authenticate mobile hosts.

Possible attacks against the Migrate TCP options include both denial-of-service attacks and methods of migrating connections away from their appropriate end hosts. We discuss these attacks below, and either show why the Migrate options are not vulnerable, or explain why the attack presents no additional threat in relation to standard TCP.

5.1 Denial of service

SYN flooding is a common form of Denial-of-Service (DoS) attack, and most modern TCP implementations have taken great care to avoid consuming unnecessary resources unless a three-way handshake is complete. To validate a Migrate request, the correspondent host performs a significant computation (the SHA-1 hash), which implies we need to be especially vigilant against DoS attacks that attempt to deplete the CPU resources of a target host. The validation is not performed unless an attacker succeeds in guessing a valid, pre-computable token (with a 1 in 2^{64} probability); since a RST message is generated if either the token or the request is invalid, an attacker has no way to identify when it has found a valid token. Because a would-be attacker would therefore have to issue roughly 2^{63} Migrate SYNs to force a request validation, we argue that the TCP Migrate option does not introduce any additional DoS concerns above standard TCP.

5.2 Connection hijacking

Since a Migrate request contains a hash of both the SYN segment's sequence number and migrate request sequence number, a replayed Migrate option can only be used until either a new byte of data or another migrate connection is sent on the connection. Since self-migration is not allowed, duplicate Migrate SYNs (received outside of the three-way handshake) are ignored by the peer TCP. If, however, the mobile host moves rapidly to a another new location, a replayed Migrate SYN could be used to migrate the connection back to the mobile host's previous IP, which may have been subsequently assumed by the attacker. In order to prevent this attack, the Migrate Request option processing ignores the source address and port in duplicate packets, as a valid request from a relocated mobile host would include a higher request number.

More worrisome, however, is the fact that once a Migrate SYN has been transmitted, the token is known by any hosts on the new path, and denial-of-service attacks could be launched by sending bogus Migrate SYNs with valid tokens. If a mobile host includes a new Migrate-Permitted option in its Migrate SYN, however, the window of opportunity when the previous connection token can be used (if it was snooped) is quite small—only until the new three-way handshake is successfully completed.

5.3 Key security

The connection key used by the Migrate option is negotiated via Elliptic Curve Diffie-Hellman to make it extremely difficult even for hosts that can eavesdrop on the connection in both directions to guess the key. Without sufficient information to verify possible keys off-line, an attacker would have to continually generate Migrate SYNs and transmit them to one of the end hosts, hoping to receive a SYN/ACK in response to a correct guess. Clearly such an attack is of little concern in practice, as the expected 2^{63} SYN packets required to successfully guess the key would generate sufficient load as to be a DoS problem in and of themselves.

Hosts that lie on the path between end hosts, however, have sufficient information (namely the two Elliptic Curve Diffie-Hellman components) to launch an attack against the Elliptic Curve system itself. The best known attack is a distributed version of Pollard's rho-algorithm [31], which [18] uses to show that a 193-bit EC system would require $8.52 \cdot 10^{14}$ MIPS years, or about $1.89 \cdot 10^{12}$ years on a 450Mhz Pentium II, to defeat.

While this seems more than secure against ordinary attackers, an extremely well-financed attacker might be able to launch such an attack on a long-running connection in the not-too-distant future. The obvious response is to increase the key space. Unfortunately, we are restricted by the 40-byte limitation on TCP options. Given the prevalence of the MSS (4 bytes), Window Scale (3 bytes), SACK Permitted (2 bytes), and Timestamp (10 bytes) options (of which we are already using 8 bytes) in today's SYN segments, the 20-byte Migrate-Permitted option is already as large as is feasible. We argue that further securing the connection key against brute-force attacks from hosts on the path between the two end hosts is largely irrelevant, given the ability of such hosts to launch man-in-the-middle attacks against TCP with much less difficulty!

The security of TCP connections, migrateable or not, continues to remain with the authentication of end hosts, and the establishment of strong session keys to authenticate ongoing communication. Although we have taken care to ensure the Migrate option does not further decrease the security of TCP connections, the latter are inherently insecure, as IP address spoofing and sequence number guessing are not very difficult. Hence we strongly caution users concerned with connection security to use additional application-layer cryptographic techniques to authenticate end points and the payload traffic.

5.4 IPsec

When used in conjunction with IPsec [5], there are additional issues raised by the use of the Migrate options. IPsec Security Associations (SAs) are established on an IP-address basis. When a connection with an associated SA is migrated, a new SA must be established with the new destination address before communication is resumed. If the establishment of a this new SA conflicts with existing policy, the connection is dropped. This seemingly unfortunate result is actually appropriate. Since IPsec's Security Policy Database (SPD) is keyed on IP network address, the policies specified within speak to a belief about the trustworthiness of a particular portion of the network.

If a mobile host attaches to a foreign network, any security assumptions based on its normal point of attachment are invalid. If the end host itself continues to have sufficient credentials independent of its point of attachment, an end-to-end authentication method should be used, and a secure tunnel established for communication over the untrusted network. A discussion of such techniques is outside of the scope of this document.

6 Implementation

We have implemented this architecture in the Linux 2.2.15 kernel, using Bind 8.2.2-P3 as the name server for mobile hosts. The IPv4 TCP stack has been modified to support the Migrate options. Connection migration can be affected through two methods. Applications with open connections may explicitly request a migration by issuing an `ioctl()`

on the connection's file descriptor specifying the address to migrate to. Most current applications, however, lack a notification method so the system can inform them the host has moved. Hence we also provide a mechanism for processes to migrate open connections, regardless of whether they have the file descriptor open or not.

This is done through the Linux `/proc` file system. A directory `/proc/net/migrate` contains files of the form *source address*`:`*source port*`->`*dest address*`:`*dest port* for each open connection that has successfully negotiated the Migrate-Permitted option. These files are owned by the user associated with the process that opened the connection. Any process with appropriate permissions can then write a new IP address to these files, causing the corresponding connection to be migrated to the specified address. This method has the added benefit of being readily accessed by a user directly through the command line.

It is expected that mobile hosts will run a mobility daemon that tracks current points of network attachment, and migrates open connections based on some policy about the user's preference for certain methods of attachment. For instance, when an 802.11 interface comes up on a laptop that previously established connections on a CDPD link, it seems likely that the user would opt to migrate most open connections to the address associated with the 802.11 link. Similarly the daemon could watch for address changes on attached interfaces (possibly as a result of DHCP lease expirations and renewals) and migrate connections appropriately. We plan to implement such a daemon in the near future.

6.1 Experiments

Figure 5 shows the network topology used to gather the TCP traces shown in figures 6 and 7. The traces were collected at the fixed basestation, which is on the path between the fixed host and both mobile host locations. We conducted TCP bulk transfers from a server on the fixed host to a client on the mobile host. The client initiates the connection from one location, and migrates to another location at some later point. Both mobile host locations use identical connections, a 19.2Kbps serial link with $\approx$100ms round-trip latency. The basestation and fixed host are on a 100Mbps Ethernet segment, hence the link to the mobile host is the connection bottleneck. This topology is intentionally simple in order to isolate the various subtleties of migrating TCP connections, as discussed below.

Figure 6 shows the TCP sequence trace of a migrated TCP connection. At time $t \approx 4.9s$ the mobile host moved to a new address and

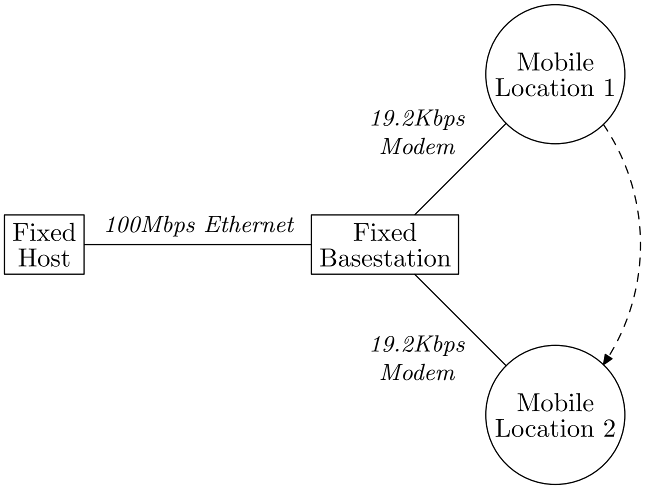


Fig. 5. Network topology used for migration experiments

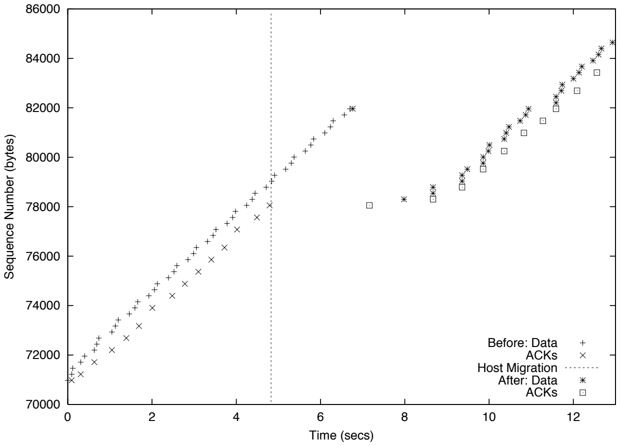


Fig. 6. A TCP connection sequence trace showing the migration of an open connection

issued a Migrate SYN, as depicted by the dotted line. Since the host is no longer attached at its previous address, all of the enqueued segments at the bottleneck are lost. (The amount of lost data is bounded by the advertised receive window of the mobile host. A host that moves frequently across low-bandwidth connections may wish to advertise a smaller receive window to reduce the number of wasted segments.) Finally, at $t \approx 6.8s$ the fixed host's SYN/ACK passes through the bottleneck, and is ACKed by the fixed host a RTT later.

The fixed host does not immediately restart data transmissions because the TCP Migrate options do not change the congestion-avoidance or retransmission behavior of TCP. The sender is still waiting for ACKs for the lost segments; as far as it is concerned, it has only received two (identical) ACKs—the original ACK, and one duplicate as part of the Migrate SYN three-way handshake.

Finally, at $t \approx 7.8s$ the retransmission timer expires (the interval is from the first ACK, sent earlier at $t \approx 4.9s$) and the fixed host retransmits the first of the lost segments. It is immediately acknowledged by the mobile host, and TCP resumes transmission in slow-start after the timeout.

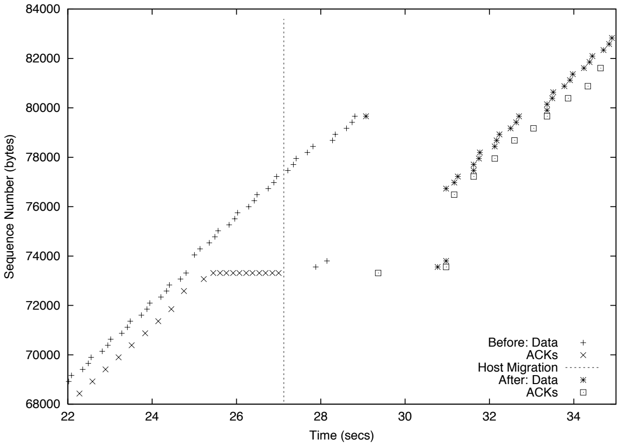


Fig. 7. A TCP Migrate connection (with SACK) sequence trace with losses just before migration

Figure 7 shows the TCP sequence trace of a similar migrate TCP connection. As before, the dashed line indicates the mobile host issued a migrate request at time $t \approx 27.1s$. This time, however, there were additional losses on the connection that occurred just before the migration, as can be seen at $t \approx 24.9s$. These segments are fast-retransmitted, and pass through the bottleneck at $t \approx 28s$ due to the DUP-ACKs generated by the remaining SYNs. Unfortunately, this is after the mobile host has migrated, so they, along with all the segments addressed to the mobile host's initial address after $t \approx 27.1s$, are lost.

At $t \approx 29s$, the Migrate SYN/ACK makes it out of the queue at the bottleneck, and the mobile host immediately generates an ACK. As in the previous example, however, the fixed host is still awaiting ACKs for previously transmitted segments. It is only at $t \approx 31s$ that the timer expires and the missing segments are retransmitted. Notice that because SACK prevents the retransmission of the previously-received segments, only those segments lost due to the mobile host's address change are retransmitted, and the connection continues as before. The success of this trace demonstrates that the Migrate options work well with SACK due to the consistency of the sequence space across migrations.

6.2 Performance enhancements

Several enhancements can be made by implementations to improve overall connection throughput during connection migration. The most obvious of these is issuing three DUP-ACKs immediately after a migrate request, thereby triggering the fast-retransmit algorithm and avoiding the timeout seen in the previous example [7]. By preempting the timeout, the connection further avoids dropping into slow-start and congestion avoidance.

Such techniques should be used with care, however, as they assume the available bandwidth of the new path between mobile and fixed host is on the same order-of-magnitude as the previous path. For migrations across homogeneous technologies this may be a reasonable assumption. When moving from local to wide-area technologies, however, there may be order-of-magnitude discrepancies in the available bandwidth. Hence we do not include such speed-ups in the TCP Migrate specification, and leave it to particular implementations to responsibly evaluate the circumstances and provide behavior compatible with standard TCP.

7 Deployment issues

As with any scheme for mobility support, there are some deployment issues to be addressed. By pushing the implementation of mobility mechanisms—connection migration in particular—to the end points, our system requires changes to each transport protocol. Fortunately, our TCP connection migration protocol can be generalized to other UDP-based protocols with little difficulty. Significant examples include streaming protocols such as RTP and proprietary protocols like Real, Quicktime and Netshow. We note that most of these already have a control channel used for congestion and quality control, and such applications would likely wish to be informed of changes due to mobility as well. Furthermore, we argue that not all applications *require* network-layer mobility, especially those characterized by short transactions where an application-level retry of the transaction is easy to perform; we therefore make the case using the end-to-end argument that mobility might be best implemented as a higher-level, end-to-end function just like reliability.

Perhaps the biggest limitation of our approach is that both peers cannot move *simultaneously.*[7] Because our scheme has no anchor point like Mobile IP's home agent, any IP address change must be completed before the other can proceed. We do not view this as a serious limitation to the widespread applicability of the protocol, since we are primarily targeting infrastructure-based rather than ad-hoc network topologies in this work.

In addition to these two limitations, there are several issues that crop up when one considers presently-deployed applications. While it is currently possible for Internet hosts to be re-addressed while operating (due to a DHCP lease expiration or similar event), it is quite rare. Hence some applications have made assumptions about the stability of network addresses, which are no longer valid in our architecture. We discuss some of these issues below.

7.1 Address caching

There is a class of applications that store IP addresses within the application, and communicate these addresses to a remote host. Such applications would not function properly under our architecture. They are readily identifiable, however, as another currently widely-deployed technology also breaks such applications: Network Address Translators (NATs). While the wisdom of Network Address Translation is a hotly debated topic, there is little chance it will disappear any time soon. Hence most applications designed today take care not to transmit addresses as part of the application-layer communication, and therefore

---

[7] "Simultaneously" is defined as whenever the intervals between address change and the (would-be) reception of the Migrate SYN by the corresponding host for both end hosts overlap.

will also work in our architecture. In fact, one can make the case that such applications are broken, since IP addresses are only identifiers of attachment points, not hosts.

Another, larger class of applications cache the results of *gethostbyname()*, and may not perform further hostname resolution.[8] Furthermore, DNS resolvers themselves cache hostname bindings as discussed in Section 3. Unfortunately many older name servers enforce a local TTL minimum, often set to five minutes. Since newer versions of popular name servers adhere to the TTL specified in the returned resource record, this problem should disappear as upgrades are made.

7.2 Proxies and NATs

Proxies actually help the deployment of our scheme, as we only need to modify the proxy itself, and all communications through the proxy will support mobility. Similarly, NATs can also provide transparent support without remote system modification. In fact, a NAT doesn't even need a modified TCP stack. It need only snoop on TCP SYNs (which it does anyway), note the presence of a Migrate-Permitted option, and snoop for the SYN/ACK (which it does anyway). If the SYN/ACK does not contain a Migrate-Permitted option, the NAT can support connection migration internal to its network by inserting a corresponding Migrate-Permitted option, and continuing to snoop the flow looking for any Migrate SYNs. It need only fabricate a corresponding SYN/ACK and update its address-to-port mappings, without passing anything to the end host. Further, by avoiding any explicit addressing in migrate requests, the Migrate options function properly though legacy NATs, and even allow a mobile host to move between NATs, as connections may change not only address but port as well.

7.3 Non-transactional UDP applications

Many UDP applications are transactional in nature. UDP is, by definition, a datagram protocol, and an inopportune change of IP address is only one of many reasons for an unsuccessful UDP transaction. The transaction will need to be retried, although a new hostname binding should be obtained first.

There exists at least one glaring exception to this rule. The Network File System protocol (NFS) represents one of the most prevalent UDP

---

[8] Some popular Web browsers display this behavior.

applications in use today and uses IP addresses in its mount points.[9] We believe, given the characteristics of network links likely to be encountered by mobile hosts, it is likely that NFS-over-TCP is a better choice than UDP. Otherwise, a mobile host would need to dismount and re-mount NFS filesystems upon reconnection.

8 Conclusion

This paper presents an end-to-end architecture for Internet host mobility that makes no changes to the underlying IP communication substrate. It uses secure updates to the DNS upon an address change to allow Internet hosts to locate a mobile host, and a set of connection migration options to securely and efficiently negotiate a change in the IP address of a peer without breaking the end-to-end connection. We have implemented this architecture in the Linux operating system and are encouraged by the ease with which mobility can be achieved without any router support, the flexibility to mobile hosts provided by it, and performance comparable to Mobile IP with route optimization.

Our architecture allows end systems to choose a mobility mode best suited to their needs. Routing paths are efficient with no triangle routing, and any connection involving the mobile host shares fate only with the communicating peer and not with any other entity like a home agent. When a mobile host is in a foreign network and communicating with another host, the disruption in connectivity caused by a sudden IP address change is proportional to the round-trip time of the connection. When a mobile host accepts no passive connections, the protocol does not require even the DNS update notification, and seamless connectivity across host mobility is achieved using completely end-to-end machinery.

The security of our approach is based on a combination of the well-documented secure DNS update protocol in conjunction with a new secure connection migration mechanism. Our architecture and implementation function across a variety of other components of the Internet architecture, including firewalls, NATs, proxies, IPsec, and IPv6. We believe that our architecture scales well even when most Internet hosts become mobile because lookups and updates are distributed across administratively-delegated, replicated DNS servers.

---

[9] We note that most other advanced file systems, such as Coda [23] and newer versions of NFS use TCP, which gives good congestion control and reliability behavior.

We note that our connection migration scheme, the MIGRATE_WAIT state in particular, avoids address assignment race conditions, but does *not* support host disconnectivity. Hence, as with Mobile IP and other mobility schemes, TCP connections may be lost if the mobile host's relocation is accompanied by a prolonged period of disconnectivity. We are hopeful our end-to-end approach may be extended to support general host disconnectivity as well.

9 Acknowledgements

We thank John Ankcorn, Frans Kaashoek, Eddie Kohler, Robert Morris, Srinivasan Seshan, Tim Sheppard, and Karen Sollins for helpful comments on earlier drafts of this paper. We are indebted to David Andersen, who helped improve the security of our initial Migrate scheme, and David Mazieres, who suggested we use Elliptic Curve Diffie-Hellman key exchange for additional key strength.

References

1. August 1988.
2. William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *Proc. ACM SOSP '99*, pages 186–201, December 1999.
3. Akamai Technologies, Inc. `http://www.akamai.com`.
4. American National Standards Institute. Public key cryptography for the financial service industry: The elliptic curve digital signature algorithm. ANSI X9.62 - 1998, January 1999.
5. Randall Atkinson. Security architecture for the internet protocol. RFC 1825, IETF, August 1995.
6. Hari Balakrishnan, Srinivasan Seshan, and Randy H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 1(4):469–481, December 1995.
7. Ramon Caceres and Liviu Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE JSAC*, 13(5), June 1995.
8. Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, IETF, March 1997.
9. Donald E. Eastlake, 3rd. Secure domain name system dynamic update. RFC 2137, IETF, April 1997.
10. Paul Ferguson and Daniel Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, IETF, January 1998.

11. Sumit Gupta and A. L. Narasimha Reddy. A client oriented, IP level redirection mechanism. In *Proc. IEEE Infocom '99*, March 1999.
12. Christian Huitema. Multi-homed TCP. Internet Draft, IETF, May 1995. (expired).
13. IEEE. Wireless medium access control (MAC) and physical layer (PHY) specifications. Standard 802.11, 1999.
14. Van Jacobson. Congestion avoidance and control. In *Proc. ACM SIGCOMM '88* [1], pages 314–329.
15. Van Jacobson, Robert Braden, and David Borman. TCP extensions for high performance. RFC 1323, IETF, May 1992.
16. Anthony D. Joseph, Joshua A. Tauber, and M. Frans Kaashoek. Mobile computing with the rover toolkit. *IEEE Trans. on Computers*, 46(3):337–352, March 1997.
17. Phil Karn. Qualcomm white paper on mobility and IP addressing. `http://people.qualcomm.com/karn/papers/mobility.html`, February 1997.
18. Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. `http://www.cryptosavvy.com`, November 1999.
19. David Maltz and Pravin Bhagwat. MSOCKS: An architecture for transport layer mobility. In *Proc. IEEE Infocom '98*, March 1998.
20. Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP selective acknowledgment options. RFC 2018, IETF, October 1996.
21. Paul V. Mockapetris and Kevin Dunlap. Development of the domain name system. In *Proc. ACM SIGCOMM '88* [1], pages 123–133.
22. Robert T. Morris. A weakness in the 4.2BSD UNIX TCP/IP software. Computing science technical report 117, AT&T Bell Laboratories, Murray Hill, New Jersey, February 1985.
23. Lily B. Mummert, Maria R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proc. ACM SOSP '95*, pages 143–155, December 1995.
24. Jayanth Mysore and Vaduvur Bharghavan. A new multicasting-based architecture for internet host mobility. In *Proc. ACM/IEEE Mobicom '97*, pages 161–172, September 1997.
25. National Institute of Standards and Technology. The Secure Hash Algorithm (SHA-1). NIST FIPS PUB 180-1, U.S. Department of Commerce, April 1995.
26. Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Proc. ACM SOSP '97*, pages 276–287, October 1997.
27. Charles E. Perkins and Pat R. Calhoun. Mobile IP challenge/response extensions. Internet Draft, IETF, February 2000. `draft-ietf-mobileip-challenge-09.txt` (work in progress).
28. Charles E. Perkins and David B. Johnson. Mobility support in IPv6. In *Proc. ACM/IEEE Mobicom '96*, pages 27–37, November 1996.

29. Charles E. Perkins and David B. Johnson. Route optimization in mobile IP. Internet Draft, IETF, February 2000. `draft-ietf-mobileip-optim-09.txt` (work in progress).
30. Charles E. Perkins, ed. IP mobility support. RFC 2002, IETF, October 1996.
31. J. Pollard. Monte carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.
32. John Postel, ed. Transmission Control Protocol. RFC 793, IETF, September 1981.
33. Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM TOCS*, 2(4):277–288, November 1984.
34. W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, Reading, Massachusetts, 1994.
35. Susan Thomson and Thomas Narten. IPv6 stateless address autoconfiguration. RFC 2462, IETF, December 1998.
36. Paul Vixie, Susan Thomson, Yakov Rekhter, and Jim Bound. Dynamic updates in the domain name system (DNS UPDATE). RFC 2136, IETF, April 1997.
37. Robert Zuccherato and Carlisle Adams. Using elliptic curve Diffie-Hellman in the SPKM GSS-API. Internet Draft, IETF, August 1999. `draft-ietf-cat-ecdh-spkm-00.txt` (work in progress).

Proceedings of HotOS VIII,
Elmau/Oberbayern, Germany,
May 2001.

Reconsidering Internet Mobility

Alex C. Snoeren, Hari Balakrishnan and M. Frans Kaashoek [*]

MIT Laboratory for Computer Science
`{snoeren, hari, kaashoek}@lcs.mit.edu`

Abstract. *Despite the popularity of mobile computing platforms, appropriate system support for mobile operation is lacking in the Internet. This paper argues this is not for lack of deployment incentives, but because a comprehensive system architecture that efficiently addresses the needs of mobile applications does not exist. We identify five fundamental issues raised by mobility—location, preservation of communication, disconnection handling, hibernation, and reconnection—and suggest design guidelines for a system that attempts to support Internet mobility.*

In particular, we argue that a good system architecture should (i) eliminate the dependence of higher protocol layers upon lower-layer identifiers; (ii) work with any application-selected naming scheme; (iii) handle (unexpected) network disconnections in a graceful way, exposing its occurrence to applications; and (iv) provide mobility services at the mobile nodes themselves, rather than via proxies. Motivated by these principles, we propose a session-oriented, end-to-end architecture called Migrate*, and briefly examine the set of services it should provide.*

1 Introduction

The proliferation of laptops, handheld computers, cellular phones, and other mobile computing platforms connected to the Internet has triggered much research into system support for mobile networking over the past few years. Yet, when viewed as a large-scale, heterogeneous,

---

[*] This research was funded by DARPA (Grant No. MDA972-99-1-0014), NTT Corporation, Intel, and IBM. Alex C. Snoeren is supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship

distributed system, the Internet is notoriously lacking in any form of general support for mobile operation.

We argue that previous work has failed to comprehensively address several important issues. This paper discusses some of these issues and describes a session-oriented architecture we are developing to preserve end-to-end application-layer connectivity under various mobile conditions.

Mobility raises five fundamental problems:

1. **Locating the mobile host or service:** Before any communication can be initiated, the desired end-point must be located and mapped to an addressable destination.
2. **Preserving communication:** Once a session has been established between end points (typically applications), communication should be robust across changes in the network location of the end points.
3. **Disconnecting gracefully:** Communicating applications should be able to rapidly discern when a disconnection at either end, or a network partition, causes communication to be disrupted.
4. **Hibernating efficiently:** If a communicating host is unavailable for a significant period of time, the system should suspend communications, and appropriately reallocate resources.
5. **Reconnecting quickly:** Communicating peers should detect the resumption of network connectivity in a timely manner. The system should support the resumption of all previously established communication sessions without much extra effort on the part of the applications.

Most current approaches provide varying degrees of support for the first two problems. The last three—disconnection, hibernation, and reconnection—have received little attention outside of the file system context [19]. We argue that a complete—and useful—solution must address all these issues.

One need look no further than interactive terminal applications like `ssh` or `telnet`, one of the Internet's oldest applications, for a practical example of the continuing lack of support for these important components. A user with an open session might pick up her laptop and disconnect from the network. After traveling for some period of time, she reconnects at some other network location and expects that her session continue where it left off. Unfortunately, if there was any activity on the session during the period of disconnectivity, she will find the connection aborted upon reconnection to the network. The particular details of the example are irrelevant, but demonstrate just how lacking current support is, even for this simple scenario.

Based on our own experience developing various mobile protocols and services [3, 5, 14, 26] and documented reports of several other researchers over several years [9, 13, 15, 18, 28], we identify four important guidelines that we believe should be followed as hints in designing an appropriate network architecture for supporting mobile Internet services and applications:

1. *Eliminate lower-layer dependence from higher layers.* A large number of problems arise because many higher layers of the Internet architecture use identifiers from lower layers, assuming they will never change during a connection.
2. *Do not restrict the choice of naming techniques.* Dynamic naming and location-tracking systems play an important role in addressing mobility. In general, whenever an end point moves, it should update a naming system with its new location—but forcing all applications to use a particular naming scheme is both unrealistic and inappropriate.
3. *Handle unexpected disconnections gracefully.* We advocate treating disconnections as a common occurrence, and exposing them to applications as they occur.
4. *Provide support at the end hosts.* Proxies are attractive due to their perceived ease of deployment. However, it becomes markedly more difficult to ensure they are appropriately located when hosts are mobile.

We elaborate upon these guidelines in Section 2. They have served as a guide in our development of an end-to-end, session-oriented system architecture, called *Migrate*, over which mobile networking applications and services can be elegantly layered. We describe our proposed architecture in Section 3, discussing how it addresses four of the five problems mentioned above: preserving communication, and handling disconnection, hibernation, and resumption. We do not provide or enforce a particular location or naming scheme, instead leveraging domain-specific naming services (e.g., DNS, service discovery schemes [3, 12], etc.) for end-point location.

An attractive feature of our architecture is that it accomplishes these tasks without sacrificing common-case performance. Migrate provides generic mechanisms for managing disconnections and reconnections in each application session, and for handling application state and context. We briefly discuss related work in Section 4 before concluding in Section 5.

2 Design guidelines

In this section, we elaborate on our four design guidelines for supporting applications on mobile hosts.

2.1 Eliminate lower-layer dependence

The first step in enabling higher-layer mobility handling is to remove inter-layer dependences. In a 1983 retrospective paper on the DoD Internet Architecture, Cerf wrote [8]: "TCP's [dependence] upon the network and host addresses for part of its connection identifiers" makes "dynamic reconnection" difficult, "a problem . . . which has plagued network designers since the inception of the ARPANET project in 1968." The result is that when the underlying network-layer (IP) address of one of the communicating peers changes, the end-to-end transport-layer (TCP) connection is unable to continue because it has bound to the network-layer identifier, tacitly (but wrongly) assuming its permanence for the duration of the connection.

A host of other problems crop up because of similar linkages. For example, the increasing proliferation of network address translators (NATs) in the middle of the network has caused problems for applications (like FTP) that use network- and transport-layer identifiers as part of their internal state. These problems can be avoided by removing any assumption of stability of lower-layer identifiers. If a higher layer finds it necessary to use a lower-layer identifier as part of its internal state, then the higher layer should allow for it to change, and continue to function across such changes.

Furthermore, each layer should expose relevant changes to higher layers. In today's Internet architecture, applications have almost no control over their network communication because lower layers (for the most part) do not concern themselves with higher-layer requirements. When important changes happen at a lower layer, for example to the network-layer address, they are usually hidden from higher layers. The unfortunate consequence of this is that it makes it hard for any form of adaptation to occur.

For example, a TCP sender attempts to estimate the properties of the network path for the connection. A significant change in the network-layer attachment point often implies that previously discovered path properties are invalid, and need to be rediscovered. This consequence is not limited to classical TCP congestion management—for example, if mobile applications are notified of changes in their environment and given the power to effect appropriate changes, significant improvements in both performance and usability can be realized [19, 21]. Similar results have also been shown in the network layer [9, 13, 30], and in the area of transport optimization over wireless links [5, 7, 26].

2.2 Beware the Siren song of naming

Many researchers have observed that the first problem raised by mobility, namely locating the mobile host or service, can be addressed through a sophisticated naming system, hence most proposals for managing Internet mobility attempt to provide naming and location services as a fundamental part of the mobility system.[1] Unfortunately, the tight binding between naming schemes and mobility support often causes the resulting system to be inefficient or unsuitable for various classes of applications. For example, Mobile IP assumes that the destination of each packet needs to be *independently* located, thereby necessitating a home agent to intercept and forward messages to a mobile host. The utility of alternative proposals to use agile naming [3] or IP multicast [20] for mobility support hinges on widespread deployment of their location systems.

We believe that inexorably binding mobility handling with naming unnecessarily complicates the mobility services, and restricts the ability to integrate advances in naming services. On the face of it, it appears attractive that a “good” naming scheme can provide the level of indirection by which to handle mobility. In practice, however, it is important to recognize and separate two distinct operations. The first is a “location” operation: The process of finding an end point of interest based on an application-specific name. The second is a “tracking” operation: Preserving the peer-to-peer communication in some way. There are two problems with using a new idealized naming scheme: First, there are a large number of ways in which applications describe what they are looking for, which forces this ideal naming scheme to perform the difficult task of accommodating them all. Experience shows that each application is likely to end up using a naming scheme that best suits it (e.g. INS, DNS, JINI, UPnP), rather than suffer the inadequacies of a universal one. Second, if this tracking is done through the same name resolution mechanism, every packet would invoke the resolution process, adding significant overhead and degrading performance.

We therefore suggest that an application use whichever naming scheme is sufficiently adept at providing the appropriate name-to-loca-

---

[1] Indeed, the authors of this paper are guilty of having taken this position in the past.

tion binding in a timely fashion. This service is used at the beginning of a session between peers, or in the (unlikely) event that all peers change their network locations "simultaneously." At all other times, the onus of preserving communication across moves rests with the peers themselves. In the common case when only a subset of the peers moves at a time, the task of reconnection is efficiently handled by the peers themselves. We have previously described the details of such a scheme in the context of TCP connection migration [26].

2.3 Handle unexpected disconnections

The area of Internet mobility that has received the least attention is support for efficient disconnection and reconnection. While significant work has been done in the area of disconnected file systems [15, 19], less attention has been paid to preserving application communication when a disconnection occurs, enabling it to quickly resume upon reconnection. The key observation about disconnections is that they are usually unexpected. Furthermore, they last for rather unpredictable periods of time, ranging from a few seconds to several hours (or more). Today's network stacks terminate a connection as soon as a network disconnection is detected, with unfortunate consequences—the application (and often the user) has to explicitly reinitiate connectivity and application state is usually lost.

Like all other aspects of network communication, we believe the system should therefore provide standard support for unexpected disconnection, enabling applications to gracefully manage session state, releasing system resources and reallocating them when communication is restored. Even if the duration of the disconnection period is short enough to avoid significantly impacting communication or draining system resources, the disconnection and ensuing reconnection events are often hidden by current network stacks, leaving the higher network layers and application to eventually discover (often with unfortunate results) that network conditions have changed dramatically.

2.4 Provide services at the end points

A great deal of previous work in mobility management has relied on a proxy-based architecture, providing enhanced services to mobile hosts by routing communications through a (typically fixed) waypoint that is not collocated with the host [5, 10, 11, 17, 22, 28]. It is often easier to deploy new services through a proxy, as the proxy can provide enhanced services in a transparent fashion, inter-operating with legacy systems. Unfortunately, in order to provide adequate performance, it is not only necessary to highly engineer the proxy [17], but locate the proxy appropriately as well.

Several researchers have proposed techniques to migrate proxy services to the appropriate location, avoiding the need to preconfigure locations [10, 27]. Unfortunately, all candidate proxy locations must be appropriately preconfigured to participate. Further, in the face of general mobility, proxies (or at least their internal state) must be able to move with the mobile host in order to remain along the path from the host to its correspondent peers. This is a complex problem [28]; we observe that it can be completely avoided if the support is collocated with the mobile host itself.

3 Migrate approach

We now describe the Migrate approach to mobility, which leverages application naming services and informed transport protocols to provide robust, low-overhead communication between application end points. We describe a session-layer protocol that handles both changes in network attachment point and disconnection in a seamless fashion, but is flexible enough to allow a wide variety of applications to maintain sufficient control for their needs.

3.1 Service model

The number of communication paradigms in use on the Internet remains small, but the type and amount of mobility support needed varies dramatically across modalities [9]. In particular, the notion of a session is application-dependent and varies widely, from a set of related connections (e.g. FTP's data and control channels) to an individual datagram exchange such as those often found in RPC-based applications (e.g. a cached DNS response). As the lengths of sessions grow longer and they become more complex in terms of the system resources they consume, applications can benefit from system support for robust communication between application end points. However, due to the disparate performance and reliability requirements of different session-based applications, it is important that a mobility service enables the application to dictate its requirements through explicit choice of transport protocols and policy defaults.

Hence we propose an optional session layer. This layer presents a simple, unified abstraction to the application to handle mobility: a session. Sessions exist between application-level end points, and can survive changes in the transport, network, and even other session layer protocol states. It also includes basic check-pointing and resumption facilities for periods of disconnection, enabling comprehensive, session-based state management for mobile-aware applications. Unlike previous network-layer approaches, our session layer exports the specifics of the lower layers to the application, and provides an API to control them, if the application is inclined to do so.

3.2 Session layer

Applications specify their notion of a session by explicitly joining together related transport-layer connections (or destinations in connectionless protocols). Once established, a session is identified by a locally-unique token, or Session ID, and serves as the system entity for integrated accounting and management. The session layer exports a unified session abstraction to the application, managing the connections as a group, adapting to changes in network attachment point as needed. The selection of network end point and transport protocol, however, remains completely under the application's control.

To assist in the timely detection of connectivity changes, the session layer accepts notification from lower layers (e.g., loss of carrier, power loss, change of address, etc.), the application itself, or appropriately authorized external entities that may be concurrently monitoring connection state [4]. Since a session may span multiple protocols, connections, destinations, and application processes, there may be several sources of connectivity information. Regardless of the source, the session manager handles notification of disconnection and reconnection in a consistent fashion.

3.2.1 Disconnection. If a host can no longer communicate with a session end point due to mobility, as signaled by changes in the network layer state, transport layer failure, or other mechanisms, it informs the application. If the application is not prepared to handle intermittent connectivity itself, the session layer provides appropriate management services, depending on the transport layers in use, including data buffering for reliable byte streams. Specifically, it may block or buffer stream sockets, selectively drop unreliable datagrams, etc. Additional application and transport-specific services can be provided, such as disabling TCP keep-alives.

Depending on the system configuration, the session layer may need to actively attempt to reestablish communication, or it may be notified by network or transport layers when it becomes available again. System policy may dictate trying multiple network interfaces or transport protocols. In either case, if the period of disconnection becomes appropriately long (as determined by system and application configuration), it will attempt to conserve resources by reducing the state required in the network, transport, and session layers (with possibly negative performance implications upon reconnection), and notify the application, enabling additional, domain-specific resource reallocation.

3.2.2 Reconnection. Upon reattachment, a mobile host contacts each of its correspondent hosts directly, informing them of its new location. Some transport layers may be unable to adequately or appropriately handle the change in network contexts. In that case, the session layer can restart them, using the session ID to re-sync state between the end points. In either case, the session layer informs the application of reattachment, and resynchronizes the state of the corresponding session layers.

The complexity of synchronization varies with the transport protocols in use; a well-designed transport layer can handle many things by itself. By using a transport-layer token, and *not* a network layer binding, the persistent connection model can provide limited support for changes in attachment point, often with better performance than higher-layer approaches [23, 26]. Similarly, the performance of even traditional transport protocols can be enhanced when the network layer exposes the appropriate state [5, 7]. Similarly, grouping multiple transport instances between the same end points into sessions can provide additional performance improvement [4, 24].

Legacy transport protocols may be completely unable to handle changes in network addresses. In that case, the session layer may initiate an entirely new connection, and resynchronize them transparently at the session layer. In the worst case, the application itself may be unable to handle unexpected address changes, and provide no means of system notification. Such applications are still supported via IP encapsulation. The correspondent session layers establish an IP tunnel to the new end point, and continue to send application data using the old address.

If a correspondent end point is no longer reachable (possibly because the other end point also moved), the application is instructed to perform another naming/location resolution operation in attempt to locate the previous correspondent, returning a network end point (host, protocol, port) to use for communication. The particular semantics of suitable alternative end points and look-up failure are application specific. It may be a simple matter of another application-layer name resolution (perhaps a fresh DNS query), or the application may which wish to perform its own recovery in addition to or in place of reissuing the location query.

While the amount overhead varies with the capabilities of the available lower layer technologies, overhead is incurred almost exclusively during periods of disconnectivity and reconnection. This provides high performance for the common case of communication between static peers.

3.3 State management

In a spirit similar to Coda, our architecture considers disconnection to be a natural, transient occurrence that should be handled gracefully by end hosts. For extended periods of disconnection, resource allocation becomes an additional concern. While managing application state is outside the scope of our architecture, enabling efficient strategies is decidedly not. In particular, since disconnection often occurs without prior notice, applications may require system support to reclaim resources outside of their control.

There has been a great deal of study on application specific-methods of dealing with disconnected or intermittent operation. Most of it has focused on providing continued service at the disconnected client, and has not addressed the scalability of servers. If our approach becomes popular, and disconnected sessions begin to constitute a non-negligible fraction of the connections being served, servers will need to free resources dedicated to those stalled connections, and be able to easily reallocate them later. We are considering a variety of state management services the session layer should implement, and briefly hypothesize about two: migrating session state between the system and application, and providing contextual validation of session state.

3.3.1 State migration. We believe the session abstraction may be a useful way to compartmentalize small amounts of connection state, reducing the amount of state applications need to store themselves, and simplifying its management. Furthermore this state could be tagged as being associated with a particular communication session, and managed in an efficient fashion together with system state [6]. System support may allow intelligent paging or swapping of associated state out of core if the period of disconnection becomes too long.

3.3.2 Context management. There is a significant amount of context associated with a communication session, and it may be the case that some (or all) of it will be invalidated by disconnection and/or reconnection. In particular, previous work has shown that context changes in the transport layer can be leveraged to adapt application protocol state [25]. Hence any state the session layer manages needs to be revalidated, possibly internally, possibly through application-specific up-calls. Changes in context may dictate that buffers be cleared, data be reformatted, alternate transport protocols be selected, etc. This requires a coherent contextual interface between the application and the session layer.

4 Related work

The focus of the Migrate architecture is on preserving end-to-end application communication across network location changes and disconnections. Much work has been done in the area of system support for mobility over the past few years; this section outlines the work most directly related to ours.

At the network-layer, several schemes have been proposed to handle mobile routing including Mobile IP [22] and multicast-based mobility [20]. Mobile IP uses a home agent as to intercept and forward packets, with a route optimization option to avoid triangle routing. The home-agent-based approach has also been applied at the transport layer, as in MSOCKS [17], where connection redirection was achieved using a split-connection proxy, providing so-called transport-layer mobility. Name resolution and message routing were integrated to implement a "late binding" option that tracks highly mobile services and nodes in the Intentional Naming System [3].

Most TCP-specific solutions for preserving communication across network-layer changes [23, 26] do not handle the problems associated with connections resuming after substantial periods of disconnectivity. A "persistent connection" scheme where the communication end-points are location independent was proposed for TCP sockets and DCE RPC [29], but the mapping between global endpoint names and current physical endpoints is done through a global clearinghouse, which notifies everyone of binding updates. Session layer mobility [16] explored moving entire sessions by utilizing a global naming service to provide endpoint bindings; address changes are affected through a TCP-specific protocol extension.

5 Conclusion

In this paper, we have defined five salient issues concerning host mobility in the Internet. We presented a set of design guidelines for building a system to address these issues, distilled from a decade of research in mobile applications and system support for mobility on the Internet. Following these principles, we outlined *Migrate*, a basic session-based architecture to preserve end-to-end application-layer communication in the face of mobility of the end points. We believe the general abstractions for disconnection, hibernation, and reconnection provided by the session layer define an appropriate set of interfaces to enable more advanced system support for mobility.

References

1. September 1997.
2. March 2001.
3. William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *Proc. ACM SOSP*, pages 186–201, December 1999.
4. Hari Balakrishnan, Hariharan Rahul, and Srinivasan Seshan. An integrated congestion management architecture for Internet hosts. In *Proc. ACM SIGCOMM*, pages 175–187, August 1999.
5. Hari Balakrishnan, Srinivasan Seshan, and Randy H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 1(4):469–481, December 1995.
6. Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proc. USENIX OSDI*, pages 45–58, February 1999.
7. Ramon Caceres and Liviu Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE JSAC*, 13(5):850–857, June 1995.
8. Vinton G. Cerf and Edward Cain. The DoD Internet architecture model. *Computer Networks*, 7:307–318, October 1983.
9. Stuart Cheshire and Mary Baker. Internet mobility 4x4. In *Proc. ACM SIGCOMM*, pages 318–329, August 1996.
10. Mike Dahlin, Bharat Chandra, Lei Gao, Amjad-Ali Khoja, Amol Nayate, Asim Razzaq, and Anil Sewani. Using mobile extensions to support disconnected services. Technical report, UT Austin, April 2000.
11. Mark Gritter and David Cheriton. An architecture for content routing support in the internet. In *Proc. 3rd USITS* [2], pages 37–48.
12. E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Ver. 2. RFC 2608, June 1999.
13. Jon Inouye, Jim Binkley, and Jonathan Walpole. Dynamic network reconfiguration support for mobile computers. In *Proc. ACM/IEEE Mobicom* [1], pages 13–22.
14. Anthony D. Joseph, Joshua A. Tauber, and M. Frans Kaashoek. Mobile computing with the rover toolkit. *IEEE Trans. on Computers*, 46(3):337–352, March 1997.

15. James Kistler and M. Satyanarayanan. Disconnected operation in the Coda filesystem. In *Proc. ACM SOSP*, pages 213–225, October 1991.
16. Bjorn Landfeldt, Tomas Larsson, Yuri Ismailov, and Aruna Seneviratne. SLM, a framework for session layer mobility management. In *Proc. IEEE ICCCN*, pages 452–456, October 1999.
17. David Maltz and Pravin Bhagwat. MSOCKS: An architecture for transport layer mobility. In *Proc. IEEE Infocom*, pages 1037–1045, March 1998.
18. Dejan Milojicic, Frederick Douglis, and Richard Wheeler. *Mobility: Processes, Computers, and Agents.* Addison Wesley, Reading, Massachusetts, 1999.
19. Lily Mummert, Maria Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proc. ACM SOSP*, pages 143–155, December 1995.
20. Jayanth Mysore and Vaduvur Bharghavan. A new multicasting-based architecture for internet host mobility. In *Proc. ACM/IEEE Mobicom* [1], pages 161–172.
21. Brian Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Proc. ACM SOSP*, pages 276–287, October 1997.
22. Charles E. Perkins. IP mobility support. RFC 2002, October 1996.
23. Xun Qu, Jeffrey Xu Yu, and Richard P. Brent. A mobile TCP socket. In *Proc. IASTED Intl. Conf. on Software Eng.*, November 1997.
24. S. Savage, N. Cardwell, and T. Anderson. The Case for Informed Transport Protocols. In *Proc. HotOS VII*, pages 58–63, March 1999.
25. Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan. Fine-grained failover using connection migration. In *Proc. 3rd USITS* [2], pages 221–232.
26. Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Proc. ACM/IEEE Mobicom*, pages 155–166, August 2000.
27. Amin Vahdat and Michael Dahlin. Active names: Flexible location and transport of wide-area resources. In *Proc. 2nd USITS*, October 1999.
28. Bruce Zenel and Dan Duchamp. A general purpose proxy filtering mechansism applied to the mobile environment. In *Proc. ACM/IEEE Mobicom* [1], pages 248–259.
29. Yongguang Zhang and Son Dao. A "persistent connection" model for mobile and distributed systems. In *Proc. IEEE ICCCN*, pages 300–307, September 1995.
30. Xinhua Zhao, Claude Castelluccia, and Mary Baker. Flexible network support for mobile hosts. *ACM MONET*, 6(1), 2001.

Proc. of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001), Rome, Italy, July 2001.

Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks

Benjie Chen, Kyle Jamieson, Hari Balakrishnan and Robert Morris*

MIT Laboratory for Computer Science
{`benjie, jamieson, hari, rtm`}`@lcs.mit.edu`

Abstract. This paper presents *Span*, a power saving technique for multi-hop ad hoc wireless networks that reduces energy consumption without significantly diminishing the capacity or connectivity of the network. Span builds on the observation that when a region of a shared-channel wireless network has a sufficient density of nodes, only a small number of them need be on at any time to forward traffic for active connections. Span is a distributed, randomized algorithm where nodes make local decisions on whether to sleep, or to join a *forwarding backbone* as a *coordinator*. Each node bases its decision on an estimate of how many of its neighbors will benefit from it being awake, and the amount of energy available to it. We give a randomized algorithm where coordinators rotate with time, demonstrating how localized node decisions lead to a connected, capacity-preserving global topology.

Improvement in system lifetime due to Span increases as the ratio of idle-to-sleep energy consumption increases. Our simulations show that with a practical energy model, system lifetime of an 802.11 network in power saving mode with Span is a factor of two better than without. Additionally, Span also improves communication latency and capacity.

1 Introduction

Minimizing energy consumption is an important challenge in mobile networking. Enough progress has been made on low-power hardware

---

* This research was funded in part by NTT Corporation under the NTT-MIT collaboration and by Intel Corporation.

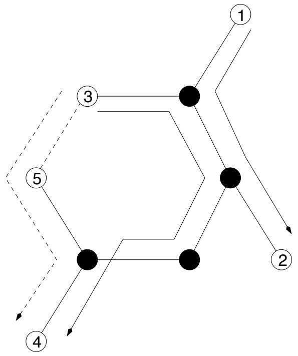


Fig. 1: A connected backbone does not necessarily preserve capacity. In this connected topology, black nodes are coordinators. Nodes that are within radio range of each other are connected by solid or dotted lines. Solid lines represent connections to and between coordinators. Packets between nodes 3 and 4 may contend for bandwidth with packets between nodes 1 and 2. On the other hand, if node 5 was a coordinator, no contention would occur.

design for mobile devices that the wireless network interface is often a device's single largest consumer of power. Since the network interface may often be idle, this power could be saved by turning the radio off when not in use. In practice, however, this approach is not straightforward: a node must arrange to turn its radio on not just to send packets, but also to receive packets addressed to it and to participate in any higher-level routing and control protocols. The requirement of cooperation between power saving and routing protocols is particularly acute in the case of multi-hop ad hoc wireless networks, where nodes must forward packets for each other. Coordination of power saving with routing in ad hoc wireless networks is the subject of this paper.

A good power-saving coordination technique for wireless ad-hoc networks ought to have the following characteristics. It should allow as many nodes as possible to turn their radio receivers off most of the time, since even an idle receive circuit can consume almost as much energy as an active transmitter. On the other hand, it should forward packets between any source and destination with minimally more delay than if all nodes were awake. This implies that enough nodes must stay awake to form a connected backbone. The algorithm for picking this backbone should be distributed, requiring each node to make a local decision. Furthermore, the backbone formed by the awake nodes should provide about as much total capacity as the original network, since otherwise congestion may increase. This means that paths that could operate without interference in the original network should be represented in the backbone. For example, Figure 1 illustrates a topology that violates this principle. In this topology, black nodes are coordinators. Nodes that are within radio range of each other are connected by solid or dotted lines. Packets between nodes 3 and 4 may contend for bandwidth with packets between nodes 1 and 2 (solid arrows). On the other hand, if node 5 was a coordinator, node 3 can send packets to node 4 via the path shown by the dotted arrow, and no contention would occur.

A good coordination technique should not make many assumptions about the link layer's facilities for sleeping; it should work with any link-layer that provides for sleeping and periodic polling, including 802.11's ad-hoc power saving mode. Finally, power saving should inter-operate correctly with whatever routing system the ad-hoc network uses.

The algorithm presented in this paper, *Span,* fulfills the above requirements. Each node in the network running Span makes periodic, local decisions on whether to sleep or stay awake as a *coordinator* and participate in the forwarding backbone topology. To preserve capacity, a node volunteers to be a coordinator if it discovers, using information it gathered from local broadcast messages, that two of its neighbors cannot communicate with each other directly or through one or two existing coordinators. To keep the number of redundant coordinators low and rotate this role amongst all nodes, each node delays announcing its willingness by a random time interval that takes two factors into account: the amount of remaining battery energy, and the number of pairs of neighbors it can connect together. This combination ensures, with high probability, a capacity-preserving connected backbone at any point in time, where nodes tend to consume energy at about the same rate. Span does all this using only local information, and consequently scales well with the number of nodes. Our simulation results, with energy parameters from measurements of today's 802.11 wireless interfaces, show that system lifetime with Span is more than a factor of two better than without Span, for a range of node densities, without much reduction in overall forwarding capacity.

The rest of the paper describes and evaluates Span. Section 2 reviews related work. Section 3 describes Span's algorithms and its interactions with the link layer. Section 4 presents our implementation of Span on top of an IEEE 802.11 link layer in the ns-2 network simulator. Section 5 presents performance results of several experiments. Finally, Section 6 concludes.

2 Related work

The set of coordinators elected by Span at any time is a connected dominating set of the graph formed by the nodes of the ad hoc network. A connected dominating set S of a graph G is a connected subgraph of G such that every vertex u in G is either in S or adjacent to some v in S. Routing using connected dominating sets of a graph can reduce the search space for routes [6, 27].

Das and Bharghavan [6] approximate the minimum connected dominating set of an ad hoc network, and route packets using nodes from that set. The set of coordinators elected by Span, however, has the additional property of being capacity preserving. Consequently, the connected dominating set elected by Span is likely to be larger than a minimal connected dominating set. For example, the black nodes in Figure 1 form a minimal connected dominating set. However, Span's election algorithm would additionally elect node 5 to be a coordinator to preserve capacity.

Wu and Li [27] propose a distributed algorithm for approximating connected dominating sets in an ad hoc network that also appears to preserve capacity. In a later paper, Wu an Gao [26] discuss power aware routing using the connected dominating sets. Their algorithm is similar to Span's coordinator election algorithm. Span, however, elects fewer coordinators because it actively prevents redundant coordinators by using randomized slotting and damping.

The recent GAF [29] scheme of Xu *et al.* has similar goals to Span. In GAF, nodes use geographic location information to divide the world into fixed square grids. The size of each grid stays constant, regardless of node density. Nodes within a grid switch between sleeping and listening, with the guarantee that one node in each grid stays up to route packets. Span differs from GAF in two important ways. First, unlike GAF, Span does not require that nodes know their geographic positions. Instead, Span uses local broadcast messages to discover and react to changes in the network topology. Second, Span integrates with 802.11 power saving mode nicely: non-coordinator nodes can still receive packets when operating in power saving mode.

In AFECA [28], each node maintains a count of the number of nodes within radio range, obtained by listening to transmissions on the channel. A node switches between sleeping and listening, with randomized sleep times proportional to the number of nearby nodes. The net effect is that the number of listening nodes is roughly constant, regardless of node density; as the density increases, more energy can be saved. AFECA's constants are chosen so that there is a high probability that the listening nodes form a connected graph, so that ad hoc forwarding works. An AFECA node does not know whether it is required to listen in order to maintain connectivity, so to be conservative AFECA tends to make nodes listen even when they could be asleep. Span differs from AFECA in that, with high likelihood, Span never keeps a node awake unless it is absolutely essential for connecting two of its neighbors. Furthermore, Span explicitly attempts to preserve the same overall system capacity as the underlying network where all nodes are awake, which ensures that no increase in congestion occurs.

The PAMAS power-saving medium access protocol [18, 23] turns off a node's radio when it is overhearing a packet not addressed to it. This approach is suitable for radios in which processing a received packet is expensive compared to listening to an idle radio channel. Kravets and Krishnan [13] present a system in which mobile units wake up periodically and poll a base station for newly arrived packets. Like Stemm and Katz [24], they show that setting the on/off periods based on application hints reduces both power and delay. Span assumes the presence of an ad hoc polling mechanism such as that provided by 802.11, and could potentially work in concert with application hints; such hints would apply only to sleeping nodes, not coordinators. Smith *et al.* [15] propose an ad hoc network that elects a virtual base station to buffer packets for local nodes. They do not, however, attempt to make sure that enough of these base stations are present to preserve connectivity in a multi-hop ad hoc network.

Minimum-energy routing [22] saves power by choosing paths through a multi-hop ad hoc network that minimize the total transmit energy. This approach has been extended by Chang and Tassiulas [4] to maximize overall network lifetime by distributing energy consumption fairly. In this protocol, nodes adjust their transmission power levels and select routes to optimize performance. Ramanathan and Rosales-Hain describe distributed algorithms that vary transmission power and attempt to maintain connectedness [19]. Rodoplu and Meng give a distributed algorithm to produce minimum-power routes by varying node transmission power [20]. Wattenhofer *et al.* [25] describe a topology maintenance algorithm using similar underlying radio support, but their algorithm guarantees global connectedness using directional information. Span controls whether or not the receiver is powered on, rather than controlling the transmit power level. It also pays close attention to overall system capacity, in addition to maintaining connectivity.

An alternative approach is described by Heinzelman *et al.*, whose LEACH protocol selects rotating cluster-heads to collect local information and transmit it to a base station in a wireless sensor network [10].

Like LEACH, Intanagonwiwat *et al.*'s directed diffusion mechanism [11] takes advantage of aspects of sensor networks, particularly the possibility of aggregating and compressing data, that are not present in general-purpose networks.

In general, the basic idea that a path with many short hops is sometimes more energy-efficient than one with a few long hops could be applied to any ad hoc network with variable-power radios and knowledge of positions. This technique and Span's are orthogonal, so their benefits could potentially be combined.

3 Span design

Span adaptively elects "coordinators" from all nodes in the network. Span coordinators stay awake continuously and perform multi-hop packet routing within the ad hoc network, while other nodes remain in power-saving mode and periodically check if they should wake up and become a coordinator.

Span achieves four goals. First, it ensures that enough coordinators are elected so that every node is in radio range of at least one coordinator. Second, it rotates the coordinators in order to ensure that all nodes share the task of providing global connectivity roughly equally. Third, it attempts to minimize the number of nodes elected as coordinators, thereby increasing network lifetime, but without suffering a significant loss of capacity or an increase in latency. Fourth, it elects coordinators using only local information in a decentralized manner—each node only consults state stored in local routing tables during the election process.

Span is proactive: each node periodically broadcasts `HELLO` messages that contain the node's status (i.e., whether or not the node is a coordinator), its current coordinators, and its current neighbors. From these `HELLO` messages, each node constructs a list of the node's neighbors and coordinators, and for each neighbor, a list of its neighbors and coordinators.

As shown in Figure 3, Span runs above the link and MAC layers and interacts with the routing protocol. This structuring allows Span to take advantage of power-saving features of the link layer protocol, while still being able to affect the routing process. For example, non-coordinator nodes can periodically turn on their radios and listen (as in the 802.11 power-saving mode [1]) or poll (as in LPMAC [15]) for their packets. Span leverages a feature of modern power-saving MAC layers, in which if a node has been asleep for a while, packets destined for it are not lost but are buffered at a neighbor. When the node awakens, it can retrieve these packets from the buffering node, typically a coordinator.

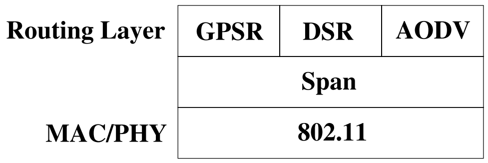


Fig. 2: Span is a protocol that operates under the routing layer and above the MAC and physical layers. The routing layer uses information Span provides, and Span takes advantage of any power saving features of the underlying MAC layer.

Span also requires a modification to the route lookup process at each node—at any time, only those entries in a node's routing table that correspond to currently active coordinators can be used as valid next-hops (unless the next hop is the destination itself).

A Span node switches state from time to time between being a coordinator and being a non-coordinator. A node includes its current state in its `HELLO` messages. The following sections describe how a node decides that it should announce that it is a coordinator, and how it decides that it should withdraw from being a coordinator.

3.1 Coordinator announcement

Periodically, a non-coordinator node determines if it should become a coordinator or not. The following *coordinator eligibility rule* in Span ensures that the entire network is covered with enough coordinators:

> **Coordinator eligibility rule:** A non-coordinator node should become a coordinator if it discovers, using only information gathered from local broadcast messages, that two of its neighbors cannot reach each other either directly or via one or two coordinators.

This election algorithm does not yield the minimum number of coordinators required to merely maintain connectedness. However, it roughly ensures that every populated radio range in the entire network contains at least one coordinator. Because packets are routed through coordinators, the resulting coordinator topology should yield good capacity.

Announcement contention occurs when multiple nodes discover the lack of a coordinator at the same time, and all decide to become a coordinator. Span resolves contention by delaying coordinator announcements with a randomized backoff delay. Each node chooses a delay value, and delays the `HELLO` message that announces the node's volunteering as a coordinator for that amount of time. At the end of the delay, the node reevaluates its eligibility based on `HELLO` messages recently received, and makes its announcement if and only if the eligibility rule still holds.

We consider a variety of factors in our derivation of the backoff delay. Consider first the case when all nodes have roughly equal energy, which implies that only topology should play a role in deciding which nodes become coordinators. Let N_i be the number of neighbors for node i and let C_i be the number of additional pairs of nodes among these neighbors that would be connected if i were to become a coordinator and forward packets. Clearly, $0 \leq C_i \leq \binom{N_i}{2}$. We call $\frac{C_i}{\binom{N_i}{2}}$ the *utility* of node i. If nodes with high C_i become coordinators, fewer coordinators in total may be needed in order to make sure every node can talk to a coordinator; thus a node with a high C_i should volunteer more quickly than one with smaller C_i.

If there are multiple nodes within radio range that all have the same utility, Span prevents too many of them becoming coordinators. This is because such coordinators would be redundant—they would not increase system capacity, but simply drain energy. If the potential coordinators make their decisions simultaneously, they may all decide to become coordinators. If, on the other hand, they decide one at a time, only the first few will become coordinators, and the rest will notice that there are already enough coordinators and go back to sleep. To handle this, we use a randomized "slotting-and-damping" method reminiscent of techniques to avoid multiple retransmissions of lost packets by multicast protocols, such as XTP [5], IGMP [8] and SRM [9]: the delay for each node is randomly chosen over an interval proportional to $N_i \times T$, where T is the round-trip delay for a small packet over the wireless link.

Thus, when all nodes have roughly equal energy, the above discussion suggests a backoff delay of the form:

$$delay = \left(\left(1 - \frac{C_i}{\binom{N_i}{2}}\right) + R\right) \times N_i \times T \qquad (1)$$

The randomization is achieved by picking R uniformly at random from the interval $(0, 1]$.

Consider the case when nodes may have unequal energy left in their batteries. We observe that what matters in a heterogeneous network is not necessarily the absolute amount of energy available at the node, but the amount of energy scaled to the maximum amount of energy that the node can have. Let E_r denote the amount of energy (in Joules) at a node that still remains, and E_m be the maximum amount of energy available at the same node. A reasonable (but not the only) notion of fairness can be achieved by ensuring that a node with a larger value of E_r/E_m is more likely to volunteer to become a coordinator more quickly than one with a smaller ratio. Thus, we need to add a decreasing function of E_r/E_m that reflects this, to Equation 1. There are an infinite number of such functions, from which we choose a simple linear one: $1 - E_r/E_m$. In addition to its simplicity, this choice is attractive because it ensures that the rate with which a node reduces its propensity to advertise (as a function of the amount of energy it has left), is constant. (We experimented with a few other functions, including an exponentially decaying function of E_r/E_m and an inversely decaying function of E_r/E_m; the simple linear one worked best.)

Combining this with Equation 1 yields the following equation for the backoff delay in Span:

$$delay = \left(\left(1 - \frac{E_r}{E_m}\right) + \left(1 - \frac{C_i}{\binom{N_i}{2}}\right) + R\right) \times N_i \times T \tag{2}$$

Observe that the first term does not have a random component; thus if a node is running low on energy, its propensity to become a volunteer is guaranteed to diminish relative to other nodes in the neighborhood with similar neighbors.

In a network with uniform density and energy, our election algorithm rotates coordinators among all nodes of the network. It achieves fairness because the likelihood of becoming a coordinator falls as a coordinator uses up its battery. In practice, however, ad hoc networks are rarely uniform. Our announcement rule adapts to non-uniform topology: a node that connects network partitions together will always be elected a coordinator. This property preserves capacity over the lifetime of the network. Because of Span's emphasis on capacity-preservation to the extent possible, such critical nodes will unavoidably die before other less-critical ones. However, in a mobile Span network, a given node is rarely stuck in such a position, and this improves fairness dramatically.

3.2 Coordinator withdrawal

Each coordinator periodically checks if it should withdraw as a coordinator. A node should withdraw if every pair of its neighbors can reach each other either directly or via one or two other coordinators.

In order to also rotate the coordinators among all nodes fairly, after a node has been a coordinator for some period of time, it marks itself as a tentative coordinator if every pair of neighbor nodes can reach each other via one or two other neighbors, even if those neighbors are not currently coordinators. A tentative coordinator can still be used to forward packets. However, the coordinator announcement algorithm described above treats a tentative coordinator as a non-coordinator. Thus, by marking itself as tentative, a coordinator gives its neighbors a chance to become coordinators. A coordinator stays tentative for W_T amount of time, where W_T is the maximum value of Equation 2. That is,

$$W_T = 3 \times N_i \times T \tag{3}$$

If a coordinator has not withdrawn after W_T, it clears its tentative bit. To prevent an unlucky low energy node from draining all of its energy once it becomes a coordinator, the amount of time a node stays as a coordinator before turning on its tentative bit is proportional to the amount of energy it has (E_r/E_m).

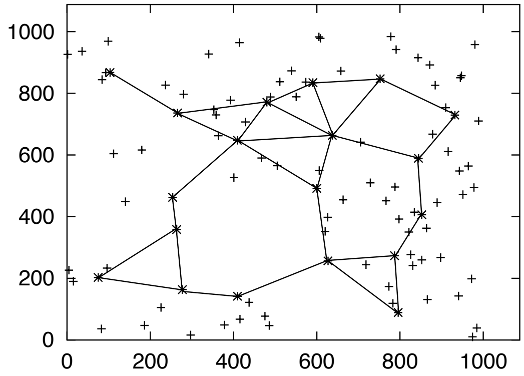


Fig. 3: A scenario with 100 nodes, 19 coordinators, and a radio range of 250 meters. The nodes marked "$*$" are coordinators; the nodes marked "$+$" are non-coordinator nodes. Solid lines connect coordinators that are within radio range of each other.

While Span uses local `HELLO` messages to propagate topology information, it does not depend on them for correctness: when `HELLO` messages are lost, Span elects more coordinators, but does not disconnect the backbone. Figure 3 shows the result of our election algorithm at a random point in time on a network of 100 nodes in a 1000 meter × 1000 meter area, where each radio has an isotropic circular range with a 250 meter radius. Solid lines connect coordinators that are within radio range of each other.

4 Simulator implementation

This section describes our implementation of Span, geographic forwarding, the 802.11 power saving mode (with our own improvements), and the energy model we used in our simulations. We ran our Span implementation in the `ns-2` network simulator environment.

4.1 Span and geographic forwarding

Our implementation uses a geographic forwarding algorithm. We chose to implement geographic forwarding primarily because of its simplicity; Span can be used with other routing protocols as well.

Span's election algorithm requires each node to advertise its coordinators, its neighbors, and if it is a coordinator, a tentative coordinator, or a non-coordinator. To reduce protocol overhead, we piggyback Span `HELLO` information (see Section 3.1) onto the broadcast updates required by geographic forwarding. See Table 1. Each node enters all the information it receives in broadcast updates into a *neighbor table*. Consequently, this neighbor table contains a list of neighbors and coordinators, and for each neighbor, a list of its neighbors and coordinators.

Source ID
Node position
Is coordinator
Is tentative
Coordinator list
Neighbor List

Table 1: `HELLO` packet for Span and geographic forwarding. Italized fields are Span specific information.

Geographic forwarding forwards packets using a greedy algorithm. The source node annotates each packet with the geographic location of the destination node. Upon receiving a packet for a node not in radio range, a coordinator forwards the packet to a neighboring coordinator that is closest to the destination. If no such coordinator exists, the packet is forwarded to a non-coordinator that is closer to the destination. Otherwise, we know that a packet has encountered a void, and so it is dropped. (We did not implement an idea like GPSR [12], which ameliorates the effects of voids.)

Our simulations do not use a location service. Instead, each sender uses the `GOD` module of `ns` to obtain the geographic location of the destination node. Hence, our simulation results may be better than one might expect with a real location service, such as GLS [14].

Our geographic forwarding algorithm also implements MAC-layer failure feedback and interface queue traversal [2, 12]. These mechanisms allow the routing layer to readily remove unresponsive nodes from its routing table and rescue packets using these nodes as the next hop.

4.2 Coordinator election

A node uses information from its neighbor table to determine if it should announce or withdraw itself as a coordinator. Figure 4 shows the coordinator announcement algorithm. A non-coordinator node periodically calls `check-announce-coordinator` to determine if it should become a coordinator or not. `check-announce-coordinator` first computes C, the number of additional neighbor pairs that would be connected if the node becomes a coordinator, using `connect-pair`. If $C > 0$, the node computes *delay* using Equation 2 and waits for *delay* seconds before recomputing C. If C continues to be greater than 0 after *delay* seconds, the node announces itself as a coordinator. `connect-pair` calculates the number of would-be connected neighbor pairs by iterating through the node's neighbors in the neighbor table. A similar routine exists for checking if every pair of neighbor nodes can reach each other via one or two other neighbors. That routine is used by the withdraw algorithm.

In addition to the coordinator election algorithm shown in Figure 4, we implemented a special case for electing coordinators. The geographic routing algorithm can readily detect that a coordinator has left the region through MAC layer failure feedback. However, the Span election algorithm may not react fast enough to elect new coordinators. In the worst case, nodes must wait until the old coordinator information has expired in the neighbor table before a new coordinator can be elected. Because geographic forwarding falls back to using non-coordinators to route packets if coordinators do not exist, a non-coordinator node an-

```
// a non-coordinator node periodically calls this routine to see if it should become a coordinator
check-announce-coordinator()
    $C$ = connect-pairs()
    if $C > 0$ {
        calculate $delay$ using Equation 2, using $C$ as $C_i$
        wait $delay$
        if connect-pairs() $> 0$ {
            announce itself as a coordinator
        }
    }

// returns number of neighbor pairs a node can connect if it becomes a coordinator
connect-pairs()
    $n = 0$
    for each neighbor $a$ in neighbor table {
        for each neighbor $b$, $b > a$, in neighbor table {
            if share-other-coordinators($a$, $b$) == false {
                $n \leftarrow n + 1$
            }
        }
    }
    return $n$

// returns true if neighbors a and b are connected by one or two other coordinators
share-other-coordinators($a$, $b$)
    // coordinator lists are kept in the neighbor table
    for each coordinator $c\_a$ in $a$'s coordinator list {
        if $c\_a$ equals self {
            continue
        }
        else if $c\_a$ in $b$'s coordinator list {
            return true
        }
        // try to see if we know a path from $a$ to $b$ via two coordinators
        else if $c\_a$ in neighbor table {
            for each coordinator $c\_c\_a$ in $c\_a$'s coordinator list {
                if $c\_c\_a$ equals self {
                    continue
                }
                else if $c\_c\_a$ in $b$'s coordinator list {
                    return true
                }
            }
        }
    }
    return false
```

Fig. 4: Coordinator announcement algorithm.

nounces itself as a coordinator if it has received a large number of packets to route in the recent past. If this coordinator turns out to be redundant, the coordinator withdraw algorithm will force the node to withdraw itself as a coordinator soon after.

4.3 802.11 Ad Hoc power-saving mode

Span determines when to turn a node's radio on or off, but depends on the low level MAC layer to support power saving functions, such as buffering packets for sleeping nodes. We have implemented Span on top of the 802.11 MAC and physical layers with ad hoc power saving support [1].

802.11 ad hoc power-saving mode uses periodic *beacons* to synchronize nodes in the network. Beacon packets contain timestamps that synchronize nodes' clocks. A beacon period starts with an ad hoc traffic indication message window (*ATIM window*), during which all nodes are listening, and pending traffic transmissions are advertised. A node that receives and acknowledges an advertisement for unicast or broadcast traffic directed to itself must stay on for the rest of the beacon period. Otherwise, it can turn itself off at the end of the ATIM window, until the beginning of the next beacon period. After the ATIM window, advertised traffic is transmitted. Since traffic cannot be transmitted during the ATIM window, the available channel capacity is reduced.

When the 802.11 MAC layer is asked to send a packet, it may or may not be able to send it immediately, depending on which ATIM's have been sent and acknowledged in the immediately preceding or current, ATIM window. If the packet arrives at the MAC during the ATIM window, or if the advertisement for the packet has not been acknowledged, it needs to be buffered. In our implementation, we buffer packets for two beacon periods. Packets that have not been transmitted after two beacon periods are dropped.

The beacon period and ATIM window size greatly affect routing performance [21]. While using a small ATIM window may improve energy savings, there may not be enough time for all buffered packets to be advertised. Using an ATIM window that is too large not only decreases available channel utilization, it may also not leave enough room between the end of the ATIM window and the beginning of the next beacon period to transmit all advertised traffic. We have experimentally determined that a beacon period of 200 ms and an ATIM window size of 40 ms result in good throughput and low loss rate.

Aside from decreased channel capacity, 802.11 power saving mode (without Span) also suffers from long packet delivery latency: for each hop that a packet traverses, the packet is expected to be delayed for half a beacon period.

4.4 Improving 802.11 using Span

Using Span on top of 802.11 ad hoc power saving mode can improve routing throughput and packet delivery latency. Because coordinators do not operate in power saving mode, packets routed between coordinators do not need to be advertised or delayed. To further take advantage of the synergy between Span and 802.11 power saving mode, we have made the following modifications to our simulation of 802.11 power saving mode.

- **No advertisements for packets between coordinators.** Packets routed between coordinators are marked by Span. While the MAC layer still needs to buffer these packets if they arrive during the ATIM window, it does not send traffic advertisements for them. To ensure that Span does not provide incorrect information due to topology changes, the MAC maintains a separate neighbor table. The MAC layer uses a bit in the MAC header of each packet it sends to notify neighbors of its power saving status. Since the MAC layer can sniff the header of every packet, including RTS packets, this neighbor table is likely to be correct. When a node withdraws as a coordinator, advertisements for traffic to that node will be sent during the next ATIM window. This optimization allows the ATIM window to be reduced without hurting throughput.
- **Individually advertise each broadcast message.** With unmodified 802.11 power saving mode, a node only needs to send one broadcast advertisement even if it has more than one broadcast message to send. This is because once a node hears an advertisement for a broadcast message, it stays up for the entire duration of the beacon period. Since most traffic to non-coordinator nodes in our network would be broadcast messages sent by Span and the geographic routing protocol, we modified the MAC so each broadcast message must be explicitly advertised. For example, if a node receives 5 broadcast advertisements, no unicast advertisements, and then 5 broadcast messages after the ATIM window, it can safely turn itself off.
- **New advertised traffic window.** With unmodified 802.11 power saving mode, if a node receives a unicast advertisement, it must remain on for the rest of the beacon period. In a Span network, packets routed via non-coordinator nodes are rare. To take advantage of this, we introduced a new *advertised traffic window* in the MAC. The advertised traffic window is smaller than the beacon period. It starts at the beginning of the beacon period, and extends beyond the end of the ATIM window. Outside the ATIM window but inside the advertised traffic window, advertised packets and packets to coordinators can be transmitted. Outside the advertised traffic window, however, only packets between coordinators can be transmitted. This allows a node in power saving mode to turn itself off at the end of the advertised traffic window until the next beacon period.

These three modifications allow each node to use a long beacon period and a short ATIM window. The short ATIM window improves channel utilization, while the long beacon period increases the fraction of time a non-coordinator node can remain asleep. In our simulations, we used a beacon period of 300 ms, an ATIM window of 20 ms, and an advertised traffic window of 100 ms. We set the propagation delay T in Equation 2 to be the length of a beacon period.

Span does not require these modifications, but does better when they are implemented. Section 5 compares performance of Span with the modified 802.11 power saving mode, unmodified 802.11 in ad hoc power saving mode, and unmodified 802.11 without power saving mode.

4.5 Energy model

To accurately model energy consumption, we took measurements of the Cabletron Roamabout 802.11 DS High Rate network interface card (NIC) operating at 2 Mbps in base station mode. To measure power consumed by the card, we powered a portable computer solely with its AC adapter (without the battery), and measured the voltage across a resistor placed in series with the card on the computer to obtain the instantaneous current through the NIC. The voltage across the NIC remained constant at all times, thus from the instantaneous current measurement, we calculated the instantaneous power consumed by the card. We summarize the time-averaged results in Table 2, and note that these closely match the results obtained by Feeney and Nilsson [7] for similar 802.11 network interface cards in the ad hoc mode.

We obtained the “Rx” state measurement by putting the card into non-power saving mode, and measuring the power required to listen for a packet, decode it, and pass its contents up to the host. The “idle” state measurement was obtained in the same manner, but measuring only the power required to listen for a packet. In contrast, the “sleep” state measurement was obtained by putting the card into power saving mode, and measuring the average (lower, and near-constant) power consumption during the part of the power saving cycle where the card was not listening for packets. The key point to note is the large difference between the power consumption of the idle and sleeping modes. This suggests that putting the non-coordinator nodes that do not have data to transmit in sleeping mode can be beneficial.

5 Performance evaluation

To measure the effectiveness of Span, we simulated Span, with geographic forwarding, on several static and mobile topologies. Simulation

Tx	Rx	Idle	Sleeping
1400 mW	1000 mW	830mW	130mW

Table 2: Power consumption of the Cabletron 802.11 network card in the Tx (transmit), Rx (receive), Idle, and Sleeping modes.

results show that Span not only performs well by extending network lifetime, it out-performs unmodified 802.11 power saving network in handling heavy load, per-packet delivery latency, and network lifetime.

5.1 Simulation environment

We simulated Span in the `ns-2` [17] network simulator using the CMU wireless extensions [16]. The geographic forwarding algorithm, as described in Section 4.1, routes packets from source to destination. Span runs on top of the 802.11 MAC layer with power saving support and modifications described in Section 4.3. In this section, we compare performance of Span against both unmodified 802.11 MAC in power saving mode and unmodified 802.11 MAC not in power saving mode. For convenience, we will refer to them as Span, 802.11 PSM, and 802.11.

To evaluate Span in different node densities, we simulate 120-node networks in square regions of different sizes. Nodes in our simulations use radios with a 2 Mbps bandwidth and 250 meters nominal radio range. Twenty nodes send and receive traffic. Each of these nodes send a CBR flow to another node, and each CBR flow sends 128 byte packets. In Section 5.2 we vary the rate of the CBR traffic to measure performance of Span under different traffic load. In other experiments, each sender sends three packets per second, for a total of 60 Kbps of traffic.

To ensure that the packets of each CBR flow go through multiple hops before reaching the destination node, 10 source and destination nodes are placed, uniformly at random, on each of two 50 meter-wide, full-height strips located at the left and right of the simulated region. A source must send packets to a destination node on the other strip. The initial positions of the remaining 100 nodes are chosen uniformly at random in the entire simulated region. Thus, the square root of the area of the simulated region and the number of hops needed by each packet are approximately proportional.

Source and destination nodes never move. They stay awake at all times so they can send and receive packets at higher throughputs. However, they do not participate in coordinator elections. Thus, only

100 nodes can become coordinators. In mobile experiments, the motion of the remaining 100 nodes follows the *random waypoint* model [3]: initially, each node chooses a destination uniformly at random in the simulated region, chooses a speed uniformly at random between 0 and 20 m/s, and moves there with the chosen speed. The node then pauses for an adjustable period of time before repeating the same process. The degree of mobility is reflected in the pause time. By default, we used a pause time of 60 seconds.

For simplicity, we did not use a location service in our simulations. Instead, a router obtains the location of the destination node from the `GOD` module in `ns`. Since the location lookup is only required once per flow at the sender, we believe the overhead produced by the location service is not likely to change our results. Nevertheless, location services such as GLS [14] can be used with Span.

All experimental results in this section are averages of five runs on different randomly-chosen scenarios. We define *node density* (as used in our graph axis labels) as the number of nodes that are not sources or destinations per radio range, an area of $250^2 \times \pi$ square meters.

		Span			802.11 PSM			802.11		
Area	Density	Loss	Lat (ms)	Hops	Loss	Lat (ms)	Hops	Loss	Lat (ms)	Hops
500m×500m	78.5	0.0%	23.4	2.8	0.0%	423	2.4	0.0%	5.69	2.4
750m×750m	34.9	0.0%	30.7	4.5	0.0%	739	4.0	0.0%	11.2	4.0
1000m×1000m	19.6	0.4%	40.5	6.1	0.1%	1032	5.4	0.0%	16.9	5.4
1250m×1250m	12.6	1.9%	45.2	7.8	10.7%	1391	7.3	7.0%	20.6	7.3

Table 3: Performance of geographic forwarding with Span, 802.11 PSM, and 802.11 as node density and area of simulation region changes. Span delivers packets using slightly more hops. Span's packet delivery latency is higher than 802.11's, but is significantly less than that of 802.11 PSM.

5.2 Capacity Preservation

One of Span's goals is to preserve total network capacity, by making sure that if there are non-conflicting paths in the underlying network, there are similar non-conflicting paths in the coordinator backbone. This section compares the capacity available in a Span network with the capacity in an ordinary 802.11 network. We measure capacity by the number of packets the network can successfully deliver per unit time;

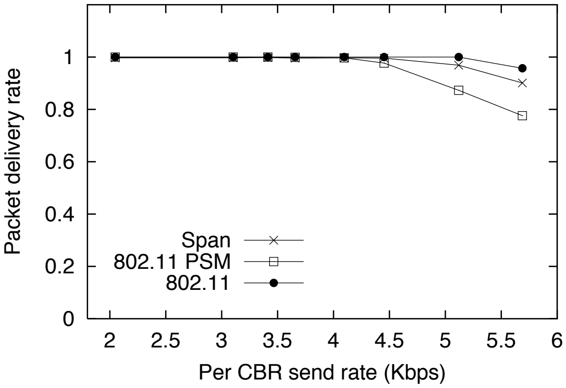


Fig. 5: Packet delivery rate as a function of per-CBR-flow bit rate. Each packet traverses six hops. Under higher traffic load, Span delivers more packets than 802.11 PSM, but slightly less than 802.11.

capacity is inversely proportional to the network's packet loss rate. Additionally, we show that despite using fewer nodes to forward packets, Span does not significantly increase delivery latency and number of hops each packet traverses.

Figure 5 shows packet delivery rate as the bit rate of each CBR flow increases. There is no motion in these simulations. The simulation region has an area of 1000 meters × 1000 meters. On average, each packet traverses 6 hops.

Unmodified 802.11 PSM drops significantly more packets than Span when the CBR flow rate increases past 4 Kbps. Most of these packet drops occur either because the ATIM window is not long enough to allow all buffered unicast packets to be advertised, or because after the ATIM window there is not enough time until the start of the next beacon period for all advertised packets to be transmitted. After two beacon periods of buffering, all packets are dropped by the MAC. Because Span does not need to advertise traffic between coordinators and uses a shorter ATIM window and longer beacon period, Span delivers more packets.

Span has higher loss rates than regular 802.11 when the bit rate increases beyond 4.5 Kbps. This increase in loss rate is largely due to the fact that Span uses a 20 ms ATIM window per 300 ms beacon period, which reduces utilization by 6.7%. Additionally, using fewer nodes to forward packets may decrease potential channel utilization even more: each time a node exponentially backs off to avoid collision, there is a greater chance that the channel becomes unoccupied for a longer period of time.

Table 3 shows the routing behavior and loss rates of Span, 802.11 PSM, and 802.11 with a 3 Kbps per CBR flow rate. We vary the simulation area to change node density and the number of hops each packet needs to traverse. There is no motion in these simulations. Despite using fewer nodes to forward packets, Span delivers packets using only a slightly higher number of hops. Span's packet delivery latency is higher than that of 802.11, but significantly lower than that of 802.11 PSM. With 802.11 PSM, each hop accounts for roughly 200 ms of latency, which corresponds with the 200 ms beacon period used.

Span reduces the number of voids encountered by geographic routing: coordinators are elected to connect neighboring nodes, and are therefore unlikely to occur at the edge of a void. Thus, Span has a lower loss rate than both 802.11 and 802.11 PSM when the node density is low.

These results show that Span does not significantly degrade network capacity, and can forward more packets than 802.11 PSM under high load. Furthermore, Span increases packet latency only slightly, despite using a fewer number of nodes to forward packets.

5.3 Effects of Mobility

Figure 6 shows the effects of mobility on packet loss rate. In these simulations an area of 1000 meters × 1000 meters is used. Each simulation lasts 400 seconds. Nodes follow the random waypoint motion model, and the length of the pause time reflects the degree of mobility.

The degree of mobility does not significantly affect routing with Span coordinators. Span consistently performs better than both 802.11 PSM and 802.11. Most packet drops in these simulations are caused by temporary voids created by mobility. Because geographic forwarding with Span encounters fewer voids, its loss rate is lower.

5.4 Coordinator Election

Ideally, Span would choose just enough coordinators to preserve connectivity and capacity, but no more; any coordinators above this minimum just waste power. This section compares the number of coordinators Span chooses with the number that would be required to form a hexagonal grid layout, shown in Figure 7; the hex grid layout of nodes, while

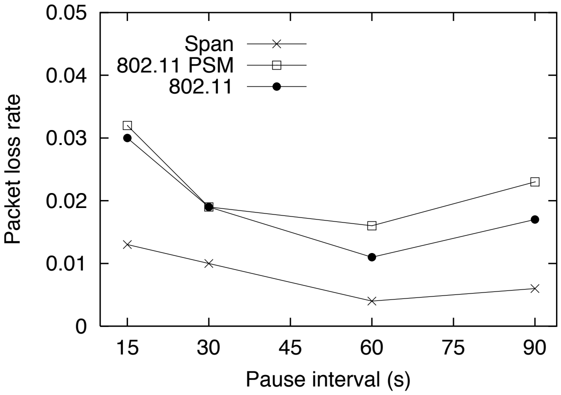


Fig. 6: Packet loss rate as a function of pause time. The simulation area is a 1000 meter × 1000 meter square. Mobility does not affect Span very much, and geographic forwarding with Span delivers more packets than with 802.11 PSM and 802.11 because it encounters fewer voids.

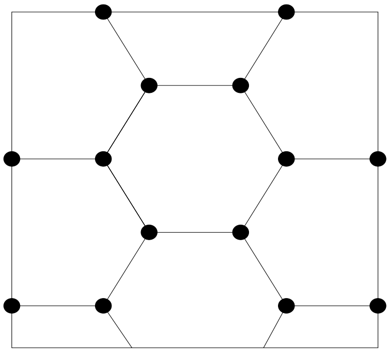


Fig. 7: An approximation to an optimal layout of coordinators in a 1000 meter × 1000 meter area. There are 14 coordinators in this layout.

perhaps not optimal, produces a connected backbone in every direction with very few coordinators.

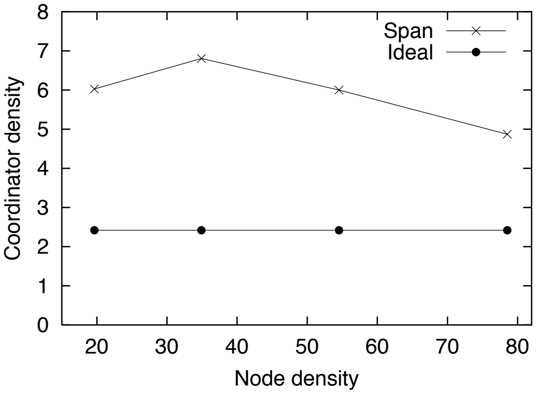


Fig. 8: Ideal and actual coordinator density as a function of node density. The ideal curve represents an approximate lower bound on the number of coordinators needed. Span elects more coordinators than the ideal case because of lower node density, coordinator rotation, and announcement collision.

The hexagonal grid layout of coordinators place a coordinator at each vertex of a hexagon. Every coordinator can communicate with the three coordinators that it is connected to through an edge of a hexagon, which is 250 meters long (the radio range). Each hexagon has six coordinators, but each coordinator is shared by three hexagons. Therefore each hexagon is only responsible for two coordinators. Each hexagon has an area of 162,380 m^2. Thus, given a simulation area of d^2 meters, the number of coordinators expected in this area, C_{ideal} is

$$C_{ideal} = 2 \cdot \frac{d^2}{162380} \tag{4}$$

Figure 8 shows coordinator density as a function of node density. For each node density, coordinator density is computed from the average number of coordinators elected by Span over 500 seconds of five mobile simulations. Points on the "Ideal" curve in Figure 8 are computed using the ideal number of coordinators predicted by Equation 4.

Span elects more coordinators than Equation 4 suggests. There are two reasons for this. First, Equation 4 describes a layout in a network that is dense enough such that there is a node at every corner of every hexagon. When the node density is moderate, on the other hand, more nodes are needed to provide connectivity between the hexagons. Second, to rotate coordinators among all nodes, the optimal set of coordinators may not always be selected.

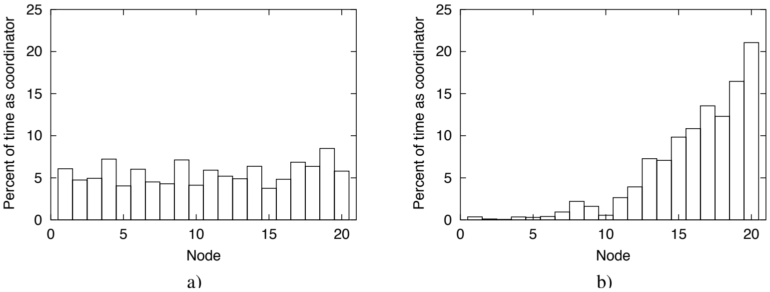


Fig. 9: Percent of time each node in a 20 node, 100 meter × 100 meter network spent as a coordinator during 7200 seconds of simulation. In a), each node starts with 10,000 J of energy. This graph shows that Span rotates coordinators equally among all the nodes. In b), each node starts with $2000{+}400i$ J of energy, where i is the node ID. This graph shows that Span is more likely to elect coordinators with more energy.

Figure 9 shows the percent of time a node in a 20 node, 100 meter × 100 meter network spent as a coordinator during 7200 seconds of simulation. Because the entire network falls within a single radio range, only one node is elected as a coordinator at any given time. Consequently, if Span rotates the coordinator equally among all nodes in the network, each node should spend 5% of the total simulation time as a coordinator, as shown in Figure 9a. In this simulation, each node starts with 10,000 J of energy, and spends roughly the same amount of time as the coordinator. In Figure 9b, each node starts with $2000{+}400i$ J of energy, where i is the node ID. For example, the first node starts out with 2400 J, the second node starts out with 2800 J, and so on. Figure 9b shows that the energy term in Equation 2 allows Span to elect nodes with high amount of energy as coordinators.

5.5 Energy Consumption

This section evaluates Span's ability to save energy. The potential for savings depends on node density, since the fraction of sleeping nodes depends on the number of nodes per radio coverage area. The energy savings also depends on a radio's power consumption in sleep mode and the amount of time that sleeping nodes must turn on their receivers to listen for 802.11 beacons and Span `HELLO` messages.

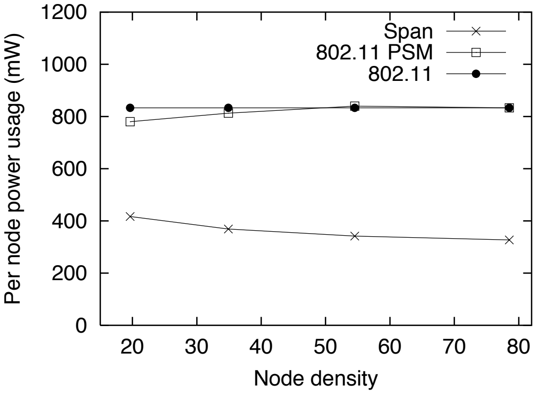


Fig. 10: Per-node power usage. Span provides significant amount of savings over 802.11 PSM and 802.11.

Figure 10 shows the per-node power usage in networks running Span, 802.11 PSM, and 802.11. These numbers are calculated from the initial energy and the energy remaining at each of the 100 mobile nodes over 500 seconds. Each value is an average over 5 mobile simulations. From these results, we find that Span provides a considerable amount of energy savings over 802.11, while 802.11 PSM saves essentially no power. This is because geographic forwarding needs to send broadcast messages. With 802.11 PSM, each time a node receives a broadcast advertisement, it must stay up for the entire beacon period. This prevents non-coordinators from going back to sleep. When the node density is low, the number of broadcast messages in a radio range decreases, and 802.11 PSM yields a small amount of energy savings.

We also find that as density increases, a smaller fraction of the nodes are elected coordinators. Consequently, we expect energy savings to increase. In practice, however, energy savings do not increase as much.

To understand why, we estimate the amount of energy used in a Span system based on an estimate of the average fraction of time a node must run its radio in idle mode. We call this fraction f_{idle}:

$$f_{idle} = \frac{C}{N} + \left(1 - \frac{C}{N}\right) \cdot f_{up} \tag{5}$$

In Equation 5, N is the total number of nodes, C is the number of coordinators elected, and f_{up} is the fraction of the time a node in sleep mode must wake up to listen for beacons and `HELLO` messages. Span uses a 20 ms ATIM window per 300 ms of beacon period. Thus the smallest value for f_{up} is 0.067. In the worst case, f_{up} can be as high as 0.333, when a non-coordinator node must stay up for the entire duration of the advertised traffic window (100 ms).

We define α as the ratio of the power consumption of the radio in sleep mode to the power consumption of the radio in idle mode. Then, using f_{idle}, the amount of energy savings can be estimated as

$$\frac{1}{f_{idle} + \alpha * (1 - f_{idle})} \tag{6}$$

Note that because f_{idle} depends on $\frac{C}{N}$, and that the coordinator density stays the same for different node densities, the gain in energy savings also depends on the node density.

Figure 11 plots Equation 6 as a function of α, substituting C_{ideal} and 0 as values for C and f_{up}. This figure shows that the amount of energy saving increases rapidly, as the value of α decreases. Our energy model uses $\alpha = 0.157$ from measurements. Figure 12 plots Equation 6 as a function of f_{up}, using C_{ideal} and 0.157 as values for C and α. This figure shows that as f_{up} increases, the gain in energy savings decreases as well. These two figures explain why in Figure 10, the gain in energy savings is a sub-linear function of node density.

We can calculate the actual values of f_{up} in our experiments using statistics gathered from the simulations, summarized in Table 4. The numbers in the f_{up} column are calculated using Equation 5, using values from the "Idle time" column as f_{idle}. We substitute C/N with numbers in the "Time as coordinator" column divided by 500 seconds. This column suggests that Span broadcast messages are expensive when density is high—the large number of broadcast messages per radio range keeps nodes awake for a longer period of time.

The numbers in the "Power" column in Table 4 correspond to the data points in Figure 10. The "Tx/Rx power" column shows the amount of energy used to send and receive broadcast and data packets.

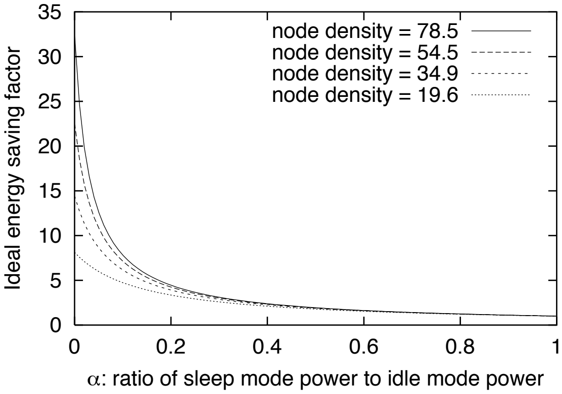


Fig. 11: Energy savings as a function of α, computed using Equation 6, substituting C_{ideal} and 0 as values for C and f_{up} when computing f_{idle}. This figure shows that as α increases, the amount of energy saving decreases significantly. In our experiments, we used $\alpha = 0.16$.

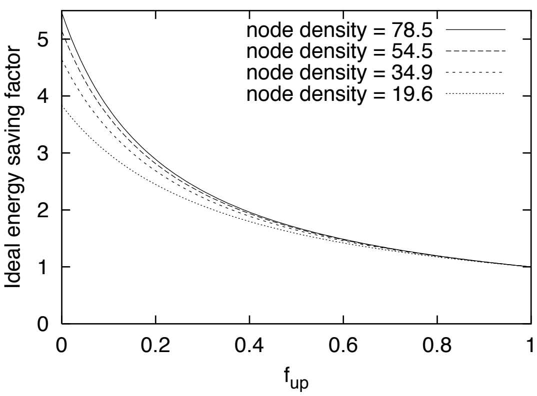


Fig. 12: Energy savings as a function of f_{up}, computed using Equation 6, substituting C_{ideal} and 0.157 as values for C and α. This figure shows that as f_{up} increases, the amount of energy saving decreases as well. In our experiments, f_{up} was between 0.185 and 0.263.

Density	Sleep time	Idle time	Time as coordinator	Power	Tx/Rx power	f_{up}
78.5	374 s	126 s	32.7 s	327 mW	21 mW	0.201
54.5	364 s	136 s	46.6 s	342 mW	22 mW	0.197
34.9	348 s	152 s	75.8 s	369 mW	26 mW	0.180
19.6	318 s	182 s	121.3 s	417 mW	32 mW	0.160

Table 4: Amount of time each node spends in sleep and idle mode, as a coordinator, and the energy consumption of each node as node density changes. The Tx/Rx power column shows the power used to transmit and receive data and broadcast packets. It shows that the energy spent routing packets are not significant. The f_{up} column shows the fraction of each beacon period that a node is awake. At higher densities, broadcast messages keep each node up for a longer period of time.

Numbers in this column are calculated by subtracting from numbers in the "Power" column the power used by the node in idle and sleep modes, without sending or receiving packets. For example, when density is 78.5 nodes per radio rage, a node spends 374 of the 500 seconds in sleep mode, and only 126 seconds in idle mode. Given that the node uses 830 mW in idle mode and 130 mW in sleep mode (see Table 2), if the node is not sending or receiving packets, its power consumption should be 306 mW. The fact that the node's actual power consumption is 327 mW implies that sending and receiving packets use 21 mW. Numbers in the "Tx/Rx power" column suggest that for the kind of radios we are using, sending and receiving packets do not consume much energy in comparison.

Results in this section show that Span reduces per node power consumption by a factor of 2 or more over 802.11 PSM and 802.11. However, the amount of energy savings does not increase significantly as node density increases.

5.6 Node Lifetime

This section shows that Span distributes the costs of being a coordinator in a way that extends the useful lifetime of every node in the network. Figures 13 shows results from several mobile experiments. In these experiments, the 20 source and destination nodes start with 2000 Joules of energy, and the remaining 100 forwarding nodes start with 300 Joules of energy. The 802.11 and PSM curves represent simulation results on a 500m×500m area. With 100 nodes routing packets,

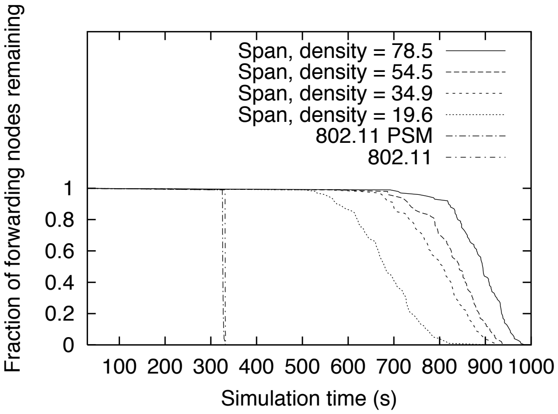


Fig. 13: Fraction of nodes remaining as a function of simulation time. With Span, nodes remain alive for significantly longer periods of time.

the node density is 78.5 nodes per radio range. Results with other node densities are similar. Span curves represent results over several node densities. Without Span, nodes critical to multi-hop routing die around the same time, 335 seconds into the simulation. With Span, the first node failure occurs 505 seconds into the simulation when node density is 19.6 nodes per radio range, 556 seconds into the simulation when node density is 34.9, 574 seconds into the simulation when node density is 54.5, and 692 seconds into the simulation when node density is 78.5. The packet delivery rate does not drop below 90% until 681 seconds into the simulation when node density is 19.6, 887 seconds into the simulation when node density is 34.9, 912 seconds into the simulation when node density is 54.5, and 962 seconds into the simulation when node density is 78.5.

6 Conclusion

This paper presents *Span*, a distributed coordination technique for multi-hop ad hoc wireless networks that reduces energy consumption without significantly diminishing the capacity or connectivity of the network. Span adaptively elects *coordinators* from all nodes in the network, and rotates them in time. Span coordinators stay awake and perform multi-hop packet routing within the ad hoc network, while other nodes remain in power-saving mode and periodically check if they should awaken and become a coordinator.

With Span, each node uses a random backoff delay to decide whether to become a coordinator. This delay is a function of the number of other nodes in the neighborhood that can be bridged using this node, and the amount of energy it has remaining. Our results show that Span not only preserves network connectivity, it also preserves capacity, decreases latency, and provides significant energy savings. For example, for a practical range of node densities and a practical energy model, our simulations show that the system lifetime with Span is more than a factor of two better than without Span.

The amount of energy that Span saves increases only slightly as density increases. This is largely due to the fact that the current implementation of Span uses the power saving features of 802.11, in which nodes periodically wake up and listen for traffic advertisements. Section 5.5 shows that this approach can be extremely expensive. This warrants investigation into a more robust and efficient power saving MAC layer, one that minimizes the amount of time each node in power saving mode must stay up.

7 Acknowledgments

We thank John Ankcorn for providing power measurement hardware and advice.

References

1. Wireless LAN Medium Access Control and Physical Layer Specifications, August 1999. IEEE 802.11 Standard (IEEE Computer Society LAN MAN Standards Committee).
2. J Broch, D Johnson, and D Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. In *Internet draft, IETF Mobile Ad Hoc Networking Working Group*, December 1998.
3. J Broch, D Maltz, D Johnson, Y Hu, and J Jetcheva. A Performance Comparison of Multi-hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Dallas, TX, August 1998.
4. J. Chang and L. Tassiulas. Energy Conserving Routing in Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM*, Tel Aviv, Israel, 2000.
5. G. Chesson. XTP/protocol engine design. In *Proceedings of the IFIP WG6.1/6.4 Workshop*, Rüschlikon, May 1989.

6. Bevan Das and Vaduvur Bharghavan. Routing in Ad-Hoc Networks Using Minimum Connected Dominating Sets. In *Proceedings of the IEEE International Conference on Communications (ICC '97)*, June 1997.
7. Laura Feeney and Martin Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *Proceedings of IEEE INFOCOM*, Anchorage, AK, 2001.
8. W. Fenner. *Internet Group Management Protocol, Version 2*, Nov 1997. RFC-2236.
9. S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing. In *Proceedings of the ACM SIGCOMM*, pages 342–356, Boston, MA, September 1995.
10. Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient Communication Protocols for Wireless Microsensor Networks. In *Proceedings of the Hawaaian International Conference on Systems Science*, January 2000.
11. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Boston, MA, August 2000.
12. Brad Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Boston, MA, August 2000.
13. Robin Kravets and P. Krishnan. Application-driven Power Management for Mobile Communication. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Dallas, TX, October 1998.
14. Jinyang Li, John Jannotti, Douglas De Couto, David Karger, and Robert Morris. A Scalable Location Service for Geographic Ad-Hoc Routing. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, August 2000.
15. William Mangione-Smith, Phil Seong Ghang, Sean Nazareth, Paul Lettieri, Walk Boring, and Rajeev Jain. A Low Power Architecture for Wireless Multimedia Systems: Lessons Learned From Building a Power Hog. In *1996 International Symposium on Low Power Electronics and Design Digest of Technical Papers*, Monterey, CA, August 1996.
16. CMU Monarch Extensions to `ns`. `http://www.monarch.cs.cmu.edu/`.
17. ns Notes and Documentation. `http://www.isi.edu/vint/nsnam/`, 2000.
18. C.S. Raghavendra and Suresh Singh. PAMAS: Power Aware Multi-Access Protocol with Signaling for Ad Hoc Networks. *ACM Computer Communication Review*, pages 5–26, July 1998.
19. R. Ramanathan and R. Rosales-Hain. Topology Control of Multi-hop Wireless Networks Using Transmit Power Adjustment. In *Proceedings of IEEE INFOCOM*, Tel Aviv, Israel, March 2000.

20. V. Rodoplu and T. H. Meng. Minimum Energy Mobile Wireless Networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, volume 3, pages 1633–1639, Atlanta, GA, June 1998.
21. Christian Rohl, Hagen Woesner, and Adam Wolisz. A Short Look on Power Saving Mechanisms in the Wireless LAN Standard Draft IEEE 802.11. In *Proceedings of the the 6th WINLAB Workshop on Third Generation Wireless Systems*, New Brunswick, NJ, March 1997.
22. Timothy Shepard. A Channel Access Scheme for Large Dense Packet Radio Networks. In *Proceedings of the ACM SIGCOMM*, pages 219–230, Stanford University, CA, August 1996.
23. Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-Aware Routing in Mobile Ad Hoc Networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Dallas, TX, 1998.
24. Mark Stemm and Randy Katz. Reducing Power Consumption of Network Interfaces in Hand-held Devices. In *Proceedings of the Third Workshop on Mobile Multimedia Communications (MoMuC-3)*, Princeton, NJ, 1996.
25. Roger Wattenhofer, Li Li, Paramvir Bahl, and Yi-Min Wang. Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM*, Anchorage, AK, 2001.
26. Jie Wu and Ming Gao. On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks. In *Proceedings of the 30th Annual International Conference On Parallel Processing*, Valencia, Spain, September 2001.
27. Jie Wu and Hailan Li. On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks. In *Proceedings of the Third International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Seattle, WA, August 1999.
28. Ya Xu, John Heidemann, and Deborah Estrin. Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks. Technical Report 527, USC/ISI, October 2000.
29. Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 70–84, Rome, Italy, July 2001.

Proc. of the 6th Annual International
Conference on Mbile Computing and
Networking (MOBICOM 2000),
Boston, MA, August 2000.

The Cricket Location-Support System

Nissanka B. Priyantha, Anit Chakraborty and Hari Balakrishnan *

MIT Laboratory for Computer Science
{`bodhi, achakra, hari`}`@lcs.mit.edu`

Abstract. This paper presents the design, implementation, and evaluation of *Cricket,* a location-support system for in-building, mobile, location-dependent applications. It allows applications running on mobile and static nodes to learn their physical location by using *listeners* that hear and analyze information from *beacons* spread throughout the building. Cricket is the result of several design goals, including user privacy, decentralized administration, network heterogeneity, and low cost. Rather than explicitly tracking user location, Cricket helps devices learn where they are and lets them decide whom to advertise this information to; it does not rely on any centralized management or control and there is no explicit coordination between beacons; it provides information to devices regardless of their type of network connectivity; and each Cricket device is made from off-the-shelf components and costs less than U.S. $10. We describe the randomized algorithm used by beacons to transmit information, the use of concurrent radio and ultrasonic signals to infer distance, the listener inference algorithms to overcome multipath and interference, and practical beacon configuration and positioning techniques that improve accuracy. Our experience with Cricket shows that several location-dependent applications such as in-building active maps and device control can be developed with little effort or manual configuration.

1 Introduction

The emergence of network-enabled devices and the promise of ubiquitous network connectivity has made the development of pervasive

---

* This research was supported in part by NTT Corporation, DARPA (Grant No. MDA972-99-1-0014), and IBM.

computing environments an attractive research goal. A compelling set of applications enabled by these technology trends are context-aware, *location-dependent* ones, which adapt their behavior and user interface to the current location in space, for which they need to know their physical location with some degree of accuracy. We have started seeing the commercial deployment of such applications in outdoor settings (e.g., Hertz's NeverLost navigator on rental cars [12]), where location information is obtained via wide-area technologies like the Global Positioning System (GPS) [10] or using the cellular infrastructure. We believe that the widespread deployment of location-dependent applications *inside* office buildings and homes has the potential to fundamentally change the way we interact with our immediate environment, where computing elements will be "ubiquitous" [20] or "pervasive" [8, 4]. In particular, our work will enable a new class of location-based applications and user interactions in the context of Project Oxygen at MIT [15].

The design and deployment of a system for obtaining location and spatial information in an indoor environment is a challenging task for several reasons, including the preservation of user privacy, administration and management overheads, system scalability, and the harsh nature of indoor wireless channels. The degree of privacy offered by the system is an important deployment consideration, since people often value their privacy highly. The administrative overhead to manage and maintain the hardware and software infrastructure must be minimal because of the potentially large number (possibly several thousands in a building) of devices and networked services that would be part of the system, and the communication protocols must be able to scale to a high spatial density of devices. Finally, indoor environments often contain substantial amounts of metal and other such reflective materials that affect the propagation of radio frequency (RF) signals in non-trivial ways, causing severe multipath effects, dead-spots, noise, and interference.

Our goal is to develop a system that allows applications running on user devices and service nodes to learn their physical location. Once this information is obtained, services advertise themselves to a resource discovery service such as the MIT Intentional Naming System (INS) [2], IETF Service Location Protocol [17], Berkeley Service Discovery Service [7], or Sun's Jini discovery service [13]. User applications do not advertise themselves unless they want to be discovered by others; they learn about services in their vicinity via an active map that is sent from a map server application, and interact with services by constructing queries for services at a required location. By separating the processes of tracking services and obtaining location information, multiple resource discovery systems can be handled. By not tracking users and services, user-privacy concerns are adequately met. We emphasize that our goal is a *location-support* system, rather than a conventional *location-tracking* system that tracks and stores location information for services and users in a centrally maintained database.

Over the past many months, we have designed and implemented *Cricket*, a location-support system for building-wide deployment in the context of Project Oxygen, and have conducted several experiments with it. We have integrated it with INS for resource discovery, and an active map application, which together enable location-dependent applications (and users) to discover and interact with services. This paper describes our design goals (later in this section), system architecture and algorithms (Section 2), implementation (Section 3), experimental results (Section 4), applications (Section 5), and a detailed comparison with previous location-tracking systems (Section 6).

The design of Cricket was driven by the following specific goals, which followed from the nature of our applications and from deployment considerations:

- **User privacy.** Whenever a system for providing location information to clients has been deployed in the past, the issue of user privacy has arisen. This is because many previous systems were location *tracking* systems, where a database kept track of the locations of all the entities, including users in the system. To address this concern, we designed a location *support* system, which allows clients to learn their location without centralized tracking in order to construct location-specific queries for resources.
- **Decentralized administration.** Our goal is widespread building-wide deployment. We believe that it is not possible to deploy and administer a system in a scalable way when all control and management functions are centralized. Our design is decentralized - the "owner" of a space in a building (e.g.,the occupant of a room) configures and installs a location *beacon* that announces the identity of that space (some character string) and each beacon seamlessly integrates with the rest of the system. Location receiver hardware, called a listener, is attached to every device of interest to a user. Listeners use an inference algorithm to determine the space in which they are currently located by listening to beacon announcements. And there is no need to keep track of individual components within the system.
- **Network heterogeneity.** A wide variety of network technologies exist in most building environments. In our own laboratory, de-

vices and users connected over 10/100 Mbps Ethernet, three different types of indoor wireless LANs, cellular digital packet data (CDPD), infrared, public telephone, and power-line using X10 [21]. Independent of which technology they use to serve or gain access to information, many services and clients can benefit from learning their location in an automatic way, and we would like to accommodate them. In our design, we achieve this by decoupling the Cricket system from other data communication mechanisms.
- **Cost.** Achieving building-wide deployment requires cost-effective components. We use commercial, off-the-shelf, inexpensive components in Cricket, setting and meeting the goal of less than U.S. $10 per location beacon and listener. Our design involves no custom hardware and is small enough to fit in one's palm.
- **Room-sized granularity.** Our goal is a system where spatial regions can be determined to within a few square feet, so as to distinguish portions of rooms. This requires the ability to demarcate and determine boundaries between regions corresponding to different beacons.

Cricket uses a combination of RF and ultrasound to provide a location-support service to users and applications. Wall- and ceiling-mounted beacons are spread through the building, publishing location information on an RF signal. With each RF advertisement, the beacon transmits a concurrent ultrasonic pulse. The listeners receive these RF and ultrasonic signals, correlate them to each other, and infer the space they are currently in. We describe the details of the technologies, the system parameters and configuration, and the algorithms and protocols used in Cricket. The beacons use a decentralized randomized transmission algorithm to minimize collisions and interference amongst each other. The listeners implement a decoding algorithm to overcome the effects of ultrasound multipath and RF interference. We investigate the performance of three decoding algorithms and find that picking the location corresponding to the beacon with minimum statistical mode performs the best, maximizing the likelihood of making the correct choice. We also discuss some practical deployment considerations when using ultrasound hardware, and some location-dependent applications we have developed using Cricket.

2 System architecture

Cricket uses *beacons* to disseminate information about a geographic space to *listeners*. A beacon is a small device attached to some location within the geographic space it advertises. Typically, it is obtained by the "owner" of the location (e.g., the occupant of a room in an office or home, or a building administrator) and placed at an unobtrusive location like a ceiling or wall. Cricket does not attach any semantics to the space advertised by the beacon; any short string can be disseminated, such as the name of a server to contact to learn more about the space or a name resolver for the space to discover resources. Cricket beacons are inexpensive and more than one of them can be used in any space for fault-tolerance and better coverage.

To obtain information about a space, every mobile and static node has a listener attached to it. A listener is a small device that listens to messages from beacons, and uses these messages to infer the space it is currently in. The listener provides an API to programs running on the node that allow them to learn where they are, so that they can use this information to appropriately advertise themselves and their location to a resource discovery service.

The listener can be attached to both static and mobile nodes. For example, when a user attaches a new static service to the network (e.g., a printer), she does not need to configure it with a location or other any attribute; all she does is attach a listener to it. Within a few seconds, the listener infers its current location from the set of beacons it hears, and informs the device software about this via the API. This information can then be used in its own service advertisements. When a mobile computer has a listener attached to it, the listener constantly listens to beacons to infer its location. As the computer (e.g., a handheld computer carried by a person) moves in a building, the navigation software running on it uses the listener API to update its current location. Then, by sending this information securely to a map server (for example), it can obtain updates to the map displayed to the user. Furthermore, services appear as icons on the map that are a function of the user's current location. The services themselves learn their location information using their own listener devices, avoiding the need for any per-node configuration.

The only configuration required in Cricket is setting the string for a space that is disseminated by a beacon. The specific string is a function of the resource discovery protocol being used, and Cricket allows any one of several possibilities (in Section 5 we describe our implementation platform and integration with INS). Cricket also provides a way by which the owner of a room can securely set and change the space identifier that is sent in the advertisements. This is done by sending a special message over the same RF channel that is used for the advertisements, after authenticating the user via a password. At this stage, we have chosen to allow this change only from within physical proximity of the room or location where the beacon is located. This makes the system somewhat more secure than if we allowed this to be done from afar.

The boundaries between adjacent spaces can either be real, as in a wall separating two rooms, or virtual, as in a non-physical partition used to separate portions of a room. The *precision* of the system is determined by how well the listener can detect the boundary between two spaces, while the *granularity* of the system is the smallest possible size for a geographic space such that boundaries can be detected with a high degree of precision. A third metric, *accuracy* is used to calibrate individual beacons and listeners; it is the degree to which the distance from a beacon, estimated by a listener, matches the true distance. While our experiments show that the distance accuracy of our hardware is smaller than a few inches, what matters is the precision and granularity of the system. These depend on the algorithms and the placement of beacons across boundaries. Our goal is a system with a close-to-100% precision with a granularity of a few feet (a portion of a room).

The rest of this section describes the design of Cricket, focusing on three fundamental issues: (i) mechanism for determining the location (the beacon-listener protocol), (ii) the listener algorithms and techniques for handling beacon interference, and (iii) beacon configuration and positioning.

2.1 Determining the location

At the beginning we were hopeful that a purely RF-based system could be engineered and made to work well, providing location information at the granularity of a room, and ideally, portions of rooms. Our approach attempted to limit the coverage of an RF transmitter to define the granularity of a geographic-space, and using received signal strength to infer best location. Despite many weeks of experimentation and significant tuning, this did not yield satisfactory results [6]. This was mainly because RF propagation within buildings deviates heavily from empirical mathematical models (e.g., see also [5]), and in our environment, the corresponding signal behavior with our inexpensive, off-the-shelf radios was not reproducible across time.

We therefore decided to use a combination of RF and ultrasound hardware to enable a listener to determine the distance to beacons, from which the closest beacon can be more unambiguously inferred. We achieve this by measuring the one-way propagation time of the ultrasonic signals emitted by a beacon, taking advantage of the fact that the speed of sound in air (about 1.13 ft/ms at room temperature) is much smaller than the speed of light (RF) in air. On each transmission, a beacon *concurrently* sends information about the space over RF, together with an ultrasonic pulse. When the listener hears the RF signal, it uses the first few bits as training information and then turns on its ultrasonic receiver. It then listens for the ultrasonic pulse, which will usually arrive a short time later. The listener uses the time difference between the receipt of the first bit of RF information and the ultrasonic signal to determine the distance to the beacon. Of course, the value of the estimated distance is not as important as the decision of which the closest beacon is.

The use of time-of-flight of signals to measure distance is not a new concept. GPS uses the one-way delay of radio waves from satellites to estimate distance, while radio-altimeters in aircrafts use the time for an electromagnetic signal to reflect off the ground to determine altitude. Collision avoidance mechanisms used in robotics [16] determine the distance to obstacles by measuring the time-of-flight of an ultrasonic signal being bounced off them.

It is also possible to measure the distance using the relative velocity of two signals. It is common practice to use the time elapsed between observing a lightning (electromagnetic waves) and accompanied thunder (sound) to estimate the distance to the lightning. The Bat system (detailed in Section 6) uses this idea to determine a mobile transmitter's position in space, where an array of calibrated receivers measure the time of flight of an ultrasonic signal emitted by a mobile transmitter in response to an RF signal from a base station sent to the transmitter and all the receivers.

2.2 Reducing interference

While Cricket has the attractive property that its decentralized beacon network is easy to configure and manage, it comes at the absence of explicit coordination. There is no explicit scheduling or coordination between the transmissions of different beacons that may be in close proximity, and listeners do not transmit any information to avoid compromising privacy. This lack of coordination can cause RF transmissions from different beacons to collide, and may cause a listener to wrongly correlate the RF data of one beacon with the ultrasonic signal of another, yielding false results. Furthermore, ultrasonic reception suffers from severe multipath effects caused by reflections from walls and other objects, and these are orders of magnitude longer in time than RF multipath because of the relatively long propagation time for sound waves in air. In fact, this is one of the reasons it is hard to modulate data on the ultrasonic signal, which makes it a pure pulse. Thus, the listener's task is to gather various RF and ultrasound (US) samples, deduce and correlate the {RF,US} pairs that were sent concurrently by the different beacons, and choose the space identifier sent from the pair with the closest distance.

We decided not to implement a full-fledged carrier-sense-style channel-access protocol to avoid collisions in order to maintain simplicity and reduce overall energy consumption. Instead, we handle the problem of collisions using *randomization*. Rather than using a fixed or deterministic transmission schedule, beacon transmission times are chosen randomly with a uniform distribution within an interval $[R1, R2]$ms. Thus, the broadcasts of different beacons are statistically independent, which avoids repeated synchronization and prevents persistent collisions. The choice of random interval is governed by the number of beacons we typically expect will be within range of each other and the time it takes for the transmitted information to reach the listeners, which depends on the message size and link bandwidth. In our implementation, we use an average frequency of four times per second distributed in $[150, 350]$ms. A smaller frequency increases the amount of time before a statistically significant location inference can be made, while a higher frequency increases the probability of collisions. We plan to extend this technique to include a listening component that will allow each beacon to infer the number of beacons in its proximity and appropriately scale the beaconing frequency.

We minimize errors due to RF and ultrasonic interference among beacons by two methods: (i) proper selection of system parameters to reduce the chance of false correlations, and (ii) listener inference algorithms based on statistical analysis of correlated {RF,US} samples.

System parameters In addition to transmitting a string corresponding to the space, each beacon transmits a unique identifier. The combination of the location string and identifier is unique across the entire system. This allows the listener to correlate the RF and ultrasonic beacon signals correctly.

The raw line-of-sight range of our ultrasonic transmitter-receiver pair is around 50 feet, when both the transmitter and the receiver are facing each other. However, by mounting the ultrasonic transmitters carefully, as described in Section 3.3, we are able to reduce the effective range to around 30 feet in the absence of any obstacles. The line-of-sight range of the RF transmitter-receiver pair is about 80 feet, which drops to about 40 feet when there is an obstacle (e.g., a wall). Since RF can travel farther than an ultrasonic transmission and can also travel through certain obstacles, it is almost impossible for a listener to receive an ultrasonic signal without receiving the corresponding RF signal.

We discovered that one way to reduce the occurrence of false correlations is to use a relatively *sluggish* RF data transmission rate! Instead, if we used a high-bandwidth RF channel, the data identifying a space would reach a listener before the ultrasound pulse was detected. I.e., if S is the size in bits of the message sent over the RF channel with a transmission rate of b bits/s, and τ is the maximum propagation time for an ultrasonic signal in air between a beacon and a listener, a value of $b < S/\tau$ would mean that the ultrasonic signal corresponding to a given RF message would arrive *while* the S message bits are still being received. Together with the fact that the range of our ultrasound is smaller than our RF, this establishes that *any* potentially correlated ultrasound pulse *must* arrive while an RF message is being received. In the absence of interfering beacon transmissions, this check suffices to do the correct correlation. The specific parameters used in our implementation are described in Section 3.

We now proceed to investigate the different interference scenarios that are possible.

Interference scenarios To better understand the effects of interference and multipath (due to reflected signals) on distance estimation, we characterize the different RF and ultrasonic signals that a listener can hear. Consider the RF and ultrasonic signals sent by a beacon A and an interfering beacon I. The listener potentially hears the following signals:

- RF-A. The RF signal from A.
- US-A. The *direct* ultrasonic signal from A.
- US-RA. The *reflected* ultrasonic signal from A.
- RF-I. The RF signal from I.
- US-I. The *direct* ultrasonic signal from I.
- US-RI. The *reflected* ultrasonic signal from I.

We only need to consider the cases when a US pulse arrives while some RF signal is being received. The reception of the first ultrasonic signal US-A, US-RA, US-I, or US-RI while RF-A is being received will cause the listener to calculate the distance to A using the time interval between the detection of RF-A and the particular ultrasonic signal. This is because the listener, after receiving the RF signal from a beacon, waits for the first occurrence of an ultrasonic pulse to determine the

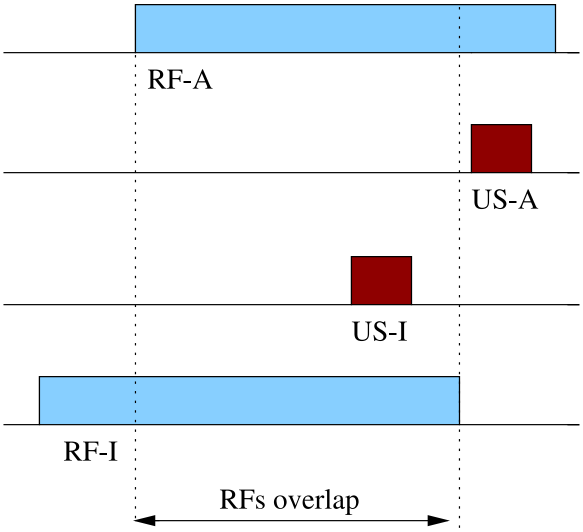


Fig. 1. RF-A:US-I interaction, with US-A arriving after US-I. The two RF transmissions overlap in time at the listener.

distance. All subsequent ultrasonic receptions that arrive during this RF message are ignored. Of course, if the direct signal US-A is the first one to be received, the listener correctly estimates the distance to A. However, the wrong correlation of any other ultrasonic signal with RF-A could be problematic.

Case 1: RF-A:US-RA. This combination with the reflected ultrasonic signal from A causes the estimated distance to be larger than the actual distance to A. This situation can occur only if the direct signal US-A was never received by the listener. However, the problems caused by this to the system can be reduced by properly aligned beacons (Section 3.3), as well as using multiple independent beacons per geographic space. In addition, in our experience, we have found that the ability of the ultrasonic waves to bend around obstacle edges (diffraction) makes this a relatively infrequent occurrence since the direct signal is usually detected before the reflected one.

Case 2: **RF-A:US-I.** This is the combination of RF-A with the direct ultrasonic signal from an interfering beacon I, which arrives *before* the ultrasonic signal US-A. Since an ultrasonic pulse can only be received by a listener while the corresponding RF data packet is being received, RF-I should also be in transit to the listener. Hence RF-A and RF-I should overlap at the listener as shown in Figure 1.

If RF-A and RF-I are comparable in signal strength, they will collide, causing the listener to ignore this event because both RF messages will be corrupted. On the other hand, if the signal strength of RF-I is substantially larger than RF-A, the two may not collide and the listener will end up calculating the *correct* distance to beacon I.

The only situation that leads to a wrong distance estimate is when the signal strength of RF-I is much smaller than RF-A, causing the listener to use the RF-A:US-I combination to determine the distance to A. We reduce the chances of this event by using RF signals with longer range than US signals. This generally ensures a strong RF reception whenever the corresponding ultrasonic signal is received (hence the receipt of US-I, in general ensures a strong RF-I).

Case 3. **RF-A:US-RI.** This occurs when a stray reflected signal from an interfering beacon I appears before US-A. As before, this can lead to wrong distance estimates as well.

Although cases 2 and 3 may lead to incorrect distance estimates, our use of randomization reduces the repeated calculation of wrong estimates. If there are a large number of beacons in close proximity to each other, there can be a non-negligible number of wrong distance estimates at the receivers. At this point, we have engineered our system to ensure that there are not more than five or six beacons that are within range of each other at any location.

In addition, listeners do not simply use the first sample pair they get to infer their best location. Rather, they collect multiple samples and use an inference algorithm for this.

Beacon position inference We develop and compare three simple algorithms to determine which the closest beacon is, overcoming the interference problems of the previous section: *Majority*, *MinMean*, and *MinMode*. In our analysis of these algorithms, the distance estimate is rounded to the nearest ten inches and the data put into different bins according to how frequently they occur. This is done for each beacon separately. Furthermore, isolated stray samples are eliminated from the analysis; a small threshold number of consistent values (two, in our implementation) are needed before the corresponding sample is included for analysis.

- *Majority.* This is the simplest algorithm, which pays no attention to the distance estimates and simply picks the beacon with the highest frequency of occurrence in the data set. This algorithm does not

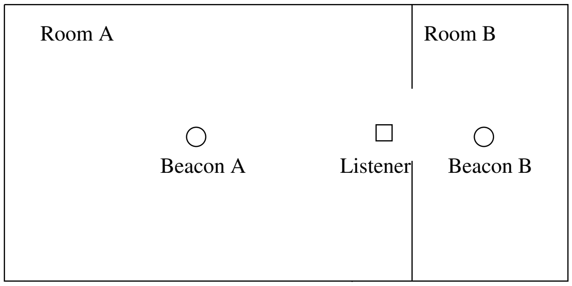


Fig. 2. The nearest beacon to a listener may not be in the same geographic space.

use ultrasonic signals for determining the closest beacon, but as we find in our experiments, this does not perform well. We investigate this primarily for comparison with the other algorithms.

- *MinMean.* Here, the listener calculates the mean distance from each unique beacon for the set of data points within the data set. Then, it selects the beacon with the minimum mean as the closest one. The advantage of this algorithm is that it can be computed with very little state, since a new sample updates the mean in a straightforward way. The problem with this algorithm is that it is not immune to multipath effects that cause the distance estimates to display modal behavior; where computing a statistic like the mean (or median) is not reflective of any actual beacon position.
- *MinMode.* Since the distance estimates often show significant modal behavior due to reflections, our approach to obtaining a highest-likelihood estimate is to compute the per-beacon statistical modes over the past n samples (or time window). For each beacon, the listener then picks the distance corresponding to the mode of the distribution, and uses the beacon that has the minimum distance value from among all the modes. We find that this is robust to stray signals and performs well in both static and mobile cases.

Section 4 discusses the results of our experiments. We note that these are by no means the only possible algorithms, but these are representative of the precision attainable with different degrees of processing at the listeners.

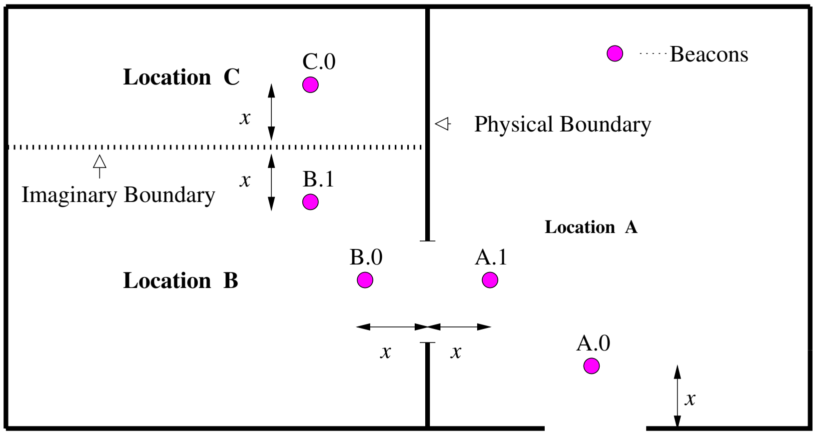


Fig. 3. Correct positioning of beacons.

2.3 Beacon positioning and configuration

The positioning of a beacon within a room or space plays a non-trivial role in enabling listeners to make the correct choice of their location. For example, consider the positioning shown in Figure 2. Although the receiver is in Room A, the listener finds the beacon in Room B to be closer and will end up using the space identifier advertised by the latter.

One way of overcoming this is to maintain a centralized repository of the physical locations of each beacon and provide this data to listeners. Systems like the Bat essentially use this type of approach, where the central controller knows where each wall- or ceiling-mounted device is located, but it suffers from two problems that make it unsuitable for us. First, user-privacy is compromised because a listener now needs to make active contact to learn where it is (observe that in Cricket, a listener is completely passive). Second, it requires a centrally managed service, which does not suit our autonomously managed environment particularly well.

Fortunately, there is a simple engineering solution to this problem that preserves privacy and is decentralized. Whenever a beacon is placed to demarcate a physical or virtual boundary corresponding to a different space, it must be placed at a fixed distance away from the boundary demarcating the two spaces. Figure 3 shows an example of this in a setting with both real and virtual boundaries. Such placement ensures that a listener rarely makes a wrong choice, unless caught within a small distance (1 foot in our current implementation) from the boundary between two beacons advertising different spaces. In this case, it is often equally valid to pick either beacon as the closest.

3 Implementation

In this section, we describe the implementation of Cricket. We describe the system parameters and hardware configuration, the API provided by the listener to applications running on the attached node, and some deployment issues with ultrasonic hardware.

3.1 System parameters and hardware

The message size of a beacon RF transmission is 7 bytes long in our implementation, and the RF transmission rate of our radios is 1200 bits/s. It therefore takes about 47 ms for the message to completely reach a listener, during which time an ultrasonic pulse can travel at most about 47 feet. The typical range of our RF radios is about 30 feet in the building. No listener can therefore be farther away than this to detect which space it is in.

Cricket is implemented using inexpensive, off-the-shelf, simple hardware parts that cost less than U.S. $10 per beacon and listener. The beacon consists of a PIC micro-controller running at 10MHz, with 68 bytes of RAM and 1024 words of program memory. It uses a low-power SAW resonator-based RF transmitter and a single-chip RF receiver, both operating in the 418 MHz unlicensed band [9] with amplitude modulation. The final component is an ultrasonic transmitter operating at 40kHz. All of these are assembled on a small board and mounted on a ceiling or high on a wall.

The listener is only slightly more complicated. It has an identical micro-controller, a single-chip RF receiver, and an ultrasonic receiver with a single-chip tone-detector circuit, instead of the corresponding transmitters. It also has a TTL to RS-232 signal converter by which it interfaces to the host device, e.g., a laptop, hand-held computer, or any other service like a printer, camera, television, etc. This interface uses the standard RS-232 protocol at 9600 bits/s.

We measured the power consumption of a beacon, since the periodic transmission of an RF signal and ultrasonic pulse will eventually run the battery down. Although we did not explicitly design the hardware for low power consumption, we find that it is quite efficient, dissipating 15 mW of power during normal operation (when it sends an RF and US signal every 250 ms on average). Currently, each Cricket beacon uses a single 9 Volt re-chargeable battery. We plan to use a solar cell with a backup re-chargeable battery in the future.

3.2 Listener API

A part of the software implemented for receiver nodes, called the LocationManager, runs on the host device that has the listener hardware attached to the serial port. The LocationManager listens on the serial port for any data coming from the listener hardware. In our implementation, the *MinMode* listener inference algorithm to analyze distance estimates is also implemented within the LocationManager, since this provides greater flexibility. The listener sends both the location information and the measured distance to the corresponding beacon, to the LocationManager for each valid RF reception.

Asynchronous to the reception of distance estimates and listener computations, applications running on the host device connect to the LocationManager and retrieve current location information using a datagram socket (UDP) interface. In fact, this allows for the possibility of obtaining this information from a remote node elsewhere on the network, which might be useful for some applications. We have not yet taken advantage of this facility in our applications.

3.3 Ultrasound deployment issues

As described in Section 2, ultrasonic interference at the receiver can lead to incorrect distance measurements. It is therefore important to reduce ultrasonic leakage to other locations while trying to provide full coverage to the location served by a Cricket beacon. We achieve this by proper alignment of the ultrasonic transmitters.

Figure 4 shows the radiation pattern of the ultrasonic transmitter used in the Cricket beacons. This is shown in (r, θ) polar coordinates, where r corresponds to the signal strength in dB; and θ corresponds to the offset in degrees from the front of the ultrasonic transmitter. From the radiation pattern, it can be seen that the direction the ultrasound transmitter facing (0°) has the maximum signal strength, while the signal strength drops to 1% (-20 dB) of the maximum value at ±50° away from the 0° direction.

We align the ultrasonic transmitter such that the direction of its peak signal strength is at 45° to the horizontal. The beacon is mounted such that the ultrasonic transmitter faces the location intended to be covered by the beacon. This causes the amount of ultrasonic energy transmitted towards distant locations to be small compared to where

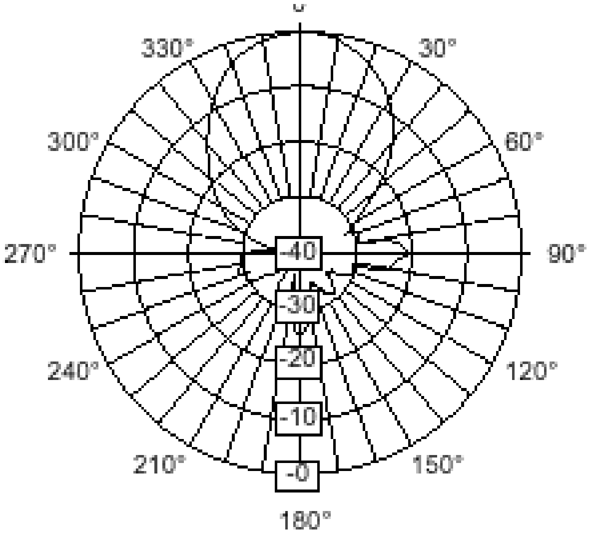


Fig. 4. The radiation pattern of an ultrasonic transmitter.

it is intended. This alignment is easily accomplished by positioning the transmitter at an angle of 45° to the circuit board of the beacon and mounting the board flat on the ceiling or wall of the room, as shown in Figure 5.

We use the velocity of sound in air to measure distances from beacons to receivers.The velocity of sound depends on environmental factors such as the ambient temperature and humidity. Within a building, these properties can exhibit both temporal and spatial variations. Temporal variations occur at different time-scales such as time of day and season of the year. We avoid errors due to such temporal variations using *relative* rather than absolute distances in determining location.

Spatial variations in temperature and humidity due to effects like direct sunlight falling in different sections of a room, the presence of heaters and air conditioners within a room, or the use of humidifiers within a room can affect ultrasound-based distance measurements. We reduce the errors caused by such spatial variations by positioning the beacons and aiming for only coarse-grained (about 10 inches) location information. For instance, supposing that beacons are always kept 2 feet away from a boundary, the distance recorded from a transmitter in an adjoining room has to decrease by ≈ 4 feet for a receiver to mistakenly assume that the adjoining room is closer. This would require a large

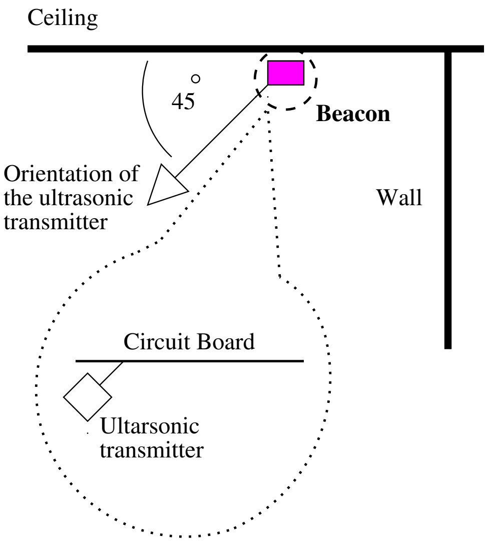


Fig. 5. Correct alignment of a Cricket ultrasonic transmitter.

variation of temperature and humidity along the path; which is highly unlikely in normal circumstances (the temperature coefficient of the velocity of sound in air is 2ft/sec per degree-Celsius).

4 Experiments

We conducted several experiments to investigate the performance of Cricket. The first experiment examines the listener performance near location boundaries, and shows that we can achieve a location granularity of 4 × 4 feet. The second experiment is aimed at investigating the robustness of the system to interference amongst beacons, and the evaluates the performance of the three location inference algorithms presented in Section 2.2 for static listeners. The third experiment examines the performance of the three decoding algorithms when a listener is mobile.

4.1 Boundary performance

Figure 6 shows the setup for this experiment. The aim of this experiment is to investigate the the ability of the listener to detect the boundary, which determines the precision of the system.

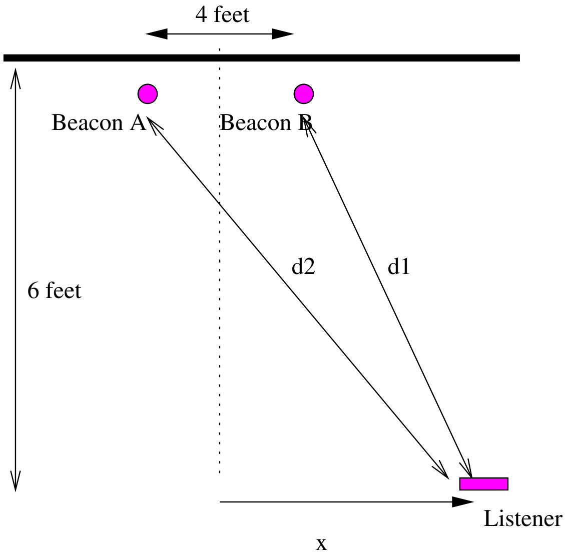


Fig. 6. Setup for experiment 1, evaluating boundary performance.

Two beacons, A and B, advertising different location strings were placed 4 feet apart on the ceiling, giving rise to a virtual boundary in the middle. Distance samples (in the form of ultrasonic pulse propagation time) were taken at 0.5-feet intervals along the x direction as shown in the figure, starting from the center. Figure 7 shows the results of this experiment, plotting the average and the standard deviation of the ultrasonic propagation times from the two beacons as a function of the displacement from the boundary x. This shows that when the listener is more than about 1 foot away from the boundary, the closest beacon can be determined accurately from the estimated distances, thus enabling the listener to determine its location accurately. Furthermore, the difference of the two average distances increases as the listener moves away from the boundary, which causes the probability of making a wrong decision by the listener to decrease as it moves away from the boundary.

This also shows that we can easily achieve a location granularity of 4×4 feet, by placing the beacons in a 4×4 feet grid. Which, effectively divided the region in to 4×4 feet cells. In the future, we plan to carry out more detailed experiments to measure the accuracy of our hardware, and the precision and granularity of the system as the density of beacons increases.

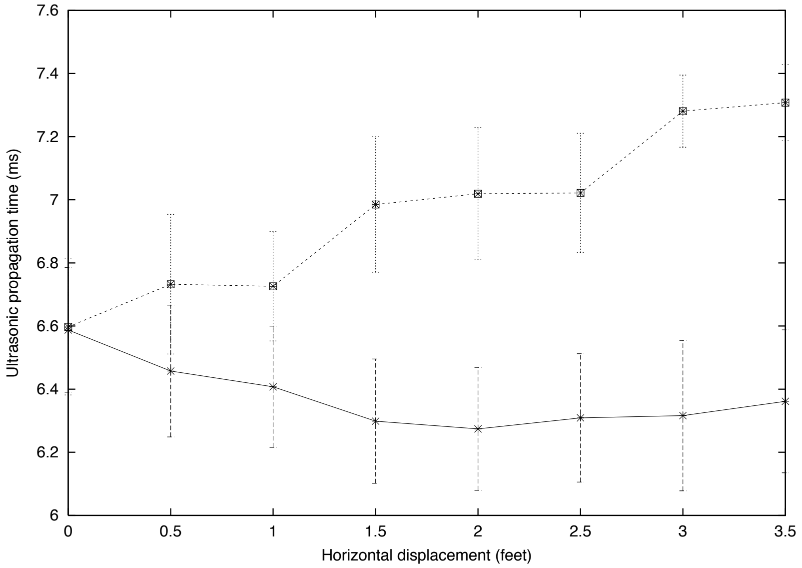


Fig. 7. Average and standard deviation (the errorbars) of ultrasonic propagation time as a function of the horizontal displacement of a listener from the boundary of two beacon regions. When the displacement is over about 1 foot, the errorbars do not overlap.

4.2 Static performance

In the second experiment, we examine the robustness of Cricket against interference amongst nearby beacons. It shows that it is indeed possible to achieve good system performance, despite the absence of any explicit coordination amongst the beacons. We also compare the performance of the three listener inference algorithms presented in Section 2.2.

Figure 8 shows the setup for this experiment. Beacons $B1$ and $B2$ provide location information within room X. Beacons $B3$ and $B4$ provide location information for rooms Y and Z. All these beacons are within the range of each others ultrasonic transmissions. To provide RF interference with no corresponding ultrasonic signals (since the range of RF exceeds that of ultrasound in Cricket), we use beacons $I1$ and $I2$ that have their ultrasonic transmitters disabled.

All the beacons were attached to the ceiling with the ultrasonic transmitters facing their respective spaces as described in Section 2.3. We gathered distance samples at locations $R1$ and $R2$ for a static listener. Observe that $R1$ is closer to the interfering sources $I1$ and $I2$

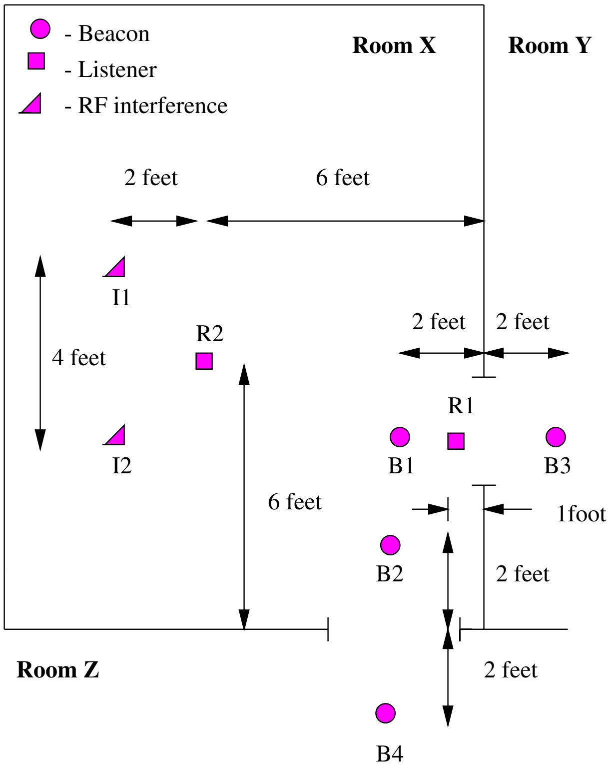


Fig. 8. Setup for experiment 2, evaluating the robustness of Cricket in the presence of interfering beacons.

than to the legitimate beacons for the room, corresponding to the presence of severe RF interference. In contrast, $R2$ is only 1 foot away from the boundary separating the rooms X and Y, showing the performance close to a boundary.

First, we determined the degree of interference caused by $I1$ and $I2$ by collecting 1000 samples of distance estimates at $R1$ and $R2$ and counting the number of values corresponding to each RF source (beacon or interferer). When the listener was at $R1$, somewhat farther from the interfering sources, there were no distance samples corresponding to the interfering RF sources. On the other hand, at $R2$ we received a total of only 7 samples corresponding to both $I1$ and $I2$, despite the fact that $R2$ is closer to $I1$ and $I2$ relative to the legitimate beacons. Table 1 summarizes these results.

Interference Source	I1	I2
Interference at R1	0.0%	0.0%
Interference at R2	0.3%	0.4%

Table 1. Degree of interference at $R1$ and $R2$ caused by $I1$ and $I2$, showing the effectiveness of the randomized beacon transmissions and system parameters.

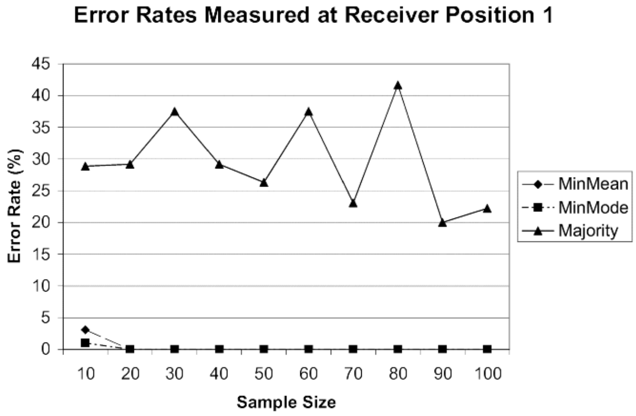


Fig. 9. Error rates at Position 1.

The samples corresponding to $I1$ and $I2$ are due to the incorrect correlation of these RF signals with ultrasonic pulses from other beacons in the vicinity of the listener. However, the randomized transmission schedule together with proper system parameters reduces the occurrence of such interference to a very small fraction of the total. This validates our claims in Section 2.2 and our design.

We now investigate the performance of the three inference algorithms, *Majority*, *MinMean*, and *MinMode*, when the listener is at $R1$ and $R2$. Here, we compute the error rate (in percent) in inferring the location by these three inference algorithms, varying the number of distance samples used for inference. The results, shown in Figure 9 (for position $R1$) and Figure 10 (for position $R2$), demonstrate that both *MinMean* and *MinMode* perform very well even when the sample size is small, even for the case when a listener ($R1$) is close to a boundary.

4.3 Mobile performance

This experiment is aimed at determining the system performance when the listener is mobile. For a mobile listener, being able to obtain accu-

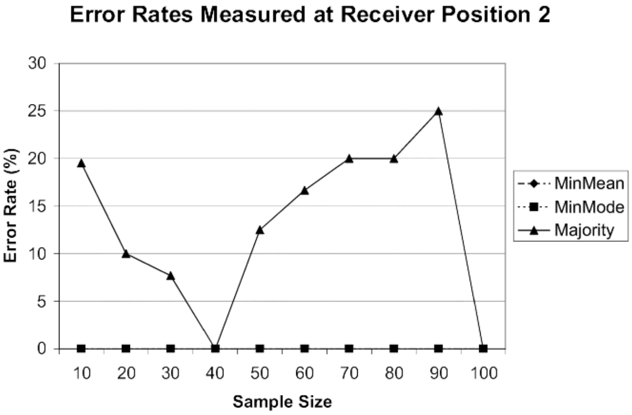


Fig. 10. Error rates at Position 2.

rate location information within a short time is important. Figure 11 shows the configuration of the beacons and the path followed by the mobile user while taking measurements. The listener was moved through each boundary at approximately the same speed each time, emulating a user's typical walking speed in a building. Each time the listener crossed a boundary, a transition event and a timestamp was logged. Once through the boundary, the listener remained stationary for a short period of time to determine how long it takes to stabilize to the correct value, and then the experiment was repeated again through the next boundary. When analyzing the data, we used the logged transition event to determine the user's actual location with respect to the location being reported by the listener. Note that in this experiment, the listener is always located relatively close to the boundaries.

Figure 12 shows the location error-rate at the listener for the experiment. The error-rate is calculated over the time period during which the listener moves around a location, after crossing a boundary. The *MinMode* performs the best among the three inference algorothms. From the results, it is evident that larger time intervals provide better results over smaller intervals, which is not surprising since a larger interval gives the algorithm more samples samples to work with. Another interesting point is that *MinMean* and *MinMode* both perform about the same over small time windows. As the time interval gets smaller the probability that a distance value sample containing only a single value per beacon increases. A small number of samples causes both the mean and the mode to be the roughly the same.

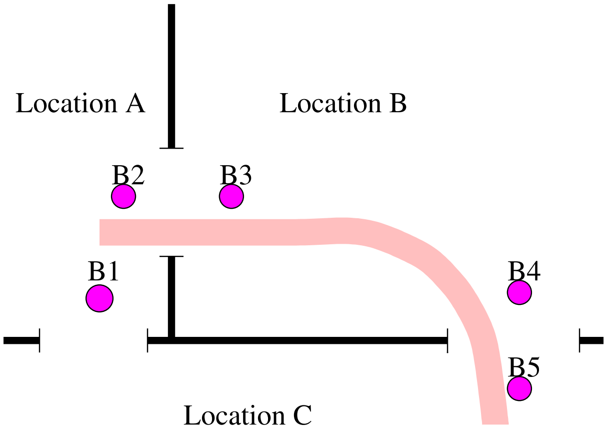


Fig. 11. Setup for experiment 3, evaluating the mobile performance of Cricket.

5 Applications

This section describes how user applications can obtain location information and use it to gain access to nearby services. As mentioned earlier, there are a number of resource discovery systems that can be used along side Cricket. We have implemented several applications using the resource discovery facility provided by the Intentional Naming System (INS), which handles service and device mobility within the naming system [1, 2].

5.1 Using virtual spaces in INS

INS uses the concept of a virtual space (vspace), which is a collection of applications/services that can communicate with each other [14]. Each vspace has a set of name resolvers that resolve name requests for entities in that vspace; each entity is described using an *intentional name*, which is a hierarchical collection of application-defined attributes and values.

The overhead for creating a vspace in INS is small. For our location-dependent applications, we create a vspace for every location of interest (e.g., a room or a floor of a building) and identify it by a string. Each beacon advertises the name of the vspace of the corresponding location, and each listener uses this name to bootstrap into its environment by

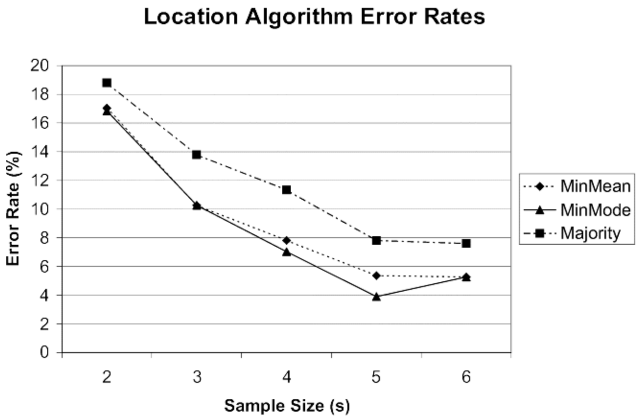


Fig. 12. Error rates for a mobile Cricket listener.

contacting INS and learning about the other existing services in that vspace.

Users and devices can also register their intentional names with the vspace for that location, which enables other entities in that vspace to detect their presence. This way the user can easily determine all the services that are located in their vspace. A user does not necessarily have to be limited to only one vspace at a time, and can select arbitrary services to use. For example, one vspace can correspond to the set of printers in a building while another corresponds to the services located on a specific floor. A user can determine the least loaded printer by querying the printer vspace, or the physically-closest, least-loaded printer by querying the vspace representing the particular floor of the building.

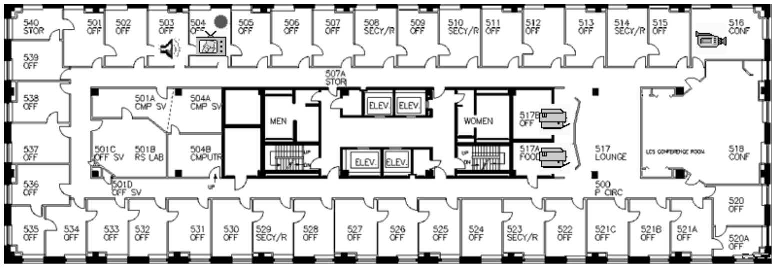


Fig. 13. Floorplan map.

5.2 Floorplan

The Floorplan is an active map navigation utility that uses Cricket and a map server to present a location-dependent “active” map to the user, highlighting her location on it as she moves. It also displays the set of services that are located in the vicinity of the user, which are dynamically updated as the user moves. Floorplan loads map images from the map server, which also provides the values of (x, y) coordinate on the map corresponding to the user’s current vspace position. As the user moves around the building, the listener infers its location and asks the map server to provide the location on the map. Floorplan also learns about various services in the vspace, and contacts those services and downloads a small icon representing each service. These icons are displayed on the map; when the user clicks on an icon, Floorplan uses INS to download a control script or program for the application represented by that icon, and load the controls into a new window so the user can control the application. Figure 13 shows an active map displayed by Floorplan; we see that the user (represented by the dot) is in room 503. It also displays four services it has found in the environment (space) :an MP3 service (represented by the speaker icon) in room 503, a TV service (represented by the TV icon) in room 504, and two printers (represented by the printer icons) in room 517. Using this, a user with no knowledge of her environment or software to control services within it can bootstrap herself with no manual configuration.

6 Related work

There are various solutions available today for device tracking and location discovery. For example, active and passive electromagnetic and optical trackers are sometimes used for tracking and tagging objects. Unfortunately, these tend to be expensive, and the performance of electromagnetic trackers is affected by the presence of metallic objects in the environment. Furthermore, these products do not usually preserve user privacy.

The rest of this section discusses three systems that influenced various aspects of Cricket, and compares their relative benefits and limitations. Table 2 summarizes the following discussion.

6.1 The BAT system

In the BAT system, various objects within the system are tagged by attaching small wireless transmitters. The location of these transmit-

System	Bat	Active Badge	RADAR	*Cricket*
User privacy	No	No	Possible, with user computation	*Yes*
Decentralized	No	No	Centralized RF signal database	*Yes*
Heterogeneity of networks	Yes	Yes	No	*Yes*
Cost	High	High	No extra component cost, but only works with one network	*Low (U.S. $10) component cost*
Ease of deployment	Difficult; requires a matrix of sensors	Difficult; requires a matrix of sensors	RF mapping	*Easy*

Table 2. Qualitative comparison of other location-tracking systems with Cricket.

ters are tracked by the system to build a location database of these objects [19, 11].

The system consists of a collection of mobile or fixed wireless transmitters, a matrix of receiver elements, and a central RF base station. The wireless transmitter consists of an RF transceiver, several ultrasonic transmitters, an FPGA, and a microprocessor, and has a unique ID associated with it. The receiver elements consist of an RF receiver, and an interface for a serial data network. The receiver elements are placed on the ceiling of the building, and are connected together by a serial wire network to form a matrix. This network is also connected to a computer, which does all the data analysis for tracking the transmitters.

The RF base station orchestrates the activity of transmitters by periodically broadcasting messages addressed to each of them in turn. A transmitter, upon hearing a message addressed to it, sends out an ultrasound pulse. The receiver elements, which *also* receive the initial RF signal from the base station, determine the time interval between the receipt of the RF signal and the receipt of the corresponding ultrasonic signal, from which they estimate the distance to the transmitter. These distances are then sent to the computer which performs the data analysis. By collecting enough distance readings, it is possible to determine the location of the transmitter with an accuracy of a few centimeters, and these are keyed by transmitter address and stored in the location database.

Bat derives its accuracy from a tightly controlled and centralized architecture that tracks users and objects. In contrast, Cricket is highly decentralized and there is no central control of any aspect of the system, which preserves user privacy, is simpler, and reduces management cost. The differences in design goals between Bat and Cricket lead to radical differences in architecture, although the use of ultrasound and RF is common to both systems.

6.2 The Active Badge system

The Active Badge[1] system was a predecessor to the Bat system, and tracks objects in an environment to store in a centralized location database [18]. Objects are tracked by attaching a badge, which periodically transmits its unique ID using infrared transmitters. Fixed infrared receivers pick up this information and relay it over a wired network. The walls of the room act as a natural boundary to infrared signals, thus enabling a receiver to identify badges within its room. A particular badge is associated with the fixed location of the receiver that hears it.

Like the Bat system, the object tracking nature of Active Badge system may introduce privacy concerns among users. Infrared also suffers from dead-spots, which Cricket and Bat are relatively immune to because they use ultrasound.

6.3 RADAR

The RADAR system implements a location service utilizing the information obtained from an already existing RF data network [3]. It uses the RF signal strength as an indicator of the distance between a transmitter and a receiver. This distance information is then used to locate a user by triangulation.

During an off-line phase; the system builds a data base of RF signal strength at a set of fixed receivers, for known transmitter positions. During the normal operation, the RF signal strength of a transmitter as measured by the set of fixed receivers, is sent to a central computer, which examines the signal-strength database to obtain the best fit for the current transmitter position.

---

[1] Active Badge is a registered trademark of Ing. C. Olivetti & C., S.p.A.

In contrast to these three projects, Cricket has different design goals: it has to handle network heterogeneity and privacy concerns, and have low management cost. It eliminates all central repositories of control or information, leading to an autonomously administered building-wide service via delegation. The beacons advertising location information are self-contained and do not need any infrastructure for communication amongst themselves. Together with the use of inexpensive, off-the-shelf hardware, this makes deployment easy and cost-effective. In summary, Cricket is a *location-support* service, not a location-tracking one.

7 Conclusion

In this paper, we presented the design, implementation, and evaluation of Cricket, a location-support system for mobile, location-dependent applications. Cricket is the result of five design goals: user privacy, decentralized administration, network heterogeneity, low cost, and portion-of-a-room granularity. Its innovative aspects include the use of beacons with combined RF and ultrasound signals in a decentralized, uncoordinated architecture. It uses independent, randomized transmission schedules for its beacons and a receiver decoding algorithm that uses the minimum of modes from different beacons to compute a maximum likelihood estimate of location. We described some deployment considerations based on our preliminary experience with Cricket and presented a comparison with three important past systems, showing that our design goals led to a different design and properties from past systems.

We are encouraged by our experience with Cricket to date and the ease with which location-dependent applications like active map and location-based services can be implemented. We have demonstrated that it is possible to implement a location-support system that maintains user privacy and has no centralized control.

8 Acknowledgements

We thank William Adjie-Winoto, Dave Andersen, Steve Bauer, Dina Katabi, Jinyang Li, Rodrigo Rodrigues, Xiaowei Yang, and the ACM MOBICOM reviewers for useful comments and suggestions that improved the quality of this paper.

References

1. W. Adjie-Winoto. A Self-Configuring Resolver Architecture for Resource Discovery and Routing in Device Networks. Master's thesis, Massachusetts Institute of Technology, May 2000.
2. Adjie-Winoto, W., Schwartz, E. and Balakrishnan, H. and Lilley, J. The design and implementation of an intentional naming system. In *Proc. ACM Symposium on Operating Systems Principles*, pages 186–201, Kiawah Island, SC, December 1999.
3. P. Bahl and V. Padmanabhan. RADAR: An In-Building RF-based User Location and Tracking System. In *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.
4. G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. An Application Model for Pervasive Computing. In *Proc. 6th ACM MOBICOM Conf.*, Boston, MA, August 2000.
5. N. Bulusu, J. Heidemann, and D. Estrin. GPS-less Low Cost Outdoor Localization For Very Small Devices. Technical Report 00-729, Computer Science Department, University of Southern California, April 2000.
6. A. Chakraborty. A Distributed Architecture for Mobile, Location-Dependent Applications. Master's thesis, Massachusetts Institute of Technology, May 2000.
7. S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Proc. 5th ACM MOBICOM Conf.*, pages 24–35, Seattle, WA, August 1999.
8. M. Dertouzos. The Future of Computing. *Scientific American*, August 1999. Available from `http://www.sciam.com/1999/0899issue/0899dertouzos.html`.
9. Federal Communications Commision. *Understanding the FCC regulations for low-power, non-licensed transmitters*, February 1996.
10. I. Getting. The Global Positioning System. *IEEE Spectrum*, 30(12):36–47, December 1993.
11. A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. In *Proc. 5th ACM MOBICOM Conf.*, Seattle, WA, August 1999.
12. Hertz Services: Hertz NeverLost. `http://www.hertz.com/serv/us/prod_lost.html`, 2000.
13. Jini (TM). `http://java.sun.com/products/jini/`, 1998.
14. J. Lilley. Scalability in an Intentional Naming System. Master's thesis, Massachusetts Institute of Technology, May 2000.
15. Oxygen home page. `http://oxygen.lcs.mit.edu/`.
16. Ultrasonics and robotics. `http://www.seattlerobotics.org/encoder/may97/sonar2.html`, May 1997.
17. J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. *Service Location Protocol*, June 1997. RFC 2165 (`http://www.ietf.org/rfc/rfc2165.txt`).

18. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.
19. A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office. *IEEE Personal Comm.*, 4(5):42–47, October 1997.
20. M. Weiser. The computer for the 21st century. *Scientific American*, September 1991.
21. X-10 home page. `http://www.x10.com/homepage.htm`.

Proc. of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001), Rome, Italy, July 2001.

The Cricket Compass for Context-Aware Mobile Applications

Nissanka Priyantha, Allen Miu, Hari Balakrishnan and Seth Teller

MIT Laboratory for Computer Science
{`bodhi, aklmiu, hari, seth`}`@lcs.mit.edu`

Abstract. The ability to determine the orientation of a device is of fundamental importance in context-aware and location-dependent mobile computing. By analogy to a traditional compass, knowledge of orientation through the *Cricket compass* attached to a mobile device enhances various applications, including efficient way-finding and navigation, directional service discovery, and "augmented-reality" displays. Our compass infrastructure enhances the spatial inference capability of the Cricket indoor location system [20], and enables new pervasive computing applications.

Using fixed active beacons and carefully placed passive ultrasonic sensors, we show how to estimate the orientation of a mobile device to within a few degrees, using precise, sub-centimeter differences in distance estimates from a beacon to each sensor on the compass. Then, given a set of fixed, active position beacons whose locations are known, we describe an algorithm that combines several carrier arrival times to produce a robust estimate of the rigid orientation of the mobile compass.

The hardware of the Cricket compass is small enough to be integrated with a handheld mobile device. It includes five passive ultrasonic receivers, each 0.8cm in diameter, arrayed in a "V" shape a few centimeters across. Cricket beacons deployed throughout a building broadcast coupled 418MHz RF packet data and a 40KHz ultrasound carrier, which are processed by the compass software to obtain differential distance and position estimates. Our experimental results show that our prototype implementation can determine compass orientation to within 3 degrees when the true angle lies between ±30 degrees, and to within 5 degrees when the true angle lies between ±40 degrees, with respect to a fixed beacon.

1 Introduction

Context-aware applications, which adapt their behavior to environmental context such as physical location, are an important class of applications in emerging pervasive computing environments [17]. Examples include *location-aware* applications that enable users to discover resources in their physical proximity [14, 20], active maps that automatically change as a user moves [22], and applications whose user interfaces adapt to the user's location. A significant amount of previous work has focused on providing device position capability indoors, including the Active Badge [26], Bat [14], RADAR [5], and Cricket [20] systems.

An important aspect of context, which is related to physical position, is the *orientation* of a device (or user) with respect to one or more landmarks in a region. A pervasive computing application can benefit from knowing this information, for instance by providing the ability to adapt a user interface to the direction in which a user is standing or pointing. Our first motivating application is called the *Wayfinder*. We envision this application to run on a handheld computer and help sighted or blind people navigate toward a destination in an unfamiliar setting. For example, the Wayfinder might lead a visitor from the entry lobby of a building to the office of the person hosting the visitor, or to a seminar room. The Wayfinder gives incremental directions to the user on dynamically retrieved ("active") maps [22, 20], using the user's position and orientation with respect to a fixed set of wireless beacons placed throughout the building. The second motivating application is called the *Viewfinder*. The user can point it in any direction, and specify a "sweep angle" and maximum distance. Using an active map integrated with a resource discovery system (e.g., the Intentional Naming System, INS [1]), the Viewfinder then retrieves and displays a representation of the set of devices and services lying inside the sector of interest specified by the user and allows the user to interact with these services via the representation on the map. A third motivating application is in the design of "augmented-reality" displays, where the user's view of the environment is overlaid with information about other objects present within that environment, and adapts to the direction that the user is looking toward [4, 25].

The underlying capability required for these applications is akin to a "software compass," which, endowed with a semantic map of its context and accurate knowledge of its own position and orientation, can inform the user of interesting resources and how to get to those resources. This paper describes the design and implementation of the *Cricket compass system*, consisting of a set of active beacons, passive hardware sensors, and associated software algorithms. This system enables a robust software compass capability for a handheld device moving about inside a building.

The operating environment in the Cricket architecture is instrumented with active beacons, each of which broadcasts its own known position over an RF channel together with an ultrasonic pulse [20]. One RF receiver and several passive ultrasonic position receivers are precisely placed on a compass board. Software running on-board uses the *differentials* in distances reported by the ultrasonic receivers to infer the orientation (or "heading") of the device. The Cricket compass reports position and orientation indoors for a handheld, mobile device, and informs an application running on the device of the position and orientation in a local coordinate system established by the fixed set of beacons.

The first challenge in deriving *orientation* for a small device arises from the need for very accurate differential distance estimates: estimating orientation to within a few degrees of the correct value requires differential distance estimates to be of sub-centimeter accuracy, which is at least an order of magnitude smaller than the currently best available linear distance estimation technologies. We show how to do this using multiple carefully placed receivers. The second challenge arises due to variation in the speed of sound due to temperature and humidity, which affects the accuracy of position estimates. Rather than explicitly measuring this parameter with climate sensors, we calculate it directly from observed propagation times.

The Cricket compass system proposed in this paper addresses several problems with existing methods for orientation estimation. A traditional magnetic compass can estimate orientation, but exhibits enormous errors when near magnetic or time-varying electric fields, both of which are rather common in most modern buildings with computers and other equipment. Orientation can be inferred from a moving position sensor, but this usually requires large or fast user motions, which is undesirable in several applications. Active sensors on user devices typically lead to systems that track users [14, 26], which suffer from potential compromises to user privacy [19, 20]. In contrast, Cricket requires a small number of beacons at known positions in each room to instrument a building, but enables location and orientation for a passive handheld device without requiring any user motion. We have built several prototype beacons and a receiver compass configuration, and report experimental data that show that our software compass correctly estimates orientation to within a few degrees. We also describe a Viewfinder application developed using this capability. The Cricket

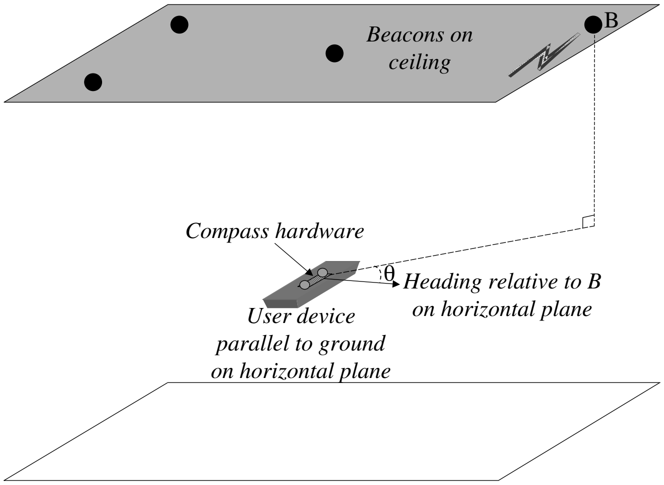


Fig. 1. Setup of beacons on the ceiling of a room and a user device with attached compass hardware. The Cricket compass solves the problem of obtaining the precise position and orientation of the user device relative to a coordinate system defined by the beacons.

system is being used in MIT's Project Oxygen in a variety of pervasive computing scenarios [17].

The rest of this paper is organized as follows. Section 2 details the design of the Cricket compass, describing its theory of operation, differential distance estimation, and coordinate determination algorithms. Section 3 discusses our implementation and Section 4 presents experimental results and an error analysis. Section 5 discusses some improvements based on our experimental results. Section 6 describes the Viewfinder application. We compare our system to previous work in Section 7 and conclude in Section 8.

2 Design of the Cricket Compass

Figure 1 shows a user device with attached compass hardware in a room with beacons placed on the ceiling. When the device is held parallel to the horizontal plane, θ is the angle formed by the heading direction shown, with the point where the perpendicular from beacon B intersects the horizontal plane. We are interested in precisely estimating θ.

The basic idea is to use one RF receiver to receive coordinate information from the beacons, and multiple, carefully placed, ultrasonic receivers on the compass attached to the device to obtain the *differential* distance estimates of a beacon to each ultrasonic receiver. θ is a function of the differential distance of the linear distance of the compass from the beacon, and of the height of the beacon (ceiling) above the plane of the compass. We obtain per-beacon linear distance estimates by differencing the arrival times of coupled RF and ultrasonic signals sent from each beacon [20]. To obtain the height of the beacon from the compass, we estimate the position coordinates of the compass from the position coordinates disseminated by multiple nearby beacons.

The rest of this section describes how this idea can be realized in practice. We start by describing how directional information can be obtained using differences in distance between a beacon and different receivers. We describe a technique to achieve the required precision of differential distance estimates, using differential phase information of the ultrasonic waves reaching the receivers. Finally, we show how to obtain accurate position coordinate information without explicit knowledge of the speed of sound, compensating for its variation with physical conditions.

2.1 Theory of operation

Figure 2 shows a beacon B, and a compass with two ultrasonic receivers, R_1 and R_2, which are located at a distance L apart from each other. The angle of rotation of the compass, θ, with respect to the beacon B, is related to the difference in distances d_1 and d_2, where d_1 and d_2 are the distances of receivers R_1 and R_2 from B. The vertical and horizontal distances from the center of the compass to B are denoted by z and x, respectively.

Figure 3 shows the beacon B from Figure 2 projected on to the horizontal plane along which the compass is aligned. In this figure, x_1 and x_2 are the projections of distances d_1 and d_2 on to the horizontal plane. We assume that the compass is held parallel to the horizontal plane.

From Figure 2:

$$x_1^2 = d_1^2 - z^2 \tag{1}$$

$$x_2^2 = d_2^2 - z^2 \tag{2}$$

$$x = \sqrt{\bar{d}^2 - z^2}$$

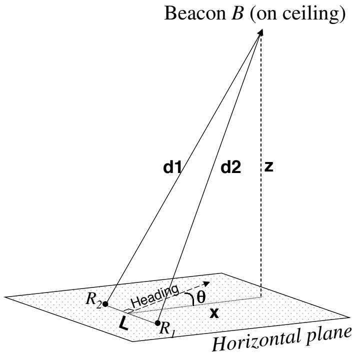


Fig. 2. Determining the angle of orientation along the horizontal plane, θ, using distance estimates. Observe that the heading is perpendicular to the line joining the ultrasonic compass receivers, R_1 and R_2, which are placed at a distance L from each other.

where $\bar{d} \approx \frac{d_1+d_2}{2}$ when $d_1, d_2 \gg L$.

From Figure 3:

$$x_1^2 = (\frac{L}{2}\cos\theta)^2 + (x - \frac{L}{2}\sin\theta)^2$$

and

$$x_2^2 = (\frac{L}{2}\cos\theta)^2 + (x + \frac{L}{2}\sin\theta)^2$$

$$\Rightarrow x_2^2 - x_1^2 = 2Lx\sin\theta$$

Substituting for x_1^2 and x_2^2 from Equations (1) and (2), we get:

$$\sin\theta = \frac{d_2 + d_1}{2Lx} \cdot (d_2 - d_1) \tag{3}$$

This may be rewritten as:

$$\sin\theta = \frac{d_2 - d_1}{L\sqrt{1 - (\frac{z}{\bar{d}})^2}} \tag{4}$$

Equation (4) implies that it suffices to estimate two quantities in order to determine the orientation of the compass with respect to a beacon: (i) $(d_2 - d_1)$, the difference in distances of the two receivers

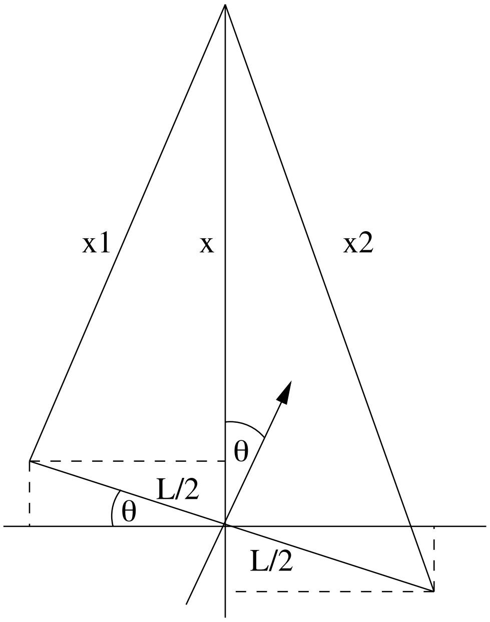


Fig. 3. A rotated compass leads to a difference in distances between the beacon and each of the receivers. This figure is the result of projecting the beacon onto the horizontal plane of the compass.

from the beacon, and (ii) $z/\bar{d}$, the ratio of the height of the beacon from the horizontal plane on which the compass is placed to the distance of the beacon from the center of the compass. In practice, however, no measurements are perfect. Our goal is to estimate each of these quantities with high precision, so as to produce a sufficiently accurate estimate of θ.

One way of precisely estimating $(d_2 - d_1)$ would be to precisely measure d_1 and d_2 separately, but that is easier said than done. Consider, for example, a situation where $L = 5$cm, and $\theta = 10°$, with a beacon at a distance of 2 meters and a height of 1 meter from the receivers. From Equation (4), the value of $(d_2 - d_1)$ in this case is only ≈ 0.6cm, which is about an order of magnitude smaller than what current technologies can achieve in terms of linear distance estimates [14, 20][1]. Since our goal is to devise a compass with physically small dimensions, comparable in

---

[1] The worst-case error in $(d_2 - d_1)$ is equal to the sum of the errors in d_1 and d_2.

size to handheld PDAs, and still achieve high directional accuracy, we need an alternative method to estimate this differential distance.

Our solution to this problem tracks the *phase difference* between the ultrasonic signals at two different receivers and processes this information. We find that this approach allows us to obtain differential distance estimates with sub-centimeter accuracy. This is described in Section 2.2.

The second quantity, $z/\bar{d}$, is estimated by determining the (x, y, z) coordinates of the compass with respect to the plane formed by the beacons (the xy plane). We do this by placing multiple beacons in a room and estimating the time it takes for the ultrasonic signal to propagate between them and the compass. However, because the speed of sound varies with ambient temperature and humidity, we must estimate this quantity as well. This is described in Section 2.3.

2.2 Estimating differential distance

Consider two ultrasonic receivers R_1 and R_2 located a distance L apart, as shown in Figure 4. Let d_1 and d_2 be the distances to receivers R_1 and R_2 from beacon B. Let $\delta d = d_1 - d_2$ and let W_1 and W_2 be the ultrasonic waveforms received by R_1 and R_2 from B. The phase difference between the waveforms at the two receivers, ϕ, depends on the difference in distances traversed from B to the receivers by the ultrasonic signal and the wavelength λ of the signal, and may be expressed as:

$$\phi = \frac{(\delta d)}{\lambda} \cdot 2\pi \tag{5}$$

We call this the *actual* phase difference between the two signals and denote it by ϕ.

Because it is difficult to correctly determine the start of periodic waveforms, we can only obtain estimates for a waveform's phase in the range $(-\pi, \pi)$ from repeated low-to-high transitions of the signal. Unfortunately, a given observed phase difference between two waveforms, α, can correspond to an infinite number of actual phase differences, all separated by 2π. This in turn leads to multiple possibilities for δd.

One way to solve this problem is to observe from Equation (5) that as long as $\delta d < \lambda/2$, $\phi = \alpha$, and there is no ambiguity. Since d_1, d_2, and L are three sides of a triangle, $L \geq |d_1 - d_2| = |\delta d|$, and we can therefore place the receivers at a distance $L < \lambda/2$ to unambiguously determine ϕ and therefore uniquely estimate $(d_1 - d_2)$.

For a 40 KHz ultrasonic waveform at a temperature of 25°C and 50% humidity, $\lambda/2 = 4.35$ mm. This is smaller than the size of most

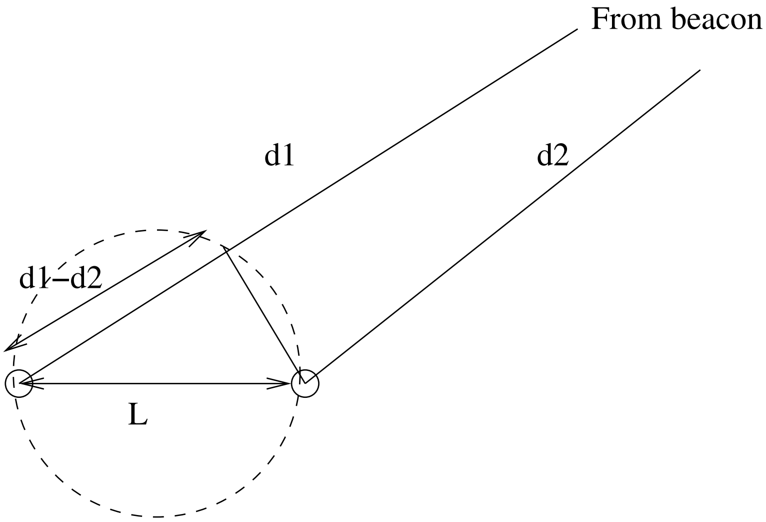


Fig. 4. Receivers R_1 and R_2 can measure the differential distance from a far-away beacon.

available ultrasonic signal receivers, which are typically on the order of about 1 cm. Lowering the carrier frequency is not an option, since this would make it audible to humans. We therefore need a nice general method to place receivers to unambiguously determine the phase difference.

One way of tackling this is to carefully place *three* receivers along a line, as shown in Figure 5, and use a *pair* of observed phase differences to estimate an actual difference. The intuition is that if the two inter-receiver distances, L_{12} and L_{23} are chosen carefully, then the actual phase difference between receivers 1 and 2 (say) can be disambiguated by using the phase difference between receivers 2 and 3, since the two phase differences are not independent.

Let ϕ_{12} and ϕ_{23} be the actual phase differences of a beacon's waveform between receivers 1 and 2 and receivers 2 and 3, respectively. Then,

$$\phi_{ij} = 2n_{ij}\pi + \alpha_{ij}$$

for each pair of receivers (i, j), where n_{ij} are integers and $-\pi < \alpha_{ij} \leq \pi$. Because the actual phase difference between two receivers is proportional to the distance traversed by the signal from the beacon to each of the receivers, $\phi_{23}/\phi_{12} = (d_2 - d_3)/(d_1 - d_2) \approx L_{23}/L_{12}$ when $d_i \gg L_{ij}$. This is shown in Figure 5.

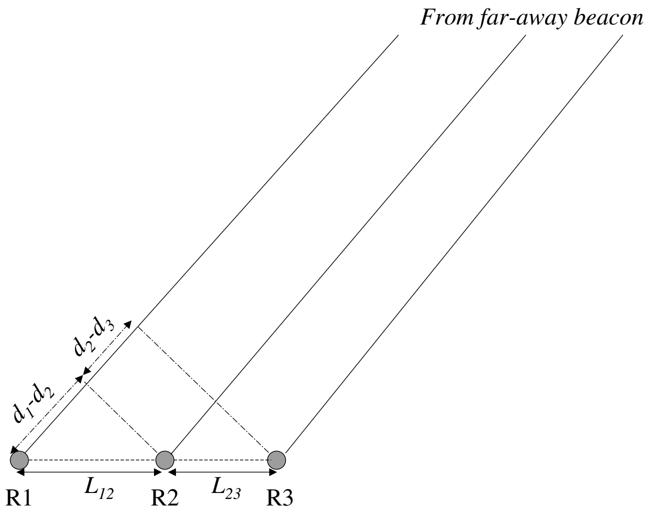


Fig. 5. Using three receivers to measure $(d_1 - d_2)$.

What we will show is that it is possible to pick L_{12} and L_{23} such that one can use two sets of observed phase differences α_{12}, α_{23} to *unambiguously* estimate the actual phase difference ϕ_{12}. In particular, we show the following result:

If L_{12} and L_{23} are relatively prime multiples of $\lambda/2$, then it is possible to use α_{12} and α_{23} to unambiguously obtain the actual phase differences ϕ_{12} and ϕ_{23}.

We argue this by contradiction. Suppose in fact there are two possible actual phase differences corresponding to a given observed phase difference for each receiver. For pair (i, j), call these differences ϕ'_{ij} and ϕ''_{ij}. Then, the following sets of equations hold:

$$\phi'_{ij} = 2n'_{ij}\pi + \alpha_{ij}$$

$$\phi''_{ij} = 2n''_{ij}\pi + \alpha_{ij}$$

Since each observed ϕ_{12} is related to the corresponding ϕ_{23} by the ratio L_{23}/L_{12}, the above equations can be rewritten as:

$$2n'_{23}\pi + \alpha_{23} = (L_{23}/L_{12})(2n'_{12} + \alpha_{12}) \tag{6}$$

$$2n''_{23}\pi + \alpha_{23} = (L_{23}/L_{12})(2n''_{12} + \alpha_{12}) \tag{7}$$

Subtracting Equation (7) from Equation (6) and rearranging, we get:

$$L_{12}(n'_{23} - n''_{23}) = L_{23}(n'_{12} - n''_{12}) \tag{8}$$

Let us express L_{ij} as $l_{ij}\lambda/2$, which expresses the separation between receivers as an integral multiple of $\lambda/2$. Equation (8) is then equivalent to:

$$l_{12}(n'_{23} - n''_{23}) = l_{23}(n'_{12} - n''_{12}) \tag{9}$$

where each of the l_{ij} and n_{ij} are integers.

Notice that $|n_{ij}|\lambda \leq \delta d$, the separation in distance between the carrier waveforms at receiver i and receiver j, and $\delta d \leq L_{ij} = l_{ij}\lambda/2$, for each pair $(i,j) = (1,2),(2,3)$. This means that $|(n'_{ij} - n''_{ij})\lambda| < 2L_{ij} = l_{ij}\lambda$. (In fact, $|(n'_{ij} - n''_{ij})\lambda|$ may be *equal to* $2L_{ij}$, but only if the beacon lies on the same horizontal plane as the compass. This situation is unlikely in practice, and detectable if it does occur.) Therefore, $|n'_{ij} - n''_{ij}| < l_{ij}$. Thus, if Equation (9) is to be satisfied, l_{12} and l_{23} *cannot* be relatively prime.

Hence, it is possible to unambiguously derive an actual phase difference (ϕ_{ij}) in the range of $[0, L_{ij}]$ from an observed one (α_{ij}) by picking L_{12} and L_{23} to be relatively prime integral multiples of $\lambda/2$. For example, we can pick $L_{12} = 2\lambda$ and $L_{23} = 1.5\lambda$. Thus, knowing ϕ, we get the exact δd needed for estimating θ in Equation (4).

Disambiguating $\boldsymbol{\theta}$ Using Equation (4) and the techniques discussed thus far, we can determine $\sin\theta$ between the compass and a particular beacon B. But as Figure 6 shows, in general, there are two locations B_1, B_2 for a beacon B that result in the same θ at the compass. This is due to symmetry of the system about the line X–X. An analytical way of understanding this is to observe that there are two values of θ in the range $[0, 2\pi)$ for a given value of $\sin\theta$. This ambiguity in the location of the beacon prevents us from determining a unique value for the heading.

We solve this by using *two* sets of non-collinear receiver-triplets to break the symmetry. We place the two sets of receiver-triplets perpendicular to each other as shown in Figure 7, and there can be now be only one position for the beacon B. We are now given an angle θ_1 relative to X-X and θ_2 relative to Y-Y, which means that $\sin\theta_1$ and $\sin\theta_2$ are known. It is easy to see that there can only be a unique solution for this configuration. These two perpendicular sets of receiver-triplets are configured using *five* receivers on the compass, as shown in Figure 8.

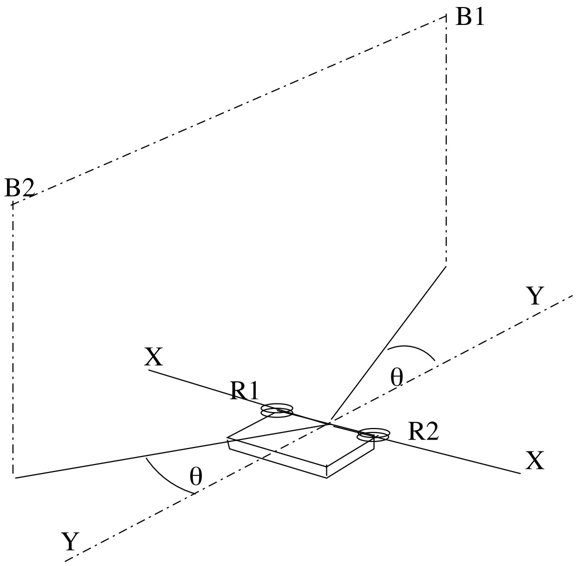


Fig. 6. θ is ambiguous—the beacon can be at either $B1$ or $B2$.

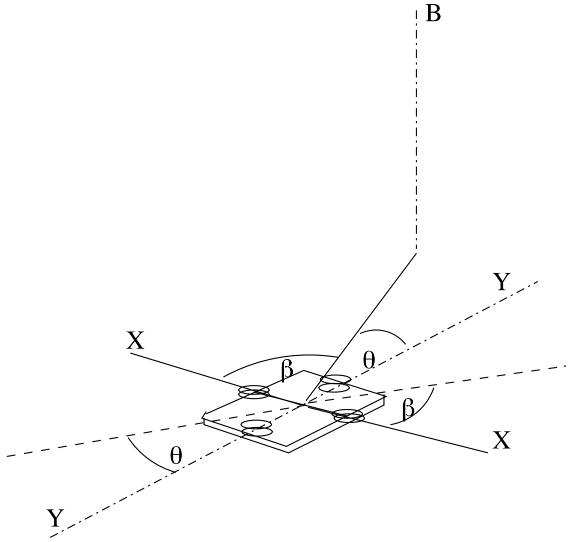


Fig. 7. Two sets of receivers can break the symmetry. One set of receiver triplets lies on the X-X line and the second set lies on the Y-Y line.

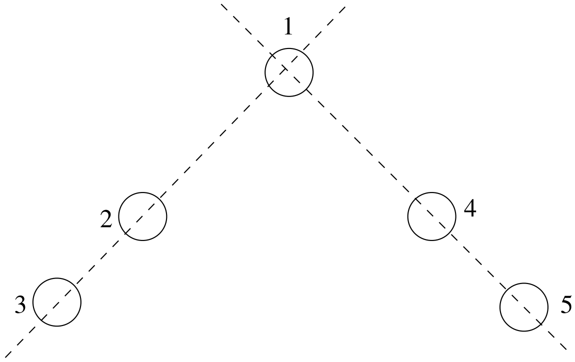


Fig. 8. Five receivers on a compass forming two perpendicular receiver-triplets, which are used to unambiguously infer the heading with respect to a beacon.

2.3 Determining compass coordinates

In the previous discussion we assumed that the receiver can determine the ratio of height to distance, $z/\bar{d}$, to a particular beacon. To enable the receiver to gather this information, we implement a coordinate system using a number of active beacons instrumented with known positions within the space. The compass determines its mean position as an (x,y,z) tuple by listening to beacon transmissions. This mechanism also enables us to determine the speed of sound in the vicinity of the compass.

Both the Bat and the Cricket systems use a combination of RF and ultrasound signals to measure distances, using the relative speeds between these two signals. However, to determine the distance accurately, it is necessary to know the speeds of both signals. The speed of RF is essentially infinite in our setting, but the speed of ultrasound depends on environmental factors such as temperature and humidity. The Bat system compensates for this variation by measuring environmental factors. The Cricket system is robust against such variation by virtue of its dependence only on relative distances.

We present a technique that enables us to determine the position in terms of (x,y,z) coordinates using 4 beacons without knowledge of the speed of sound or requiring additional environmental sensors. We use the measured propagation time $\hat{t_i}$ to each beacon, which is *proportional* to the actual distance d_i.

We implement a coordinate system within the room assuming the ceiling to be the x-y plane and z to be positive inside the room (down-

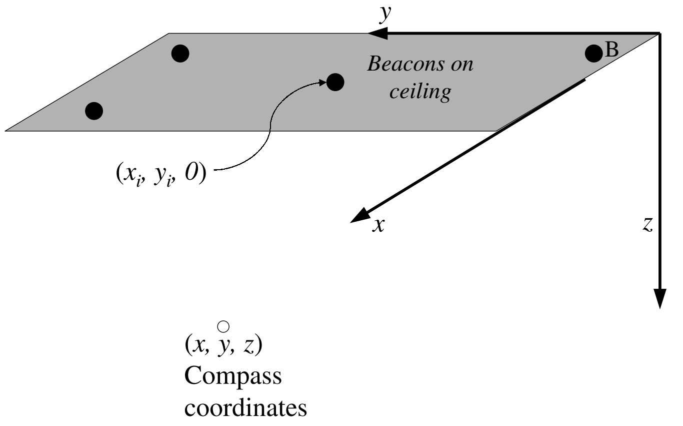


Fig. 9. The coordinate system used in Cricket; the beacons are configured with their coordinates and disseminate this information on the RF channel.

wards), as shown in Figure 9. Consider four beacons B_0, B_1, B_2, and B_3 attached to the ceiling of a room. Each beacon B_i broadcasts its coordinates $(x_i, y_i, 0)$ on the RF channel, which is sensed by the receivers on the compass. At the same time, it also broadcasts an ultrasonic pulse. A receiver, which is at an unknown coordinate (x,y,z), measures the time difference $\hat{t}_i$ between the arrival of these two signals to beacon B_i. The actual distance from the receiver to B_i is therefore equal to $v\hat{t}_i$, where v is the (unknown) speed of sound.

We can then write the following family of four equations for the unknowns (x, y, z, v). Recall that we are interested in the value of $z/\bar{d} = z/\sqrt{x^2 + y^2 + z^2}$.

$$(x - x_i)^2 + (y - y_i)^2 + z^2 = v^2 \hat{t}_i^2, \quad 0 \leq i \leq 3 \tag{10}$$

We can eliminate z^2 from these equations by subtracting each equation from the previous one, to obtain the following three *linear* equations in the three variables, x, y and v^2:

$$A \times \begin{bmatrix} x \\ y \\ v^2 \end{bmatrix} = \begin{bmatrix} x_1^2 - x_0^2 + y_1^2 - y_0^2 \\ x_2^2 - x_1^2 + y_2^2 - y_1^2 \\ x_3^2 - x_2^2 + y_3^2 - y_2^2 \end{bmatrix} \tag{11}$$

where the matrix A is given by

$$A = \begin{bmatrix} 2(x_1 - x_0) \; 2(y_1 - y_0) \; (\hat{t}_1^2 - \hat{t}_0^2) \\ 2(x_2 - x_1) \; 2(y_2 - y_1) \; (\hat{t}_2^2 - \hat{t}_1^2) \\ 2(x_3 - x_2) \; 2(y_3 - y_2) \; (\hat{t}_3^2 - \hat{t}_2^2) \end{bmatrix}$$

If the determinant of A is non-zero, then Equation (11) can be solved to determine unique values for x, y, and v^2. Substituting these values into Equation (10) then yields a value for z^2, whose positive square root yields z. Furthermore, we can use this estimate of v, to further improve the accuracy of the $d_2 - d_1$ estimation by using a better estimate of the wavelength of the ultrasonic carrier.

However, the ability to determine x, y, z, and v^2 uniquely from above set of equations depends on the values (x_i, y_i) for $0 \leq i \leq 3$. Specifically, one can show that if the beacons B_0,B_1,B_2, and B_3 are placed such that they do not all lie on the same straight line or circle, then there is always a unique solution to the above set of equations. In practice, this placement can be done easily by placing 4 beacons on the corners of a rectangle and then by moving one of the beacons some distance along the diagonal of the rectangle. An example placement of the beacons is shown in Figure 9.

Superficially, the equations above are similar to those used by GPS receivers to determine receiver position. In GPS, the beacons are satellites with precise clocks; latency from satellite to receiver is non-negligible; the propagation velocity is known (to first order) as the speed of light; and a system of equations is solved at the receiver to recover the receiver's absolute position and time [15]. In contrast, in Cricket, propagation time (for RF) is negligible; the beacons have no clocks; and the propagation speed (for ultrasound) is unknown. Our system also solves for four unknowns, three of position and one for the speed of sound in the local medium.

3 Implementation

We have implemented prototypes of the beacon and compass hardware described in Section 2. Each beacon is configured with its position in a coordinate system, which it broadcasts on a 418 MHz RF channel. Concurrent with each periodic RF broadcast, it sends a 500 μs ultrasonic pulse at 40 KHz, which are received at the compass ultrasonic receivers. Each beacon and compass has an on-board PIC microcontroller that implements the communication protocol and processes information. The rest of this section describes the details of the communication protocol between the beacons and compass, and how the compass processes the observed differential distance estimates to deduce the actual differential distances.

3.1 Protocol details

The beacons in Cricket operate in an autonomous manner, without any centralized control of when they transmit information [20]. To reduce inter-beacon interference at the receivers, each beacon senses the RF carrier before transmitting a locally unique ID and its known position coordinates. In addition, each subsequent transmission is sent at a uniformly chosen random time after the previous one. In our implementation, the average amount of time between successive transmissions is 250 ms. The packet format of the beacon includes information about the geographic space (e.g., an intentional name for resource discovery in INS [1], a URL as in CoolTown [8], etc.). Each packet is protected using a block-parity code. The compass detects collision on the RF channel and discards samples that do not pass a block-parity check, which helps it disambiguate between potentially interleaved RF/ultrasound combinations sent of separate beacons.

The processing of ultrasonic signals is more involved. The compass hardware does analog-to-digital oversampling to detect low-to-high transitions from each ultrasonic receiver. In addition to processing RF information, the on-board PIC microcontroller handles the ultrasonic signals received by the several ultrasonic receivers on the compass to obtain phase difference estimates, and passes these to the software running on the attached device.

This software processes the raw data to obtain observed differential distance estimates, and then convert them to actual differential distance estimates. It also infers the coordinates of the compass relative to the coordinate system defined by the beacons, and computes the orientation unit vector in that system. It calculates the angle relative to each beacon and uses the smallest angle to derive the orientation vector. The reason for this will be clear from Section 4, which shows that the accuracy of our system worsens at large angles (greater than about 45 degrees). This also means that the system works best when it finds at least one beacon at an angle smaller than 45 degrees—since there are at least four beacons per space on each ceiling of interest, it is relatively straightforward to place, and find, at least one beacon in standard rectangular rooms.

3.2 Differential distance estimation algorithm

In our prototype, the ultrasonic receivers are set up according to Figure 8, where $L_{12} = L_{14} = 2\lambda$, and $L_{23} = L_{45} = 1.5\lambda$. An interesting aspect of our implementation is the method used to determine the unique actual differential distance from the observed differential distance. The method uses the intuition developed in Section 2.2, where an "existence" argument was made for how to configure receivers to unambiguously resolve the actual phase difference. Although the argument was made in the "phase domain," the results hold equivalently in the "wavelength domain," where the measured values are the differential distances in terms of λ. However, the argument in Section 2.2 is not prescriptive, so we outline our implemented algorithm below.

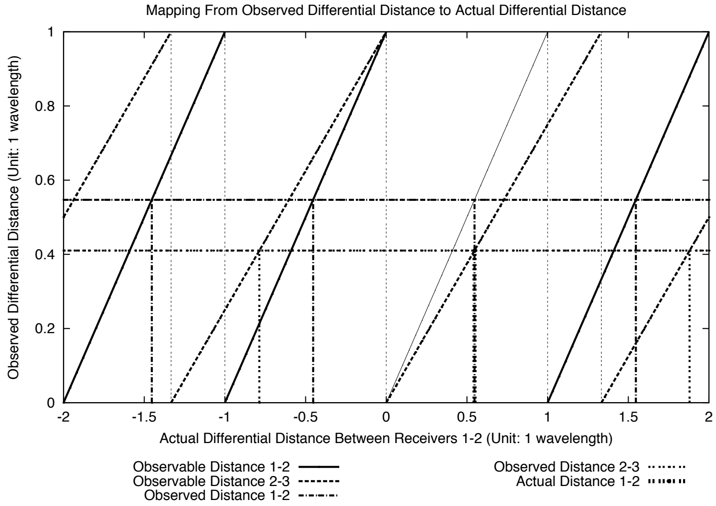


Fig. 10. Finding the actual differential distance between R_1 and R_2 by using the observed differential distances from (R_1, R_2) and (R_2, R_3).

Consider Figure 10, which plots the variation of observed differential difference $\delta d'$ as a function of the actual differential difference δd for two pairs of receivers. One of the curves (the solid line segments) shows the $\delta d'_{12}$ variation for the receiver pair (R_1, R_2), which are separated by a distance $L_{12} = 2\lambda$. The other curve (the dashed line segments) shows the variation $\delta d'_{23}$ for the receiver pair (R_2, R_3) separated by $L_{23} = 1.5\lambda$. We normalized the curves to show the observed variations of $\delta d'_{12}$ and $\delta d'_{23}$ as a function of δd_{12}; i.e., $\delta d'_{12}$ varies in the range $[0, \lambda]$ as δd_{12} varies in $[-2\lambda, 2\lambda]$.

Each curve is periodic with discontinuities. The observed value $\delta d'$ varies in the range $[0, \lambda]$ because that is the range of measurable distance between two (time-shifted) waveforms whose starting times are not known. The discontinuities are due to the fact that the observable differential distances follow the periodicity of the observed phase differences. The actual differential distances vary in the range $[-L_{12}, L_{12}]$ for δd_{12}, and in the range $[-L_{23}, L_{23}]$ for δd_{23}. But because we have normalized the curves as a function of δd_{12}, the observed phase differential curve for the receiver pair (R_2, R_3) shown in Figure 10 also varies in the range $[-L_{23} \cdot L_{12}/L_{23}, L_{23} \cdot L_{12}/L_{23}] = [-L_{12}, L_{12}]$ in the plot. The slope of each line segment is proportional to the normalized separation distance for that pair of receivers. Hence, the normalized curve for (R_1, R_2) has a slope of 1, while the curve for (R_2, R_3) has a slope of $L_{23}/L_{12} = 3/4$.

Note that because L_{12} and L_{23} are relatively prime multiples of $\lambda/2$, the periods (and discontinuities) for the two curves always differ, and the cycle of each curve (i.e., the discontinuities) do not overlap each other more than once. Consequently, the two curves do not have a repeating pattern within the interested range $[-L_{12}, L_{12}]$. Hence, we get a unique solution for the actual δd value for *any* given pair of observed $\delta d'_{12}$ and $\delta d'_{23}$ values.

Recall that the range of observable differential distances is $[0, \lambda]$. From Figure 10, we see that any observed value within this range can be mapped to four possible solutions for the actual δd_{12}. Let $A^{\delta d'_{12}}$ be the set of possible solutions derived from the observed value $\delta d'_{12}$. Graphically, these are the values on the horizontal axis extrapolated from the four intersections between the $y = \delta d'_{12}$ line and the observable differential distance curve for the receiver (R_1, R_2). Then, given an observed $\delta d'_{12}$, our task is to identify the actual differential distance from the set $A^{\delta d'_{12}}$.

By following the arguments presented in Section 2.2, we can use the observed $\delta d'_{23}$ to help us identify the correct solution. From Figure 10, the observed $\delta d'_{23}$ can be mapped to three possible solutions for the actual δd_{12}. Again, let $A^{\delta d'_{23}}$ be the set of possible solutions using the observed value $\delta d'_{23}$. Since we are guarenteed a unique solution for any given pair of observed values $\delta d'_{12}$ and $\delta d'_{23}$, we will find exactly one matching solution that exists in both $A^{\delta d'_{12}}$ and $A^{\delta d'_{23}}$. Thus, the final answer for the actual differential distance δd_{12} is a if and only if $a \in A^{\delta d'_{12}}$ and $a \in A^{\delta d'_{23}}$.

For example, Figure 10 shows that for the observed $\delta d'_{12} = 0.547$ and $\delta d'_{23} = 0.41025$, $A^{\delta d'_{12}} = \{-1.453, -0.453, 0.547, 1.547\}$ and $A^{\delta d'_{23}} = \{-0.786, 0.547, 1.880\}$. Hence, the final solution is $\delta d_{12} = 0.547$ because this value exists in both $A^{\delta d'_{12}}$ and $A^{\delta d'_{23}}$.

One caveat about this algorithm for finding the actual phase differential distance is that measurement errors may produce no matching solution that exists in both $A^{\delta d'_{12}}$ and an $A^{\delta d'_{23}}$. In such a situation, we find the closest matching solution by choosing an $a_{12} \in A^{\delta d'_{12}}$ and $a_{23} \in A^{\delta d'_{23}}$ such that $|a_{12} - a_{23}|$ is minimum. Then, we report the actual differential distance to be $\frac{a_{12}+a_{23}}{2}$.

4 Experiments

In this section, we report on several performance experiments conducted with our Cricket compass implementation. In Section 5, we outline a few improvements that we intend to implement in the future, based on what we have learned from these experiments.

We describe two distinct sets of experiments. First, we evaluate the efficacy of our differential distance estimation technique as a function of the angle θ between the compass and one fixed beacon using the techniques of Section 2.2. Then, we attach multiple beacons at different places on a ceiling and measure the accuracy of coordinate estimation using the techniques of Section 2.3. Finally, we combine the results of these experiments to perform an analytic error analysis of Equation (4) to derive an upper bound on the end-to-end errors one might expect in practice. We do this because our current prototype hardware does not allow us to obtain the average and differential beacon distances simultaneously; while we are building this combined hardware, we do want to get a sense of how accurate our system is likely to be. The following sections demonstrate that our differential distance and position estimation methods work well.

4.1 Differential Distance Estimation

In this set of experiments, we use the setup shown in Figure 2 to measure the accuracy of the differential distance values, $d_2 - d_1$, at different values of θ. We place the beacon such that it is at a height $z = 1.5m$, a horizontal distance of $x = 2.0m$ away from the receivers, and an angle θ with respect to the line joining the beacon and the receiver. The receivers are configured according to Figure 5, where $L_{12} = 2\lambda$, $L_{23} = 1.5\lambda$.

For each measurement at the specified θ, we take the *mode* of the differential distance samples to reduce the error caused by ultrasound reflections and noise. The entire experiment was repeated for three trials.

The results are shown in Figure 11, which shows the average angle estimates derived from the measured $d_2 - d_1$ values. That is, the average angle estimates were calculated by applying Equation (4) on the known values of z, x, and the average measured $d_2 - d_1$ values at each θ. From Figure 11, we see that the measured $d_2 - d_1$ values can estimate the true angle with reasonable accuracy.

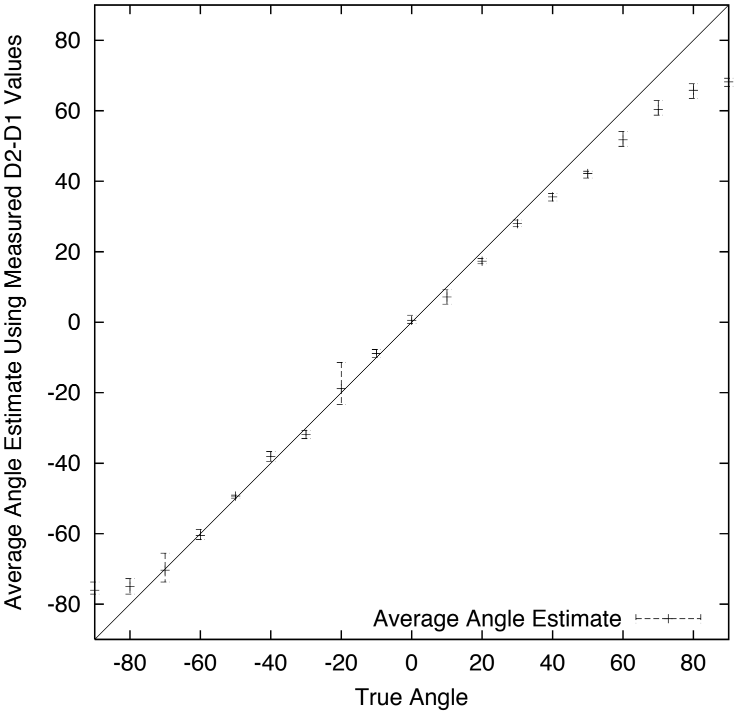


Fig. 11. Average angle estimates versus true angle values. The error bars indicate the absolute angle errors for all three trials. The line $y = x$ plots the ideal relationship between the true and estimated angle values.

Table 1 shows the average differential distance in terms of λ, the average percentage error of the differential distance for every 10° angle $-90° \leq \theta \leq 90°$, and the derived angle estimates.

$\theta°$	Actual $d_2 - d_1$ (λ)	Measured $d_2 - d_1$ (λ)	Error of $d_2 - d_1$(%)	Derived θ Estimates ($\theta°$)
-90	-1.600	-1.552	3.000	-76.021
-80	-1.576	-1.544	2.011	-74.901
-70	-1.504	-1.504	-0.033	-70.350
-60	-1.386	-1.392	-0.459	-60.482
-50	-1.226	-1.213	1.007	-49.319
-40	-1.028	-0.985	4.193	-38.021
-30	-0.800	-0.843	-5.333	-31.786
-20	-0.547	-0.516	5.707	-18.894
-10	-0.278	-0.245	11.699	-8.821
0	0	0.021	—	0.57
10	0.278	0.200	-28.015	7.184
20	0.547	0.477	-12.773	17.359
30	0.800	0.749	-6.333	27.929
40	1.028	0.931	-9.509	35.573
50	1.226	1.075	-12.320	42.202
60	1.386	1.256	-9.356	51.755
70	1.504	1.389	-7.594	60.317
80	1.576	1.459	-7.427	65.794
90	1.600	1.485	-7.167	68.196

Table 1. Differential distances (measurements averaged over 3 trials), percentage error and the derived angle estimates at each value of θ

We make three observations from these results. First, we are able to accurately estimate angles to within ±3 degrees in the range from -70 to 30 degrees, and to within ±8 degrees for angles up to 50 degrees. Second, in terms of percentage error[2] all estimated differential distances (and hence, angles for this set of experiments) have less than 13% error with the exception of $\theta = 10°$. Third, the estimates of the positive θ values consistently show a higher percentage error than those for negative θ. Moreover, they all report a value that is less than the true value. Our current hypothesis, still under active investigation, is that the causes of these errors are imperfect calibrations of the distances

---

[2] For angle estimation, the percentage error is not as interesting to most applications as absolute errors.

between the ultrasonic receivers (which a better mounting process will fix) and timing delay issues related to interrupt handlers in the PIC microprocessor on the compass. Despite these caveats, we find the ability to estimate angles to within 3-to-5 degrees for a practical range of angles promising and useful for many context-aware applications.

4.2 Distance and Position Estimation

Beacon	(x, y, z)
H	0, 121, 0
I	117, 121, 0
J	0, 0, 0
K	117, 0, 0

Table 2. Beacon coordinates (in centimeters).

For our second set of experiments, we placed four beacons on the ceiling of a room at known coordinates as shown in Table 2. Each beacon broadcasts a unique identifier, which is mapped to its known coordinates by the receiver. The receiver is placed at a specific location and collects up to 25 distinct distance samples from each of the four beacons. Because the noise and reflections of ultrasound in the environment can affect the sampled distances, we take the mode of each distance distribution for each beacon as the actual distance estimate to each beacon. We configured the compass as in the previous experiment, taking measurements at four different compass locations as shown in Table 3. At each location, we collected data across four independent trials.

Table 3 shows the coordinate estimates at each locations. We find that our position estimates can be accurate to within 5-6 centimeters, and in the worst case, to within 25 centimeters of the true value. We conducted a further investigation into the worst case situation—although it was good to less than a foot, we were interested in the underlying reasons for this behavior. We found that consistently the worst case happened when the receiver was near a wall, or when the beacon was attached close to a wall, while it was possible to obtain centimeter-level accuracy a few feet away from walls. We have since been developing techniques to handle reflections from walls, which we outline in Section 5.

Receiver Location	Actual Receiver Location (x, y, z)	Estimated Receiver Location (x, y, z)	Error (cm)
A	0.0, 121.0, 178.0	-13.8, 134.9, 193.8	25.12
B	117.0, 121.0, 178.0	123.3, 129.1, 190.3	16.05
C	0.0, 0.0, 178.0	-0.7, -5.5, 176.8	5.65
D	117.0, 0.0, 178.0	120.4, -3.0, 173.1	6.63

Table 3. Coordinate estimates at four different receiver locations.

From the coordinate estimates, we also derive a set of $z/\bar{d}$ values that are used in Equation (4). Table 4 reports the percentage error of the $z/\bar{d}$ derived from our coordinate estimates. *The results indicate that the error in $z/\bar{d}$ is at most* 2.6%*, even when near a wall, and substantially better further away.* We use this worst-case measured data in the next section to understand the theoretical error bound on overall orientation estimation using our compass.

Receiver Location	Percentage Error of z/d with Respect to Beacon			
	H	I	J	K
A	-0.50	-0.98	-0.92	0.03
B	0.38	-0.15	0.38	1.26
C	-1.65	-1.57	-0.05	-0.58
D	-2.59	-1.71	-1.75	0.28

Table 4. Percentage error of $z/\bar{d}$ with respect to each beacon for each coordinate estimation trial.

4.3 Error Analysis

We now give a simple error analysis of the angle estimation method, obtaining an expression for how it depends on the errors in the measured quantities. We use our experimental results from the previous sections that bound the accuracy with which our techniques and implementation estimate $(d2 - d1)$ and $z/\bar{d}$.

If $V(v_1, v_2)$ is a function of two *independently-measured* variables v_1 and v_2, then the error in V, $\varDelta V$, can be expressed as [6, 23]:

$$(\varDelta V)^2 = (\frac{\partial V}{\partial v_1})^2(\varDelta v_1)^2 + (\frac{\partial V}{\partial v_2})^2(\varDelta v_2)^2 \tag{12}$$

Applying this to the angle estimate θ, we get the following expression for the fractional error in θ, $\varDelta\theta$, as a function of $v_1 = d_2 - d_1$ and $v_2 = z/\bar{d}$:

$$\frac{\varDelta\theta}{\theta} = \frac{\tan\theta}{\theta} \times \sqrt{(\frac{\varDelta v_1}{v_1})^2 + (\frac{\varDelta v_2}{v_2})^2 \frac{v_2^4}{(1 - v_2^2)}} \tag{13}$$

Note that because $d_2 - d_1$ is estimated using the phase difference of the ultrasonic waveforms, and $z/\bar{d}$ is estimated using a different method combining the RF and ultrasonic signal arrival times, v_1 and v_2 satisfy the independent-measurement considerations of Equation (12).

Equation (13) shows that the error might grow to be rather large, especially for values of θ close to $\pi/2$. The physical reason for this is apparent from Figure 3, which shows that at large values of θ, small changes in x2 produce large changes in θ. Equation (13) also shows that the error might grow large when the $z/\bar{d}$ value is small. The physical reason is clear from Figure 2, which shows that as the receiver moves closer to the beacon, a small change (or error) in the differential distance produces a large change in θ.

We now apply Equation (13) to the average error values from our experimental measurements to obtain an upper bound on the expected error. For $x = 2.0m$ and $z = 1.5m$, we get $v_2 = z/\bar{d} = 0.6$. We then set $\frac{\varDelta v_2}{v_2} = 0.0259$, which is the worse average case error for $z/\bar{d}$ from Table 4. We then substitute the θ and $\frac{\varDelta v_1}{v_1}$ values from Table 1. The projected theoretical upper error bound at each θ is listed in Table 5.

We find that the theoretical upper bound on error is less than five degrees when θ is between ±40 degrees. We emphasize that this is what the theory predicts as an upper bound for each θ, and that in practice things may well be better (and are in fact better in some cases, as our reported experiments showed).

θ	$\tan\theta/\theta$	dv_1/v_1 (%)	$\Delta\theta/\theta$ (%)	$\Delta\theta^\circ$
-90	∞	3.00	∞	∞
-80	4.06	2.01	9.44	7.55
-70	2.25	0.03	2.62	1.84
-60	1.65	0.46	2.07	1.24
-50	1.37	1.01	2.10	1.05
-40	1.20	4.19	5.23	2.09
-30	1.10	5.33	6.02	1.81
-20	1.04	5.71	6.07	1.21
-10	1.01	11.70	11.88	1.19
0	1.00	—	—	—
10	1.01	28.02	28.33	2.83
20	1.04	12.77	13.37	2.67
30	1.10	6.33	7.10	2.13
40	1.20	9.51	11.51	4.61
50	1.37	12.32	16.90	8.45
60	1.65	9.36	15.59	9.36
70	2.25	7.59	17.28	12.09
80	4.06	7.43	30.54	24.43
90	∞	7.17	∞	∞

Table 5. Projected error bounds for angle estimations at each θ. The parameters are $v_2 = 0.6$ and $\frac{\Delta v_2}{v_2} = 0.0259$.

4.4 Effect of Motion

The experiments mentioned above were conducted by placing the compass on a stable platform (i.e., the linear velocity of the compass is zero). In practice, we expect the Cricket compass to be attached to mobile devices, and are interested in measuring its performance when a user walks or moves the device in their hand. We model such movement as a linear velocity and calculate the Doppler effect to examine the performance impact of such movement on the Cricket compass.

Let λ' be the observed wavelength of ultrasound due to motion, λ be the true wavelength of ultrasound from the beacon, f be the true frequency of ultrasound, and v_r be the linear velocity of the receivers in the direction towards the beacon. Then, because of the Doppler effect, we get:

$$\lambda' = \lambda - \Delta\lambda = \lambda - v_r/f$$

We use Equation (5) to derive the error $\Delta(\delta d)$ caused by the Doppler effect: z

$$\Delta(\delta d) = \frac{\phi}{2\pi} \cdot \Delta\lambda \;\; ; \;\; \Delta\lambda = \frac{v_r}{v_s} \cdot \lambda$$

where v_s is the velocity of sound. Hence, at a pedestrian walking speed of $v_r = 2.0 m/s$ and $v_s = 330 m/s$, $\Delta\lambda = 0.006\lambda$. In our implementation of the software compass, $\phi = 0.556\pi$ at $\theta = 10°$, so the error with respect to the true δd is about 1.2%. At $\theta = 40°$, the error is less than 1%.

5 Improvements

The preliminary experiments reported in the previous section show great promise, and we believe that this augurs well for the utility of our system. However, our results also raise some important issues that need to be addressed in implementation before a production system can be realized. This section describes some of these issues and our proposals to address them.

5.1 Handling Reflections

Four appropriately placed beacons can accurately estimate the position coordinates of a receiver, but our results show that the accuracy degrades when a beacon is within a few inches from a wall. This is because ultrasound reflections can cause the measured distances to be inaccurate. If there is a line-of-sight path between the beacon and the receiver, we will have a single correct[3] distance among the set of distances; if not, then several of the readings will be incorrect.

We can solve the ambiguity caused by multiple distances and errors due to incorrect distances by using *five* beacons instead of four. With five beacons, the receiver will have a set of readings containing multiple measured distances to each beacon. Now, from this set, the receiver can select four beacon values at a time, each value corresponding to a different beacon, and run the algorithm of Section 2.3 to determine its coordinate position. If the coordinates determined from two or more distinct sets of beacons are close to each other, we can select that as the correct coordinate. Otherwise, we cannot have much confidence in the correctness of the estimated coordinates (although they will likely be correct to a few inches).

---

[3] Here "correct" refers to a distance that is proportional to the actual distance.

Here, we essentially use the fifth beacon to validate the coordinates obtained using the other four; the robustness of this scheme is based on the assumption that the probability of two incorrect readings $\hat{d}_1$ and $\hat{d}_2$ giving rise to answers that coincide is negligible. An analogy might help understand why this is reasonable: Consider a line segment of length l joining two points, P_1 and P_2. We are told that a point in between them is at d_1 and d_2 away from P_1 and P_2 respectively. If both $\hat{d}_1$ and $\hat{d}_2$ are independent (and incorrect) estimates of d_1 and d_2, it is highly unlikely that the errored values will correspond to the same identical point!

5.2 Handling Diffractions

Another potential cause of error is the diffraction (bending at edges) of sound waves around obstacles. Such obstacles may not block the entire path but cause the signal to bend. If the signal arriving at the receiver is bent, then the measure angle to the beacon will have a corresponding error. The difference in distance due to bending could be on the order of millimeters, which will not be detected by the method described above since the error in the distance would be the same order of magnitude (or even less) than the accuracy of distance measurement itself.

However, the receiver can determine its orientation with respect to a fixed origin using each of the beacons it can hear from, and use values that coincide to be the right one. We intend to modify our current method of using the smallest angle and replace it with this “plurality” scheme.

5.3 Beacon and Compass Placement

One of the issues that a production-style deployment of the compass infrastructure must pay close attention to is beacon placement. From Equation 12, it is clear that the error is large when θ is large, and also when z is close to $\bar{d}$. What we would like is to ensure that, for every compass location, there is at least one beacon whose θ from that location is smaller than 45 degrees. In addition, we would also like to ensure that there is at least one visible beacon whose $z/\bar{d}$ is not bigger than some threshold value, say 0.5. This second condition means that there should be at least one beacon whose height does not “dominate” the distance to the compass, i.e., the compass should not be “directly under” all visible beacons.

For most rectangular rooms, these conditions are rather straightforward to meet without requiring a large number of beacons. In general, however, a more formal approach will be valuable to tackle this placement problem using ideas from the classical "art-gallery" problems and more recent "searchlight" problems in computational geometry. To our knowledge, constraints similar to our compass system have not been studied in the literature, and this area is open to interesting algorithm development, especially for non-rectangular rooms.

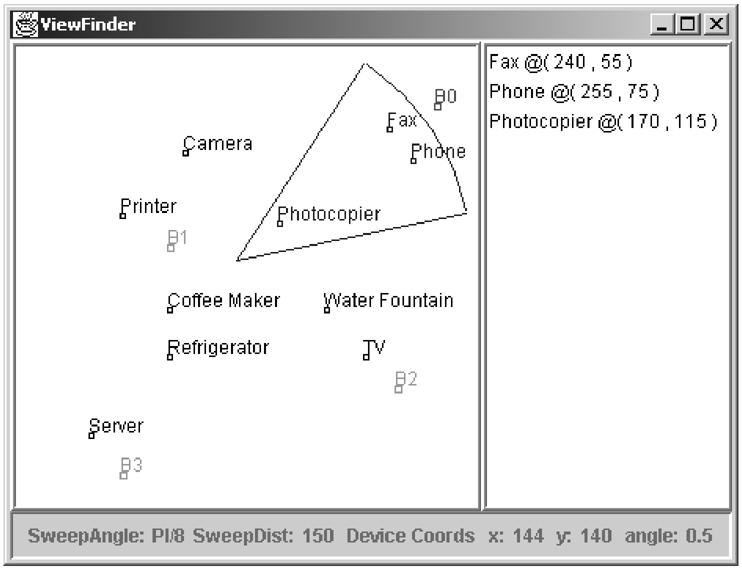
Some of the discussion in this paper assumes (perhaps tacitly) that the compass is held flat and parallel to the ground. This is not a fundamental requirement—with this requirement, all we need is the orientation with respect to one beacon, while otherwise we need the orientation with respect to at least three beacons to uniquely determine the orientation vector. Since we have at least four beacons for coordinate determination, this is not hard to accomplish.

6 The Viewfinder

We have developed the *Viewfinder* application to demonstrate the use of the location and orientation information provided by the software compass. The user defines a sweep angle β and a distance R and points a device running the Viewfinder in the desired direction. The Viewfinder then highlights the services discovered within the swept sector.

To enable this functionality, the Viewfinder application queries a resource discovery server, such as those proposed in [1, 9, 13], to obtain the global coordinates of the available services. To facilitate the bootstrapping process, the name of the server for the space is advertised on the RF channel by the beacons. We also assume that individual services use their own software compass to obtain their coordinate information, and that they advertise this information to the resource discovery system. Otherwise, a system administrator can assign global coordinates to each individual (static) service.

The Viewfinder queries the software compass for current values of the relative angle θ with respect to the beacon B, the coordinates of B, and the coordinates of the device's current location O. Then, to test whether a service S is within the user-specified sweep angle, the Viewfinder extends two vectors originating from the device's coordinates: one to B and one to S. From these vectors, the Viewfinder invokes the cosine law to find a unique solution $\theta_S = \angle SOB$, which is the angle of the service S with respect to the anchor beacon. Then the Viewfinder simply performs a series of comparisons between the relative angle values θ_S, θ, and ϕ_S to test whether S lies within the current sweep angle, and at a distance smaller than the user-specified distance.



```
Fax @( 240 , 55 )
Phone @( 255 , 75 )
Photocopier @( 170 , 115 )
```

```
SweepAngle: PI/8  SweepDist: 150  Device Coords  x: 144  y: 140  angle: 0.5
```

Fig. 12. Screen capture of the Viewfinder application. Note that the origin is at the upper left corner of the map and the angle is reported in radians, where the angle value starts at zero due east and increases counterclockwise.

Figure 12 shows a screen capture of the prototype Viewfinder application. A map of the service and beacon locations is displayed on the left panel. The tip and the body of the pie-shape figure marks the device location and its current sweep angle. Services that are within the current sweep angle appear on the right panel. The bottom panel displays the coordinate and angle values reported by the software compass.

7 Related Work

Want *et al.*'s Active Badge system, developed using infrared links, was one of the earliest indoor systems for position inference [26]. Its architecture inspired future generations including the Bat system [27] and PinPoint's local positioning system [18, 29]. In these architectures, the hardware tag attached to mobile devices is active, and responds to queries from a central controller and location database about its whereabouts. While the Bat system uses a combination of RF and ultrasound to estimate distance [14, 27], PinPoint uses spread-spectrum radio signals and multiple antennae at the controller to process messages from a tag. One of the problems with these architectures is that they track users, and lead to significant privacy concerns [19].

Bahl and Padmanabhan describe RADAR [5], an indoor RF-based location system that uses an already-existing data network to estimate position. Here, the RF signal strength is used as a measure of distance between RF transmitter and a receiver. This information is then used to locate a user using triangulation, typically using an RF signal strength map obtained by a prior instrumentation process.

Our compass system enhances the capabilities of the Cricket location system, which uses a combination of passive receivers (called "listeners") and active beacons, which provide information about a space [20]. Like the Bat system, it uses a combination of RF and ultrasound to estimate position, but uses multiple ultrasonic receivers located close to each other to infer orientation on a mobile handheld device.

The best-known system for outdoor use is the satellite-based Global Positioning System (GPS) [12, 15], which is increasingly being used in civilian applications in addition to its traditional military use. GPS does not provide the degree of precision required for mobile applications indoors because of the low RF signal strength, high RF noise, and the reflections of RF signals due to the presence of metallic objects. Bulusu *et al.* describe a low-cost location system for outdoor use [7], where the environment is instrumented with a number of fixed RF stations that periodically transmit their unique ID and position. The receivers use

RF connectivity to estimate their position relative to the known fixed RF stations.

Doherty *et al.* model the position estimation problem in ad hoc sensor networks as a convex optimization problem, showing that under some conditions it is possible for the nodes to discover their positions relative to one another [10]. Savvides *et al.* describe another approach to this problem that resembles our coordinate estimation scheme of Section 2.3. We expect variants of these approaches to be a good starting point for instrumenting beacons in our environment without having to program each beacon with its location, but programming only some of them and having the others discover their coordinates from the other beacons in their vicinity.

The Constellation system uses a combination of accelerometers, gyros, and ultrasonic sensors to estimate position and orientation [11]. Like Cricket, the Constellation relies on an active set of ultrasonic beacons to determine the initial tracking position of the device and then recursively refines the orientation estimation using information gathered by the inertial sensors. However, the tight coordination that is required between the receivers and transmitters of this system makes it unsuitable for large-scale indoor deployment. It is also unclear that this can be implemented in a handheld-like form factor.

The HiBall system uses opto-electronic tracking of hundreds or thousands of infra-red LEDs mounted in special ceiling panels [28]. It provides rapid updates of receiver position and orientation, but requires the installation of large arrays of LEDs in the ceiling and carefully machined camera at the client, which will significantly increase deployment costs.

Commerical magnetic motion trackers have been used in virtual reality and simulation applications such as head-mounted displays and biomechanic motion capture: Ascension [2], Startrak [24], and Aurora [3] are three products available today. They provide reasonably accurate estimates of the position and orientation of the target object by sending magnetic pulses and detecting the change of field strength along three orthogonal axes. These systems usually requires a centralized coordination between the magnetic transmitters and receivers and are susceptible to magnetic interference from the presence of metals or other conductive materials in the environment [16], which causes problems in many indoor environments.

Roumeliotis *et al.* describe the implementation of an orientation sensor that uses a Kalman filter to combine a compass and robot odometry with a absolute orientation signal from a “sun sensor” [21]. This system works under kinematic conditions, and its approach may be combined with Cricket to improve our system.

8 Conclusion

The Cricket compass system described in this paper reports position and orientation indoors, for a handheld, mobile device, and informs an application running on the device of the position and orientation in a local coordinate system established by the fixed set of beacons. To our knowledge, this is the first handheld-integrated system that provides a combination of orientation and position information to within a few degrees of the true value indoors, making it an attractive technology for various context-aware pervasive computing applications. It does not require large or fast user motions and works even when a traditional magnetic compass fails. The hardware configuration consists of a microcontroller, one RF receiver, and five ultrasonic receivers placed in a "V" shape a few centimeters across, processing 418 MHz RF data and 40 KHz ultrasonic signals sent from active beacons.

The challenges in deriving orientation for a small device arise from the need for sub-centimeter differential distance estimates, and from the need for accurate position estimation. We solved the first problem using multiple carefully placed receivers, deriving the mathematical conditions for placement. We solved the second problem by developing a position estimation technique that compensates for the unknown velocity of sound in an environment by observing propagation times and explicitly calculating it. Our experimental results show that we can obtain angles to within about 3 degrees of the true value in most practical settings.

9 Acknowledgments

We thank Franklin Reynolds for useful discussions on the motivation for the Cricket compass. We thank David Andersen, Rui Fan, Stephen Garland, Lars-Ake Larzon, and Suchitra Raman for several helpful comments and suggestions on previous versions of this paper.

This work was funded by NTT Inc. under the NTT-MIT research collaboration, by Acer Inc., Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under the MIT Project Oxygen partnership, and by IBM Corp. under a university faculty award.

References

1. Adjie-Winoto, W., Schwartz, E. and Balakrishnan, H. and Lilley, J. The design and implementation of an intentional naming system. In *Proc. ACM Symposium on Operating Systems Principles*, pages 186–201, Kiawah Island, SC, December 1999.
2. Ascension Technology. `http://www.ascension-tech.com/`, 2000.
3. Northern Digital Inc. - ProductsAURORA. `http://www.ndigital.com/aurora.html`, 2001.
4. R. Azuma. Tracking requirements for augmented reality. *Comm. of the ACM*, (7):50–55, July 1993.
5. P. Bahl and V. Padmanabhan. RADAR: An In-Building RF-based User Location and Tracking System. In *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.
6. Y. Beers. *Introduction to the Theory of Error.* Addison-Wesley, Reading, MA, 1957.
7. N. Bulusu, J. Heidemann, and D. Estrin. GPS-less Low Cost Outdoor Localization For Very Small Devices. Technical Report 00-729, Computer Science Department, University of Southern California, April 2000.
8. CoolTown. `http://www.cooltown.hp.com/`, 2000.
9. S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Proc. 5th ACM MOBICOM Conf.*, pages 24–35, Seattle, WA, August 1999.
10. L. Doherty, K. Pister, and L. Ghaoui. Convex position estimation in wireless sensor networks. In *Proc. IEEE INFOCOM*, April 2001.
11. E. Foxlin, M. Harrington, and G. Pfeiffer. Constellation: A Wide-Range Wireless Motion-Tracking System for Augmented Reality and Virtual Set Applications. In *Proc. ACM SIGGRAPH*, Orlando, FL, July 1998.
12. I. Getting. The Global Positioning System. *IEEE Spectrum*, 30(12):36–47, December 1993.
13. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. Simple Service Discovery Protocol/1.0. `http://search.ietf.org/internet-drafts/draft-cai-ssdp-v1-02.txt`, June 1999. Internet Draft, expires December 1999.
14. A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. In *Proc. 5th ACM MOBICOM Conf.*, Seattle, WA, August 1999.
15. B. Hoffmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System: Theory and Practice, Fourth Edition.* Springer-Verlag, 1997.
16. V. Kindratenko. Calibration of Electromagnetic Tracking Devices. *Virtual Reality: Research, Development, and Applications*, 4:139–150, 1999.
17. Oxygen home page. `http://oxygen.lcs.mit.edu/`.
18. Pinpoint home page. `http://www.pinpointco.com/`.
19. Privacy international survey. `http://www.privacyinternational.org/survey/technologies.html`.

20. N. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Proc. 6th ACM MOBICOM Conf.*, Boston, MA, August 2000.
21. S. Roumeliotis, G. Sukhatme, and G. Bekey. Smoother-based 3-d attitude estimation for mobile robot localization. In *Proc. IEEE International Conf. on Robotics and Automation*, Detroit, MI, May 1998.
22. B. Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, pages 22–32, Sep/Oct 1994.
23. B. Shchigolev. *Mathematical Analysis of Observations.* Iliffe Books Ltd., London, 1965. Originally published in the U.S.S.R. in 1960 (Russian).
24. Polhemus Star Trak. `http://www.polhemus.com/stardstech.htm`, 2000.
25. T. Turunen, T. Pyssysalo, and T. Lankila. Utilisation of Wireless Application Protocol to Implement Mobile Augmented Reality Based Services. In *Proc. W3C and WAP Workshop on Position Dependent Information Services*, February 2000. Available from `http://www.w3.org/Mobile/posdep/Oulu.html`.
26. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.
27. A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office. *IEEE Personal Comm.*, 4(5):42–47, October 1997.
28. Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. The HiBall tracker: High-performance wide-area tracking for virtual and augmented environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, December 1999.
29. J. Werb and C. Lanzl. Designing a positioning system for finding things and people indoors. *IEEE Spectrum*, 35(9):71–78, September 1998.

Proceedings of the ACM Symposium on
Applied Computing, Madrid, Spain,
March 2002, to appear.

Proxy-Based Security Protocols in Networked Mobile Devices

Matthew Burnside, Dwaine Clarke, Todd Mills, Andrew Maywah,
Srinivas Devadas and Ronald Rivest*

MIT Laboratory for Computer Science
{`event, declarke, mills, amaywah, devadas, rivest`}`@mit.edu`

Abstract. We describe a resource discovery and communication system designed for security and privacy. All objects in the system, e.g., appliances, wearable gadgets, software agents, and users have associated trusted software proxies that either run on the appliance hardware or on a trusted computer. We describe how security and privacy are enforced using two separate protocols: a protocol for secure device-to-proxy communication, and a protocol for secure proxy-to-proxy communication. Using two separate protocols allows us to run a computationally-inexpensive protocol on impoverished devices, and a sophisticated protocol for resource authentication and communication on more powerful devices.

We detail the device-to-proxy protocol for lightweight wireless devices and the proxy-to-proxy protocol which is based on SPKI/SDSI (Simple Public Key Infrastructure / Simple Distributed Security Infrastructure). A prototype system has been constructed, which allows for secure, yet efficient, access to networked, mobile devices. We present a quantitative evaluation of this system using various metrics.

1 Introduction

Attaining the goals of ubiquitous and pervasive computing [6, 2] is becoming more and more feasible as the number of computing devices

---

* This work was funded by Acer Inc., Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under the MIT Project Oxygen partnership, and by DARPA through the Office of Naval Research under contract number N66001-99-2-891702.

in the world increases rapidly. However, there are still significant hurdles to overcome when integrating wearable and embedded devices into a ubiquitous computing environment. These hurdles include designing devices smart enough to collaborate with each other, increasing ease-of-use, and enabling enhanced connectivity between the different devices.

When connectivity is high, the security of the system is a key factor. Devices must only allow access to authorized users and must also keep the communication secure when transmitting or receiving personal or private information.

Implementing typical forms of secure, private communication using a public-key infrastructure on all devices is difficult because the necessary cryptographic algorithms are CPU-intensive. A common public-key cryptographic algorithm such as RSA using 1024-bit keys takes 43ms to sign and 0.6ms to verify on a 200MHz Intel Pentium Pro (a 32-bit processor) [30]. Some devices may have 8-bit micro-controllers running at 1-4 MHz, so public-key cryptography on the device itself may not be an option. Nevertheless, public-key based communication between devices over a network is still desirable.

This paper presents our approach to addressing these issues. We describe the architecture of our resource discovery and communication system in Section 2. The device-to-proxy security protocol is described in Section 3. We review SPKI/SDSI and present the proxy-to-proxy protocol that uses SPKI/SDSI in Section 4. Related work is discussed in Section 5. The system is evaluated in Section 6.

1.1 Our approach

To allow the architecture to use a public-key security model on the network while keeping the devices themselves simple, we create a software proxy for each device. All objects in the system, e.g., appliances, wearable gadgets, software agents, and users have associated trusted software proxies that either run on an embedded processor on the appliance, or on a trusted computer. In the case of the proxy running on an embedded processor on the appliance, we assume that device to proxy communication is inherently secure.[1] If the device has minimal computational power,[2] and communicates to its proxy through a wired or

---

[1] For example, in a video camera, the software that controls various actuators runs on a powerful processor, and the proxy for the camera can also run on the embedded processor.

[2] This is typically the case for lightweight devices, e.g., remote controls, active badges, etc.

wireless network, we force the communication to adhere to a device-to-proxy protocol (cf. Section 3). Proxies communicate with each other using a secure proxy-to-proxy protocol based on SPKI/SDSI (Simple Public Key Infrastructure / Simple Distributed Security Infrastructure). Having two different protocols allows us to run a computationally-inexpensive security protocol on impoverished devices, and a sophisticated protocol for resource authentication and communication on more powerful devices. We describe both protocols in this paper.

1.2 Prototype automation system

Using the ideas described above, we have constructed a prototype automation system which allows for secure, yet efficient, access to networked, mobile devices. In this system, each user wears a badge called a K21 which identifies the user and is location-aware: it "knows" the wearer's location within a building. User identity and location information is securely transmitted to the user's software proxy using the device-to-proxy protocol.

Devices themselves may be mobile and may change locations. Attribute search over all controllable devices can be performed to find the nearest device, or the most appropriate device under some metric.[3]

By exploiting SPKI/SDSI, security is not compromised as new users and devices enter the system, or when users and devices leave the system. We believe that the use of two different protocols, and the use of the SPKI/SDSI framework in the proxy-to-proxy protocol has resulted in a secure, scalable, efficient, and easy-to-maintain automation system.

2 System architecture

The system has three primary component types: devices, proxies and servers. A *device* refers to any type of shared network resource, either hardware or software. It could be a printer, a wireless security camera, a lamp, or a software agent. Since communication protocols and bandwidth between devices can vary widely, each device has a unique *proxy* to unify its interface with other devices. The *servers* provide naming and discovery facilities to the various devices.

We assume a one-to-one correspondence between devices and proxies. We also assume that all users are equipped with K21s, whose proxies run on trusted computers. Thus our system only needs to deal with devices, proxies and the server network.

---

[3] For example, a user may wish to print to the nearest printer that he/she has access to.

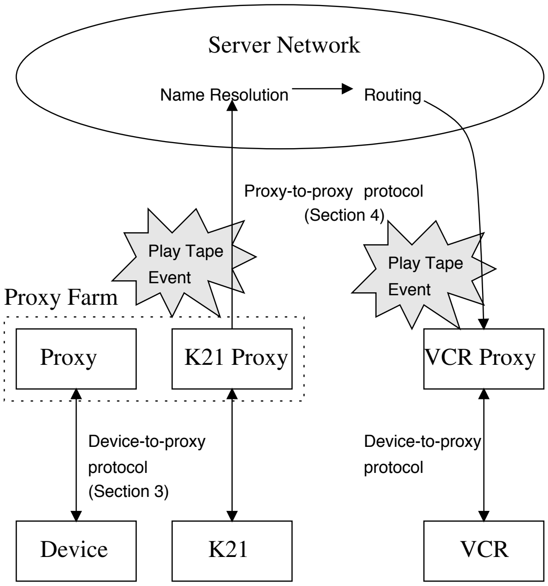


Fig. 1. System Overview

The system we describe is illustrated in Figure 1.

2.1 Devices

Each device, hardware or software, has an associated trusted software proxy. In the case of a hardware device, the proxy may run on an embedded processor within the device, or on a trusted computer networked with the device. In the case of a software device, the device can incorporate the proxy software itself.

Each device communicates with its own proxy over the appropriate protocol for that particular device. A printer wired into an Ethernet can communicate with its proxy using TCP/IP. A wireless camera uses a wireless protocol for the same purpose. The K21 (a simple device with a lightweight processor) communicates with its proxy using the particular device-to-proxy protocol described in Section 3. Thus, the device-side portion of the proxy must be customized for each particular device.

2.2 Proxy

The proxy is software that runs on a network-visible computer. The proxy's primary function is to make access-control decisions on behalf of the device it represents. It may also perform secondary functions such as running scripted actions on behalf of the device and interfacing with a directory service.

The proxy provides a very simple API to the device. The *sendToProxy*() method is called by the device to send messages to the proxy. The *sendToDevice*() method is a called by the proxy to send messages to the device. When a proxy receives a message from another proxy, depending on the message, the proxy may translate it into a form that can be understood by the proxy's particular device. It then forwards the message to the device. When a proxy receives a message from its device, it may translate the message into a general form understood by all proxies, and then forward the message to other proxies. Any time a proxy receives a message, before performing a translation and passing the message on to the device, it performs the access control checks described in Section 4.

For ease of administration, we group proxies by their administrators. An administrator's set of proxies is called a *proxy farm*. This set specifically includes the proxy for the administrator's K21, which is considered the root proxy of the proxy farm. When the administrator adds a new device to the system, the device's proxy is automatically given a default ACL, a duplicate of the ACL for the administrator's K21 proxy. The administrator can manually change the ACL later, if he desires.

A noteworthy advantage of our proxy-based architecture is that it addresses the problem of viruses in pervasive computing environments. Sophisticated virus scanning software can be installed in the proxy, so it can scan any code before it is downloaded onto the device.

2.3 Servers and the server network

This network consists of a distributed collection of independent name servers and routers. In fact, each server acts as both a name server *and* a router. This is similar to the name resolvers in the Intentional Naming System (INS) [1], which resolve device names to IP addresses, but can also route events. If the destination name for an event matches multiple proxies, the server network will route the event to all matching destinations.

When a proxy comes online, it registers the name of the device it represents with one of these servers. When a proxy uses a server to perform a lookup on a name, the server searches its directory for all names that match the given name, and returns their IP addresses.

2.4 Communication via events

We use an event-based communication mechanism in our system. That is, all messages passed between proxies are signals indicating that some event has occurred. For example, a light bulb might generate *light-on* and *light-off* events. To receive these messages, proxy x can add itself as an event-listener to proxy y. Thus, when y generates an event, x will receive a copy.

In addition, the system has several pre-defined event categories which receive special treatment at either the proxy or server layer. They are summarized in Figure 2. A developer can define his own events as well. The server network simply passes developer-defined events through to their destination.

CommandEvent Used to instruct a device to turn on or off, for example.

ErrorEvent Generated and broadcast to all listeners when an error condition occurs.

StatusChangeEvent Generated when, for example, a device changes its location.

QueryEvent When a server receives a QueryEvent, it performs a DNS (Domain Name Service) or INS lookup on the query, and returns the results of the lookup in a ResponseEvent.

ResponseEvent Generated in response to a QueryEvent.

Fig. 2. Predefined Event Types

The primary advantage of the event-based mechanism is that it eliminates the need to repeatedly poll a device to determine changes in its status. Instead, when a change occurs, the device broadcasts an event to all listeners. Systems like Sun Microsystems' Jini [26] issue "device drivers" (RMI stubs) to all who wish to control a given device. It is then possible to make local calls on the device driver, which are translated into RMI calls on the device itself.

2.5 Resource discovery

The mechanism for resource discovery is similar to the resource discovery protocol used by Jini. When a device comes online, it instructs its proxy to repeatedly broadcast a request for a server to the local subnetwork. The request contains the device's name and the IP address and port of its proxy. When a server receives one of these requests, it issues a lease to the proxy.[4] That is, it adds the name/IP address pair to its directory. The proxy must periodically renew its lease by sending the same name/IP address pair to the server, otherwise the server removes it from the directory. In this fashion, if a device silently goes offline, or the IP address changes, the proxy's lease will no longer get renewed and the server will quickly notice and either remove it from the directory or create a new lease with the new IP address.

For example, imagine a device with the name [name=foo] which has a proxy running on 10.1.2.3:4011. When the device is turned on, it informs its proxy that it has come online, using a protocol like the device-to-proxy protocol described in Section 3. The proxy begins to broadcast lease-request packets of the form ⟨[name=foo], 10.1.2.3:4011⟩ on the local subnetwork. When (or if) a server receives one of these packets, it checks its directory for [name=foo]. If [name=foo] is not there, the server creates a lease for it by adding the name/IP address pair to the directory. If [name=foo] *is* in the directory, the server renews the lease. Suppose at some later time the device is turned off. When the device goes down, it brings the proxy offline with it, so the lease request packets no longer get broadcast. That device's lease stops getting renewed. After some short, pre-defined period of time, the server expires the unrenewed lease and removes it from the directory.

3 Device-to-proxy protocol for wireless devices

3.1 Overview

The device-to-proxy protocol varies for different types of devices. In particular, we consider lightweight devices with low-bandwidth wireless network connections and slow CPUs, and heavyweight devices with higher-bandwidth connections and faster CPUs. We assume that heavyweight devices are capable of running proxy software locally (i.e., the proxy for a printer could run on the printer's CPU). With a local proxy, a sophisticated protocol for secure device-to-proxy communication is unnecessary, assuming critical parts of the device are tamper resistant. For lightweight devices, the proxy must run elsewhere. This section

---

[4] Handling the scenario where the device is making false claims about its attributes in the lease request packet is the subject of ongoing research.

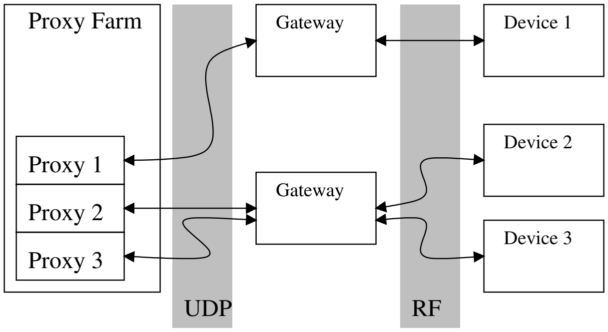


Fig. 3. Device-to-Proxy Communication overview

gives an overview of a protocol which is low-bandwidth and not CPU-intensive that we use for lightweight device-to-proxy communication.

3.2 Communication

Our prototype system layers the security protocol described below over a simple radio frequency (RF) protocol. The RF communication between a device and its proxy is handled by a gateway that translates packetized RF communication into UDP/IP packets, which are then routed over the network to the proxy. The gateway also works in the opposite direction by converting UDP/IP packets from the proxy into RF packets and transmitting them to the device.

An overview of the communication is shown in Figure 3. This figure shows a computer running three proxies; one for each of three separate devices. The figure also shows how multiple gateways can be used; device A is using a different gateway from devices B and C.

3.3 Security

The proxy and device communicate through a secure channel that encrypts and authenticates all the messages. The HMAC-MD5 [13][20] algorithm is used for authentication and the RC5 [21] algorithm is used for encryption. Both of these algorithms use symmetric keys; the proxy and the device share 128-bit keys.

Authentication HMAC (Hashed Message Authentication Code) produces a MAC (Message Authentication Code) that can validate the authenticity and integrity of a message. HMAC uses secret keys, and thus only someone who knows a particular key can create a particular MAC or verify that a particular MAC is correct.

Encryption The data is encrypted using the RC5 encryption algorithm. We chose RC5 because of its simplicity and performance. Our RC5 implementation is based on the OpenSSL [16] code. RC5 is a block cipher; it usually works on eight-byte blocks of data. However, by implementing it using output feedback (OFB) mode, it can be used as a stream cipher. This allows for encryption of an arbitrary number of bytes without having to worry about blocks of data.

OFB mode works by generating an encryption pad from an initial vector and a key. The encryption pad is then XOR'ed with the data to produce the ciphertext. Since $X \oplus Y \oplus Y = X$, the ciphertext can be decrypted by producing the same encryption pad and XOR'ing it with the ciphertext. Since this only requires the RC5 encryption routines to generate the encryption pad, separate encrypt and decrypt routines are not required.

For our implementation, we use 16 rounds for RC5. We use different 128-bit keys for encryption and authentication.

3.4 Location

Device location is determined using the Cricket location system[18, 17]. Cricket has several useful features, including user privacy, decentralized control, low cost, and easy deployment. Each device determines its own location. It is up to the device to decide if it wants to let others know where it is.

In the Cricket system, beacons are placed on the ceilings of rooms. These beacons periodically broadcast location information (such as "Room 4011") that can be heard by Cricket listeners. At the same time that this information is broadcast in the RF spectrum, the beacon also broadcasts an ultrasound pulse. When a listener receives the RF message, it measures the time until it receives the ultrasound pulse. The listener determines its distance to the beacon using the time difference.

4 Proxy to proxy protocol

SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) [7, 22] is a security infrastructure that is designed to facilitate the development of scalable, secure, distributed computing systems. SPKI/SDSI provides fine-grained access control using a local name space architecture and a simple, flexible, trust policy model.

SPKI/SDSI is a public key infrastructure with an egalitarian design. The *principals are the public keys* and each public key is a certificate authority. Each principal can issue certificates on the same basis as any other principal. There is no hierarchical global infrastructure. SPKI/SDSI communities are built from the bottom-up, in a distributed manner, and do not require a trusted "root."

4.1 SPKI/SDSI integration

We have adopted a client-server architecture for the proxies. When a particular principal, acting on behalf of a device or user, makes a request via one proxy to a device represented by another proxy, the first proxy acts like a client, and the second as a server. Resources on the server are either public or protected by SPKI/SDSI ACLs. A SPKI/SDSI ACL consists of a list of entries. Each entry has a subject (a key or group) and a tag which specifies the set of operations that that key or group is allowed to perform. To gain access to a resource protected by an ACL, a requester must include, in his request, a chain of certificates demonstrating that he is a member of a group in an entry on the ACL.[5]

If a requested resource is protected by an ACL, the principal's request must be accompanied by a "*proof of authenticity*" that shows that it is authentic, and a "*proof of authorization*" that shows the principal is authorized to perform the particular request on the particular resource. The proof of authenticity is typically a signed request, and the proof of authorization is typically a chain of certificates. The principal that signed the request must be the same principal that the chain of certificates authorizes.

This system design, and the protocol between the proxies, is very similar to that used in SPKI/SDSI's Project Geronimo, in which SPKI/SDSI was integrated into Apache and Netscape, and used to provide client access control over the web. Project Geronimo is described in two Master's theses [3, 14].

4.2 Protocol

The protocol implemented by the client and server proxies consists of four messages. This protocol is outlined in Figure 4, and following is its description:

---

[5] For examples of SPKI/SDSI ACLs and certificates, see [7] or [3].

4.2 Protocol

The protocol implemented by the client and server proxies consists of four messages. This protocol is outlined in Figure 4, and following is its description:

1. The client proxy sends a request, unauthenticated and unauthorized, to the server proxy.
2. If the client requests access to a protected resource, the server responds with the ACL protecting the resource[6] and the *tag* formed from the client's request. A tag is a SPKI/SDSI data structure which represents a set of requests. There are examples of tags in the SPKI/SDSI IETF drafts [7]. If there is no ACL protecting the requested resource, the request is immediately honored.
3. (a) The client proxy generates a chain of certificates using the SPKI/SDSI *certificate chain discovery algorithm* [4,3]. This certificate chain provides a *proof of authorization* that the user's key is authorized to perform its request.

   The certificate chain discovery algorithm takes as input the ACL and tag from the server, the user's public key (principal), the user's set of certificates, and a timestamp. If it exists, the algorithm returns a chain of user certificates which provides proof that the user's public key is authorized to perform the operation(s) specified in the tag, at the time specified in the timestamp.

   If the algorithm is unable to generate a chain because the user does not have the necessary certificates,[7] or if the user's key is directly on the ACL, the algorithm returns an empty certificate chain. The client generates the timestamp using its local clock.

   (b) The client creates a SPKI/SDSI sequence [7] consisting of the tag and the timestamp. It signs this sequence with the user's private key, and includes copy of the user's public key in the

---

[6] The ACL itself could be a protected resource, protected by another ACL. In this case, the server will return the latter ACL. The client will need to demonstrate that the user's key is on this ACL, either directly or via certificates, before gaining access to the ACL protecting the object to which access was originally requested.

[7] If the user does not have the necessary certificates, the client could immediately return an error. In our design, however, we choose not to return an error at this point; instead, we let the client send an empty certificate chain to the server. This way, when the request does not verify, the client can possibly be sent some error information by the server which lets the user know where he should go to get valid certificates.

SPKI/SDSI signature. The client then sends the tag-timestamp sequence, the signature, and the certificate chain generated in step 3a to the server.

4. The server verifies the request by:
    (a) Checking the timestamp in the tag-timestamp sequence against the time on the server's local clock to ensure that the request was made recently.[8]
    (b) Recreating the tag from the client's request and checking that it is the same as the tag in the tag-timestamp sequence.
    (c) Extracting the public key from the signature.
    (d) Verifying the signature on the tag-timestamp sequence using this key.
    (e) Validating the certificates in the certificate chain.
    (f) Verifying that there is a chain of authorization from an entry on the ACL to the key from the signature, via the certificate chain presented. The authorization chain must authorize the client to perform the requested operation.

    If the request verifies, it is honored. If it does not verify, it is denied and the server proxy returns an error to the client proxy. This error is returned whenever the client presents an authenticated request that is denied.

The protocol can be viewed as a typical challenge-response protocol. The server reply in step 2 of the protocol is a challenge the server issues the client, saying, "You are trying to access a protected file. Prove to me that you have the credentials to perform the operation you are requesting on the resource protected by this ACL." The client uses the ACL to help it produce a certificate chain, using the SPKI/SDSI certificate chain discovery algorithm. It then sends the certificate chain and signed request in a second request to the server proxy. The signed request provides proof of authenticity, and the certificate chain provides proof of authorization. The server attempts to verify the second request, and if it succeeds, it honors the request.

The timestamp in the tag-timestamp sequence helps to protect against certain types of replay attacks. For example, suppose the server logs requests and suppose that this log is not disposed of properly. If an adversary gains access to the logs, the timestamp prevents him from

---

[8] In our prototype implementation, the server checks that the timestamp in the client's tag-timestamp sequence is within five minutes of the server's local time.

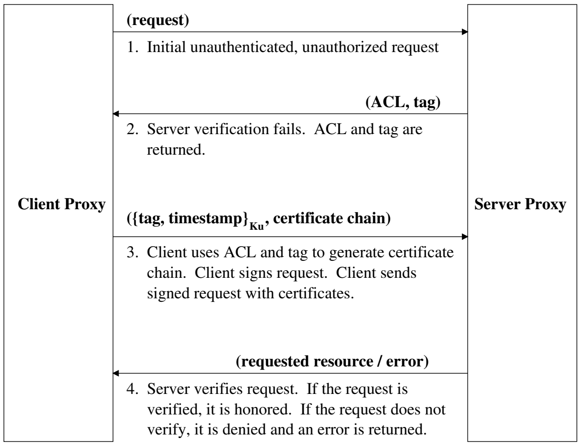


Fig. 4. SPKI/SDSI Proxy to Proxy Access Control Protocol

replaying requests found in the log and gaining access to protected resources.[9]

Additional security considerations The SPKI/SDSI protocol, as described, addresses the issue of providing client access control. The protocol does not ensure confidentiality, authenticate servers, or provide protection against replay attacks from the network.

The Secure Sockets Layer (SSL) protocol is the most widely used security protocol today. The Transport Layer Security (TLS) protocol is the successor to SSL. Principal goals of SSL/TLS [19] include providing confidentiality and data integrity of traffic between the client and server, and providing authentication of the server. There is support for client authentication, but client authentication is optional. The SPKI/SDSI Access Control protocol can be layered over a key-exchange protocol like TLS/SSL to provide additional security. TLS/SSL currently uses the X.509 PKI to authenticate servers, but it could just as well use SPKI/SDSI in a similar manner. In addition to the features already stated, SSL/TLS also provides protection against replay attacks

---

[9] In order to use timestamps, the client's clock and server's clock need to be fairly synchronized; SPKI/SDSI already makes an assumption about fairly synchronized clocks when validity time periods are specified in certificates. An alternative approach to using timestamps is to use nonces in the protocol.

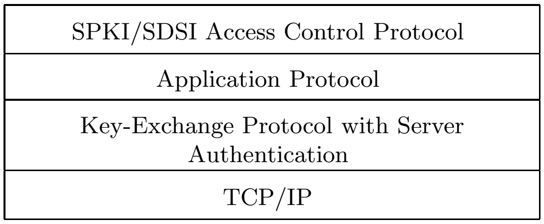


Fig. 5. Example Layering of Protocols

from the network, and protection against person-in-the-middle attacks. With these considerations, the layering of the protocols is shown in Figure 5. In the figure, 'Application Protocol' refers to the standard communication protocol between the client and server proxies, without security.

SSL/TLS authenticates the server proxy. However, it does not indicate whether the server proxy is authorized to accept the client's request. For example, it may be the case that the client proxy is requesting to print a 'top secret' document, say, and only certain printers should be used to print 'top secret' documents. With SSL/TLS and the SPKI/SDSI Client Access Control Protocol we have described so far, the client proxy will know that the public key of the proxy with which it is communicating is bound to a particular address, and the server proxy will know that the client proxy is authorized to print to it. However, the client proxy still will not know if the server proxy is authorized to print 'top secret' documents. If it sends the 'top secret' document to be printed, the server proxy will accept the document and print it, even though the document should not have been sent to it in the first place.

To approach this problem, we propose extending the SPKI/SDSI protocol so that the client requests authorization from the server and the server proves to the client that it is authorized to handle the client's request (before the client sends the document off to be printed). To extend the protocol, the SPKI/SDSI protocol described in Section 4.2 is run from the client proxy to the server proxy, and then run in the *reverse* direction, from the server proxy to the client proxy. Thus, the client proxy will present a SPKI/SDSI certificate chain proving that it is authorized to perform its request, and the server proxy will present a SPKI/SDSI certificate chain proving that it is authorized to accept and perform the client's request. Again, if additional security is needed, the extended protocol can be layered over SSL/TLS.

Note that the SPKI/SDSI Access Control Protocol is an example of the *end-to-end argument* [23]. The access control decisions are made in the uppermost layer, involving only the client and the server.

5 Related Work

5.1 Device to proxy communication

The Resurrecting Duckling is a security model for ad-hoc wireless networks [25, 24]. In this model, when devices begin their lives, they must be "imprinted" before they can be used. A master (the mother duck) imprints a device (the duckling) by being the first one to communicate with it. After imprinting, a device only listens to its master. During the process of imprinting, the master is placed in physical contact with the device and they share a secret key that is then used for symmetric-key authentication and encryption. The master can also delegate the control of a device to other devices so that control is not always limited to just the master. A device can be "killed" by its master then resurrected by a new one in order for it to swap masters.

5.2 Proxy to proxy communication

Jini [26] network technology from Sun Microsystems centers around the idea of federation building. Jini avoids the use of proxies by assuming that all devices and services in the system will run the Java Virtual Machine. The SIESTA project [8] at the Helsinki University of Technology has succeeded in building a framework for integrating Jini and SPKI/SDSI. Their implementation has some latency concerns, however, when new authorizations are granted. UC Berkeley's Ninja project [27] uses the Service Discovery Service [5] to securely perform resource discovery in a wide-area network. Other related projects include Hewlett-Packard's CoolTown [9], IBM's TSpaces [11] and University of Washington's Portolano [29].

5.3 Other projects using SPKI/SDSI

Other projects using SPKI/SDSI include Hewlett-Packard's e-Speak product [10], Intel's CDSA release [12], and Berkeley's OceanStore project [28]. HP's eSpeak uses SPKI/SDSI certificates for specifying and delegating authorizations. Intel's CDSA release, which is open-source, includes a SPKI/SDSI service provider for building certificates, and a module (AuthCompute) for performing authorization computations. OceanStore uses SPKI/SDSI names in their naming architecture.

6 Evaluation

6.1 Hardware design

Details on the the design of a board that can act as the core of a lightweight device, or as a wearable communicator, are given in [15].

6.2 Device-to-proxy protocol

In this section we evaluate the device-to-proxy protocol described in Section 3 in terms of its memory and processing requirements.

Component	Code Size (KB)	Data Size (bytes)
Device Functionality	2.0	191
RF Code	1.1	153
HMAC-MD5	4.6	386
RC5	3.2	256
Miscellaneous	1.0	0
Total	11.9	986

Table 1. Code and data size on the Atmel processor

Memory requirements Table 1 breaks down the memory requirements for various software components. The code size represents memory used in Flash, and data size represents memory used in RAM. The device functionality component includes the packet and location processing routines. The RF code component includes the RF transmit and receive routines as well as the Cricket listener routines. The miscellaneous component is code that is common to all of the other components.

The device code requires approximately 12KB of code space and 1KB of data space. The security algorithms, HMAC-MD5 and RC5, take up most of the code space. Both of these algorithms were optimized in assembly, which reduced their code size by more than half. The code could be better optimized, but this gives a general idea of how much memory is required. The code size we have attained is small enough that it can be incorporated into virtually any device.

Function	Time (ms)	Clock Cycles
RC5 encrypt/ decrypt (n bytes)	$0.163n + 0.552$	$652n + 2208$
HMAC-MD5 up to 56 bytes	11.48	45,920

Table 2. Performance of encryption and authentication code

Processing requirements The security algorithms put the most demand on the device. Table 2 breaks down the approximate time for each algorithm. The RC5 processing time varies linearly with the number of bytes being encrypted or decrypted. The HMAC-MD5 routine, on the other hand, takes a constant amount of time up to 56 bytes. This is because HMAC-MD5 is designed to work on blocks of data, so anything less than 56 bytes is padded. Since we limit the RF packet size to 50 bytes, we only analyze the HMAC-MD5 running time for packets of size less than or equal to 50 bytes.

We now examine how long it takes the device to receive a packet, process it, and send a response. In this analysis, we assume the device is receiving a packet that has 10 data bytes, making the total packet size 27 bytes, since each packet contains 17 header bytes made up of a 9-byte address field and an 8-byte message authentication field. The device broadcasts at 19.2 Kbps and we encode 8 bits into 12 bits for DC balance. To receive the packet it takes:

$$\frac{\text{packet size} + \text{RF header}}{\text{bandwidth}} = \frac{12 \cdot (27 + 4)}{19200} = 19.38\text{ms}$$

The device then takes 11.48ms to authenticate the packet and $0.163 \cdot 10 + 0.552 = 2.18$ms to decrypt it. Thus, the time for the device to receive a packet and process it is $19.38 + 11.48 + 2.18 = 33.04$ms. The device always sends back a response. In this analysis, we will assume the device responds with a packet of the same size, so the device must encrypt, authenticate, and then transmit the response which will take another 33.04ms. Thus, the device can handle approximately $\frac{1000}{33.04 \cdot 2} \approx 15$ transactions per second. We think that fifteen transactions per second is sufficient for most purposes, with a simple device.

6.3 SPKI/SDSI evaluation

The protocol described in Section 4 is efficient. The first two steps of the protocol are a standard request/response pair; no cryptography

Protocol step	Timing analysis	Approx CPU time
Cert chain discovery	The worst case is $O(n^3 l)$, where n = number of certs, and l = length of longest subject. However, the expected time is $O(nl)$.	330ms, with $n = 2$ and $l = 2$.
Chain validation	The worst case is $O(n)$, where n = number of certs.	200ms, with $n = 2$.

Table 3. Proxy-to-Proxy Protocol analysis.

is required. The significant steps in the protocol are step 3, in which a certificate chain is formed, and step 4, where the chain is verified. Table 3 shows analyses of these two steps. The paper on Certificate Chain Discovery in SPKI/SDSI [4] should be referred to for a discussion of the timing analyses. The CPU times are approximate times measured on a Sun Microsystems Ultra-1 running SunOS 5.7.

7 Conclusions

We believe that the trends in pervasive computing are increasing the diversity and heterogeneity of networks and their constituent devices. Developing security protocols that can handle diverse, mobile devices networked in various ways represents a major challenge. In this paper, we have taken a first step toward meeting this challenge by observing the need for multiple security protocols, each with different characteristics and computational requirements. While we have described a prototype system with two different protocols, other types of protocols could be included if deemed necessary.

The two protocols we have described have vastly different characteristics, because they apply to different scenarios. The device-to-proxy protocol was designed to enable secure communication of data from a lightweight device. The SPKI/SDSI-based proxy-to-proxy protocol was designed to enable communication between sophisticated devices, whose access control policies can change frequently. The proxy architecture and the use of two different protocols has resulted, we believe, in a secure, yet efficient, resource discovery and communication system.

References

1. William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremey Lilley. The Design and Implementation of an Intentional Naming System.

Operating Systems Review, 34(5):186-301, December 1999.

2. Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremey Sussman, and Deborra Zukowski. Challenges: An Application Model for Pervasive Computing. In *Proc. ACM MOBICOM*, August 2000.
3. Dwaine Clarke. SPKI/SDSI HTTP Server / Certificate Chain Discovery in SPKI/SDSI. Master's thesis, Massachusetts Institute of Technology, 2001.
4. Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security*, 2001. To appear.
5. Steven Czerwinski, Ben Zhao, Todd Hodes, Anthony Joseph, and Randy Katz. An Architecture for a Secure Service Discovery Service. In *Proc. MOBICOM*, August 1999.
6. Michael Dertouzos. The Future of Computing. *Scientific American*, August 1999.
7. Carl Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Simple Public Key Certificate. *The Internet Society*, July 1999. See http://world.std.com/~cme/spki.txt.
8. Pasi Eronen and Pekka Nikander. Decentralized Jini Security. In *Proc. of the Network and Distributed System Security Symposium*, February 2001.
9. Hewlett-Packard. CoolTown. See http://cooltown.hp.com.
10. Hewlett-Packard. e-Speak. See http://www.e-speak.hp.com.
11. IBM. TSpaces: Intelligent Connectionware. See http://www.almaden.ibm.com/cs/TSpaces.
12. Intel. Intel Common Data Security Architecture. See http://developer.intel.com/ial/security.
13. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Internet Request for Comments RFC 2104, February 1997.
14. Andrew Maywah. An Implementation of a Secure Web Client Using SPKI/SDSI Certificates. Master's thesis, Massachusetts Institute of Technology, 2000.
15. Todd Mills. An Architecture and Implementation of Secure Device Communication in Oxygen. Master's thesis, Massachusetts Institute of Technology, 2001.
16. OpenSSL. The OpenSSL Project. http://www.openssl.org.
17. N. Priyantha. Providing Precise Indoor Location Information to Mobile Devices. Master's thesis, Massachusetts Institute of Technology, January 2001.
18. N. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Proc. ACM MOBICOM*, August 2000.
19. Eric Rescola. *SSL and TLS: Designing and Building Secure Systems.* Addison-Wesley, 2001.
20. Ronald Rivest. The MD5 Message-Digest Algorithm. Internet Request for Comments RFC 1321, April 1992.

21. Ronald Rivest. The RC5 Encryption Algorithm. In *Proc. of the 1994 Leuven Workshop on Fast Software Encryption*, 2001.
22. Ronald L. Rivest and Butler Lampson. SDSI - A Simple Distributed Security Infrastructure. See http://theory.lcs.mit.edu/ rivest/sdsi10.ps.
23. J. H. Saltzer, D.P. Reed, and D. D. Clark. End-to-End Arguments in System Design. See http://www.mit.edu/∼Saltzer/publications/endtoend/.
24. Frank Stajano. The Resurrecting Duckling – What next? In *Proc. of the 8th International Workshop on Security Protocols*, April 2000.
25. Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Proc. Security Protocols, 7th International Workshop*, 1999.
26. Sun Microsystems Inc. Jini Network Techonology. http://www.sun.com/jini.
27. UC Berkeley. The Ninja Project: Enabling Internet-scale Services from Arbitrarily Small Devices. See http://ninja.cs.berkeley.edu.
28. UC Berkeley. The OceanStore Project: Providing Global-Scale Persistent Data. See http://oceanstore.cs.berkeley.edu.
29. University of Washington. Portolano: An Expedition into Invisible Computing. See http://portolano.cs.washington.edu.
30. M. Weiner. Performance Comparison of Public-key Cryptosystems. *RSA Laboratories' CryptoBytes*, 4(1), 1998.

Systems

Proceedings of the 2nd International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS'01), Krakow, Poland, 2001.

Rascal - A Resource Manager for Multi Agent Systems in Smart Spaces

Krzysztof Gajos*

MIT Artificial Intelligence Laboratory
`kgajos@ai.mit.edu`

Abstract. Multi Agent Systems (MAS) are often used as a software substrate in creating smart spaces. Many of the solutions already developed within the MAS community are applicable in the domain of smart spaces. Others, however, need to be modified or re-developed. In particular, it has to be noted that many agents acting in a physical space domain are restricted in number and capability by the scarce physical hardware available. Those limitations need to be taken into account when coordinating agent activities in a MAS in a smart space. In this paper we present Rascal, a high-level resource management system for the Intelligent Room Project, that addresses physical resource scarcities. Rascal performs the *service mapping* and *arbitration* functions for the system. Rascal is an implemented tool and has been partially deployed for day-to-day use.

1 Introduction

Building smart spaces requires distributing computation across a number of computers. The software components of a smart space need to cooperate robustly and the system must be able to cope with components being added and removed dynamically. For that reason a number

---

* This work was supported in part by the Advanced Research Project Agency of the Department of Defense under contract number F30602–92–C0204, monitored through Rome Laboratory, and in part by Acer Inc., Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under the MIT Project Oxygen partnership, and by DARPA through the Office of Naval Research under contract number N66001-99-2-891702.

of research groups have adopted an approach in which a multi-agent system (MAS) is the software substrate connecting all of the computational components of a smart space [9, 5, 8].

Agents in smart spaces have to deal with many of the same issues as agents in other MAS. At the same time, physical spaces are a domain with their own features and constraints that affect how the agents deal with certain situations.

Agents in a smart space are heavily resource-bounded because they are embedded in a physical world where all physical resources are scarce. This makes the coordination of multiple agent in a smart space all the more difficult because these physical constraints have to be taken into account. For that reason, an explicit resource management system is required in a smart space.

In this paper we present Rascal, a resource manager for the Metaglue agent platform. Metaglue [5] is a MAS developed at the MIT AI Lab for the Intelligent Room project. Rascal provides service mapping and resource access arbitration mechanisms for Metaglue agents. Rascal has been implemented and partially deployed for every-day use. Some of its advanced features are still being tested and optimized for speed.

1.1 Definitions

What is a resource manager for a smart space We believe a resource manager should be capable of performing two fundamental tasks: *resource mapping* and *arbitration.*

Resource mapping (i.e. match-making) is the process of deciding what resources can be used to satisfy a specific request.

Arbitration is ensuring that, at a minimum, resources are not being used beyond their capacities. Ideally, arbitration ensures optimal, or nearly optimal, use of scarce resources via appropriate allocation of resources to requests.

This paper is concerned with the management of high-level resources. As opposed to OS level management (memory, files, etc.) and load-balancing computationally intensive agents over multiple machines, these high-level resources include physical devices and large software components, for example, projectors, multiplexors, wires, displays, modems, user attention, software programs, screen real estate, sound input and output devices, CD players, drapes, and lamps.

For clarity, we define some potentially ambiguous terms that are used throughout the remainder of this paper:

Metaglue Metaglue [5, 10, 11] is the MAS forming the software base for all work at the Intelligent Room Project. Unlike most MAS, Metaglue provides infrastructure for close-coupling of agents (that is, it facilitates direct method calls) in addition to a message passing mechanism in order to enable faster communication among agents. Metaglue is intended for use in environments where most agents are physically close and thus good network connectivity can be assumed. Metaglue makes it easy to coordinate the startup and running of agents on any number of machines with different operating systems.

Metaglue agents are collected into "societies" which are distinct name-spaces for multiple users and spaces. A new communication and discovery model is currently being developed for inter-society communication.

Agent Agents are distinct object instances capable of providing services and making requests of the resource manager. This means agents themselves are considered to be a type of resource because they provide services (see below).

Device A physical or logical device is something akin to a projector, screen, or user-attention; devices are often, but not necessarily, represented by agents. Devices provide services and therefore are resources.

Service Services are provided by agents and devices; a single agent or device can provide more than one service and any kind of service can be provided by a number of agents or devices. For example, the `ShortTextOutput` service can be provided by the on-wall display, scrolling LED sign or a text-to-speech program. An A/V receiver is a provider of a number of services, such as an amplifier, an audio multiplexor and a radio receiver.

Resource A resource is a provider of a service. Both agents and physical devices are resources. For example, a physical LED sign is a resource (providing the LED sign hardware service) obtained and used by the `LEDSignText` Agent, which is in turn a resource (providing `TextOuput` service and `LEDSign` service) that can be obtained and used by any other agent needing those services.

2 Summary of design requirements

This section summarizes the essential requirements for designing a high-level resource management system for a smart space. Space permits only a brief overview; potential design issues are discussed in more detail in [7]. In particular, the needs for on-demand agent startup and "smart re-allocations" are motivated more extensively in [7].

2.1 Closed system assumption

We assume that Rascal will work in a closed system, i.e. one where all agents can be assumed to be trusted (but where agents can appear or leave dynamically). We can make this assumption without reducing the scalability of the system by dividing agents into *societies* . An agent society is a collection of agents that act on behalf of a single entity, such as a physical space, a person, a group of people, an institution, an information store, etc. Rascal's job is to coordinate use of resources within a society.

In cases where agents from one society need to access resources owned by a different society, a resource manager of one society can make requests of the resource manager from the other one. The resource manager from the society that owns the resource is the sole owner of the resource and can decide to take it back at any moment if necessary. The negotiation for resources among a number of societies is a somewhat different problem from managing resources within a society. For one thing, this is now an open system and access control mechanisms need to be put in place to ensure that all requesters act within their authority.

Extending our resource management system to a world with multiple societies requires having an access control system in place (see [7] for discussion) and is not covered here because this work is still in a preliminary phase.

A most common kind of situation where one society needs to make resource request of another is one in which agents acting on behalf of the user need resources to communicate information to the user. Agents acting on behalf of the user belong to one society and those controlling the space, belong to another (as in Figure 1). User's society usually will not contain physical devices and thus if, for example, an email alert agent acting on my behalf needs to tell me that a new mail has arrived for me, it will need resources (such as speech output or a display) from my office to pass the message on to me. In such situation, the email alert agent will still make a resource request of the resource manager in my society. My resource manager, seeing that it has no good resources on its own, will make a request of the resource manager of my office's society. My office will then decide whether to fulfill the request and if so, how.

2.2 Self-interested agents

Although we assume that all agents are trusted, we also assume that they are self-interested. All agents admitted to a society are assumed to

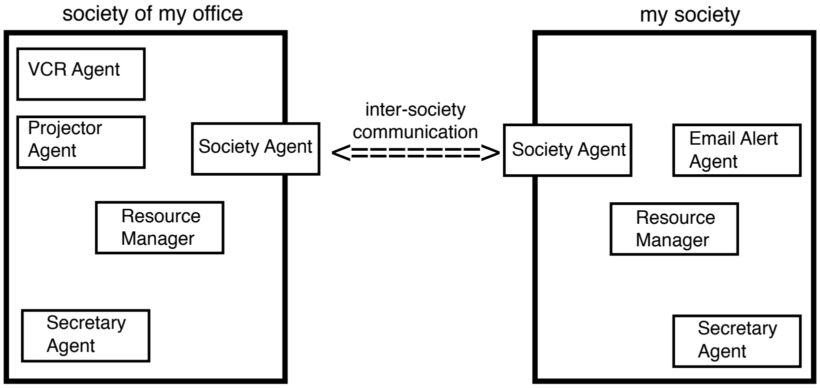


Fig. 1. Our view of the world: a society is viewed as a closed system. All agents within a society are trusted yet self-interested. A world composed of many societies is, on the other hand, viewed as an open system where, to an extent, every society can be viewed as a single agent providing a number of capabilities.

be truthful in that they do not misrepresent their needs and capabilities. They are, however, concerned solely with performing the task or tasks they were built for. For example, an agent that controls the flow of a presentation is only concerned with ensuring that slides are visible on an appropriate display; it has no knowledge of how its actions affect the abilities of other agents to perform their tasks. The assumpiton that agents have no knowledge of their peers allows a more modular design of the system.

2.3 Tightly tied to external physical resources

A special characteristic of a MAS based in a smart space, as noted before, is that it is very tightly coupled to the physical resources within that space. At the simplest level, the number of physical displays in the space is a limiting factor that determines how many visual activities (such as web browsing, sketching, watching movies, writing email) can be performed simultaneously. At a deeper level, the layout of physical connections among devices also limits their use. For example, the output of a VCR may be connected to only one projector and a TV, while computers can be dynamically connected – via a multiplexor – to any of the available displays in the space. A resource management system, such as Rascal, is necessary to keep track of available resources and arbitrate among conflicting requests for those resources.

2.4 Reasoning about absent agents

In smart spaces components can be added or removed at any moment. More often than not, however, components that are available one day, are also available the next. We believe that our system should not only be able to cope with dynamic changes of configuration, but also that the stability and predictability of the physical environment should be used to the system's advantage. One consequence of a predictable enviroment is the plausibility of reasoning about agents even before they have been started. In other words, in smart spaces, agents can be started when needed using the resources that at a given moment can be spared. For example, when one of the users within a space needs to make a slide presentation, an appropriate agent will be started on an available computer that has the appropriate presentation software, available screen space, and can be connected to an on-wall display device (such as a projector). If another presentation is started at the same time, another available computer and display device will be chosen. On-demand agent startup allows the system to adapt to the current set of available resources and prevents the system designer from having to predict all possible configurations that might be required in a space (such as the unusual case where two presentations need to run simultaneously).

2.5 Need for smrrt Re-allocations

In our system, it happens frequently that a new request can only be satisfied by taking a resource away from a previously satisfied request. But that previous request does not have to be left resource-less – there is often an alternative resource that can be used to fill it. Suppose, for example, that I request to watch the news in an office equipped with an on-wall projector and a TV set (see Figure 2). The projector is assigned to the job because it produces the largest image and has the best resolution. Then, while watching the news, I decide to also access my email agent. This agent must use the projector because it is the only display that can be used by a computer. Therefore, the projector is taken away from the news agent; ideally, instead of stopping the news agent, Rascal moves it to the TV set.

3 Building Rascal

3.1 Centralized vs. distributed

Conceptually, Rascal is a centralized system. This decision was not made lightly, but we believe the advantages of a centralized system outweight its drawbacks (such as, e.g., being a single point of failure).

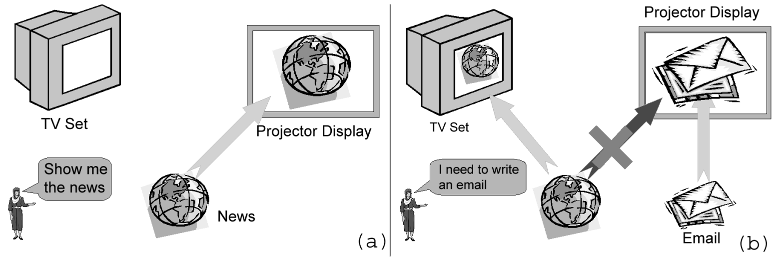


Fig. 2. Sample interaction. (a) user requests to see the news – on-wall projected display is allocated as the best resource for the task. (b) user accesses email; the only possible display for email is the on-wall projector previously allocated to the news agent. Instead of being stopped, the news agent is moved to a TV set.

Rascal was built as a separate centralized system primarily because it had to reason about absent agents. If we instead chose a distributed solution involving direct negotiation, all agents would have to be "alive" to be considered as candidates for a request. Also, a resource manager in an smart interactive, space has to be efficient. Rascal must make its decisions within a couple of seconds or less. A lot of inter-agent communication would make this goal nearly impossible.

Despite centralization, Rascal is actually not a single point of failure in the system. This is because two features of Metaglue make any agent nearly "invincible:" automatic restarting of agents ([11]) and persistent storage ([5]). If any agent dies, it will be restarted the next time any other agent tries to make a call to it. The dead agent will be restarted on any available computer, so even if the original computer hosting the agent fails, the agent will be restarted somewhere else. The persistent storage mechanism allows agents to save changes to their state whenever such changes occur. Consequently, if the agent dies and gets restarted, it can retrieve its state from before the failure and continue as if nothing had happened.

3.2 Structure

Rascal performs two major functions: *service mapping* and *arbitration* among requests for services (as defined in Section 1.1) and it is composed of three major parts: the knowledge base, the constraint satisfaction engine, and the framework for interacting with other Metaglue agents.

Service mapping is performed entirely by the knowledge-based component of Rascal. Arbitration begins in the knowledge-based part (where relative cost and utility of various resources are determined) but most of the work on arbitration is done by the constraint satisfaction engine.

The components for interacting with the rest of the Metaglue agents facilitate communication with service providers and requesters, and enable enforcement of Rascal's decision (i.e., taking previously allocated services away from requesters).

In the following sections, we present these major components of the system.

Representation and the knowledge base Upon startup, information about all available resources is loaded into Rascal's knowledge base (if more resources become available later on, they can be added dynamically). It is important to reiterate here that Rascal relies on all resources having descriptions of their needs and capabilities separate from the actual code. Those external descriptions contain a list of services that the resource can provide. Agents who provide services may in addition specify what other resources they will need in order to provide that service. For example, the `MessengerAgent` that provides a message delivery service will need one or more resources capable of providing text output service. Agents may also specify their startup needs, i.e. a list of requests that need to be fulfilled for the agent to exist. For example, an agent providing speech recognition service will need a computer, with appropriate speech recognition software installed, in order to be able to start and configure itself properly.

When Rascal considers candidates for a request, it not only makes sure that those candidate services are adequate and available – it also makes sure that the needs of those candidates can be satisfied, and so on recursively. The final selection of candidates for requests is performed by the constraint satisfaction engine. Therefore, the knowledge-based part evaluates all possible candidates for all possible requests. This request chaining proves to be extremely valuable: when the email alert agent, for example, requests a text output service, several different agents may be considered, including the LED sign and the speech output agents. The email alert agent may have its own preference as to what kind of rendition of the text output service it prefers. However, if the communication link with the actual LED sign is broken, the needs of the agent controlling the LED sign will not be satisfied and so it will not be assigned to the request.

Rascal's knowledge base is implemented in a rule-based system (JESS [6]) written in Java. The role of this component of the system is to find all plausible candidates for all requests. In addition, for each request-candidate pair, a measure of utility has to be calculated (see the next section).

In terms of representation, services provided by agents are described by the names of the Java interfaces that can be used to access them. For services provided by entities other than agents, similar hierarchical names are used (e.g. `hardware.Computer` for a computer). In addition, attribute-value pairs are used to describe services in more detail and to refine requests.

Cost-benefit analysis When resources are scarce, part of the arbitration process is deciding which requests are more important. This could be done with self-assigned priorities or economic models may be involved (e.g. [3]). In Rascal, self-assigned need levels are used in conjunction with the concept of utility of a service to the requester and its cost to others. This is a very simple and arbitrary scheme. It could easily be replaced by a different system should there be a need for that. This simple model is sufficient for the current implementation of Rascal, because of our assumption that all agents within a society can be trusted.

The basic assumption of this schema is that, given a request, each candidate resource has some utility to the requester. This utility depends on how badly the requester needs a particular request r fulfilled and on how well the resource s matches the request (Equation 1). A variety of monotonically increasing functions can be used as f_u.

$$utility(r, s) = f_u(need(r), match(r, s)) \tag{1}$$

The same method is used to calculate the utility of the already allocated resources. When a resource is taken from its current user, the system as a whole incurs cost equal to the utility of that resource to that user. Also, when a resource s_i, currently allocated to fulfill request r, is replaced with a different resource s_j, a cost is incurred. This cost is a sum of a fixed "change penalty" and the difference in utilities between the new allocation and the old one (if this difference is negative, it is set to zero) as shown in Equation 2.

$$cost(r, s_i, s_j) = changePenalty(r) + max\{0, utility(r, s_i) - utility(r, s_j)\} \tag{2}$$

The arbiter has to make sure that whenever it awards a resource to a new request, the cost of doing so should never exceed the utility of the awarded resources to the new requester.

Rascal provides a number of methods for calculating utilities and evaluating matches between requests and resources. Each resource or request description can also be accompanied by its own custom tools for performing those calculations.

Finding the right solution – The constraint satisfaction engine When the knowledge-based subsystem selects and rates all candidates for requests, a constraint satisfaction engine (CSE) is invoked to find an optimal or nearly optimal configuration that fulfills the new request without breaking any of the previous assignments.

Rascal uses a Java-based CSE (JSolver [4]) in order to enable close coupling with its other components. In order to find the right solution, a number of constraints and heuristics are involved:

- respecting limits – there are limits on how many requests can share a service.
- only some requests need to be satisfied – CSE needs to find services only for some of the requests that it knows about: the newly made request, the needs of the services assigned to satisfy this new request and all the previously satisfied requests.
- preference to local solutions – As explained in Section 2.5, it is sometimes necessary to change the assignment to a previously satisfied request. However, it is necessary to minimize such changes to the absolute minimum. Rascal's CSE has been set up in such a way that changes to old requests are only made as a last resort and have to be limited in scope. That is, it should not be possible for a new request to cause changes to a large number of other assignments. For that reason, Rascal's CSE uses following heuristics:
    - the first service considered for any previously satisfied request is the service previously allocated to the request;
    - if a different service has to be assigned, the cost of service substitution is calculated and added to the overall cost of the current new request – if the cost exceeds a preset limit, CSE backtracks;
    - the CSE is run several times, each time with a different limit to the overall cost: the first time CSE runs, the limit is set to zero in hope that a solution can be found that does not disturb any of the previously assigned requests. If this fails, the CSE is run again with a higher limit. The process is repeated until a solution is found or until the CSE is ran with a limit equal to the need of this request. In Rascal, the cost of satisfying a request cannot exceed the need to have it satisfied.

Rascal-Metaglue connection There are two major components to the Rascal-Metaglue connection mechanism: the `RascalAgent` and the `ManagedAgent`. The former makes Rascal's methods available to the rest of the Metaglue agents. The latter is a simple implementation of a Metaglue agent that all other "managed" agents inherit from. That is, all agents that want to make their services available through Rascal, or that wish to make requests through it.

4 Related work

The Facilitator Agent in Open Agent Architecture (OAA) [8] performs *task* not *resource* management. Implicit in the OAA design is the assumption that each agent has sole control over all of the resources it might need.

Applications in Hive [9] agent platform are created by explicitly connecting various components together. Thus resource conflicts are diminished because connections among agents are long-lived and predesigned, contrary to the on-demand configurations created within Rascal-enhanced Metaglue.

Jini [2] is a framework with a number of discovery and description tools but no arbitration capabilities. The arbitration component is supposed to be provided by the user.

Intentional Naming System (INS) [1] provides an extensive naming mechanism and a mechanism for choosing the best available service but it does not provide explicit arbitration mechanisms or tools for smart re-allocations.

5 Contributions

Multi-Agent Systems constitute a very powerful programming paradigm. Applying MAS to new domains often poses a number of challenges. This paper shows how the MAS approach can be applied in the domain of smart spaces, where agent coordination is constrained by the availability of physical resources. Rascal—an implememnted and tested tool for managing such resources—is presented.

6 Acknowledgments

Luke Weisman did most of the fundamental work on the Rascal-Metaglue interface. He has also done a lot of research on other approaches to resource management in Metaglue-based MAS. Dr. Howard Shrobe has provided invaluable advice throughout the duration of the project. Finally, the author would like to thank Mark Foltz for his thorough and insightful comments on this paper.

References

1. W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC, December 1999.
2. Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, and Ann Wollrath. *The Jini Specification*. Addison-Wesley, Reading, MA, 1999.
3. Jonathan Bredin, David Kotz, Daniela Rus, Rajiv T. Maheswaran, agri Imer, and Tamer Basar. A market-based model for resource allocation in agent systems. In Franco Zambonelli, editor, *Coordination of Internet Agents*. Springer-Verlag, 2000.
4. Hon Wai Chun. Constraint programming in java with jsolver. In *First International Conference and Exhibition on The Practical Application of Constraint Technologies and Logic Programming*, London, April 1999.
5. Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metaglue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.
6. Ernest J. Friedman-Hill. Jess, the java expert system shell. Technical Report Technical Report SAND98-8206, Sandia National Laboratories, 1997.
7. Krzysztof Gajos, Luke Weisman, and Howard Shrobe. Design principles for resource management systems for intelligent spaces. In *Proceedings of The Second International Workshop on Self-Adaptive Software*, Budapest, Hungary, 2001. To appear.
8. David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128, January-March 1999.
9. Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian, and Pattie Maes. Hive: Distributed agents for networking things. In *Proceedings of ASA/MA'99, the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents*, August 199.
10. Brenton Phillips. Metaglue: A programming language for multi agent systems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.
11. Nimrod Warshawsky. Extending the metaglue multi agent system. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

Hybrid Systems: Computation and
Control: 4th International Workshop
(HSCC'01), Rome, Italy, in Lecture
Notes in Computer Science, Vol. 2034,
Springer-Verlag, pp. 403–417, 2001

Hybrid I/O Automata Revisited

Nancy Lynch,[1] * Roberto Segala[2]** and Frits Vaandrager[3]***

[1] MIT Laboratory for Computer Science
`lynch@theory.lcs.mit.edu`
[2] Dipartimento di Matematica, Università di Bologna
`segala@cs.unibo.it`
[3] Computing Science Institute, University of Nijmegen
`fvaan@cs.kun.nl`

Abstract. In earlier work, we developed a mathematical *hybrid I/O automaton (HIOA)* modeling framework, capable of describing both discrete and continuous behavior. This framework has been used to analyze examples of automated transportation systems, intelligent vehicle highway systems, air traffic control systems, and consumer electronics applications. Here, we reconsider the basic definitions of the HIOA framework, in particular, the dual use of external variables for discrete and continuous communication. We present a new HIOA model that is simpler than the earlier model, due to a clearer separation between discrete and continuous activity.

1 Introduction

Recent years have seen a rapid growth of interest in *hybrid systems*—systems that contain both discrete and continuous components, typically computers interacting with the physical world. Such systems are used in many application domains, including automated transportation, avionics, automotive control, process control, robotics, and consumer electronics. Motivated by a desire to describe and reason carefully about such applications, we are continuing our efforts to adapt techniques from computer science to the setting of hybrid systems.

---

* Supported by AFOSR F49620-00-1-0097, F49620-97-1-0337; NTT MIT9904-12; NSF ACI-9876931, CCR-9909114, CCR-9804665; PATH 1784-18454LD.

** Supported by MURST project TOSCA.

*** Supported by Esprit Project 26270, Verification of Hybrid Systems (VHS).

In our previous work in this area, we developed a mathematical *hybrid I/O automaton* modeling framework [15, 16]. This framework supports description and analysis of hybrid systems using powerful methods of *parallel composition* and *levels of abstraction.* We also proved sufficient conditions for hybrid I/O automata to be *receptive*, which means that they allow time to advance to infinity independently of the input provided by the environment. We and others have used this framework to analyze examples of automated transportation systems [18, 13, 23, 22, 14, 10], intelligent vehicle highway systems [6, 12], air traffic control systems [11, 9], and consumer electronics systems [4].

In this paper, we present a *new hybrid I/O automaton model* that is considerably simpler than the earlier model, yet supports similar description and analysis methods and similar receptivity theorems. The main simplification is a clearer separation between the notions of discrete and continuous communication. We arrived at this separation as a result of reconsidering the relationship between the computer science notion of shared variable communication and the control theory notion of continuous flow across component boundaries.

Levels of abstraction, compositionality, and receptiveness for hybrid systems have also been addressed by Alur and Henzinger [2, 3] in their work on reactive modules. However, reactive modules communicate only via shared variables, and not via shared actions. In [3], a definition of receptiveness similar to the one in [15, 16] is proposed, and is shown to be preserved by composition. However, in [3], no circular dependencies ("feedback loops") are allowed among the continuous variables of the components, a restriction that greatly simplifies the analysis.

The rest of this paper is organized as follows. Section 2 defines notions that are useful for describing the behavior of hybrid systems: trajectories and hybrid sequences. Section 3 contains the theory for the *hybrid automaton (HA)* model, which has all of the structure of the HIOA model except for the division of external actions and variables into inputs and outputs. Section 4 introduces inputs and outputs, and presents the basic theory for HIOAs. Section 5 presents the new theory of receptiveness, including the main theorem, Theorem 7, stating that receptiveness is preserved by composition under certain compatibility conditions. Section 6 describes sufficient conditions for these compatibility conditions to hold, and in particular, describes Lipschitz automata.

2 Describing hybrid behavior

In this section, we give basic definitions that are useful for describing discrete and continuous system behavior, including discrete and continuous state changes, and discrete and continuous flow of information over component boundaries. Throughout this paper, we fix a *time axis* T, which is a compact subgroup of $(\mathsf{R}, +)$, the real numbers with addition.

2.1 Static and dynamic types

We assume a universal set V of *variables.* A variable represents either a location within the state of a system component, or a location where information flows from one system component to another. For each variable, we assume both a *(static) type*, which gives the set of values it may assume, and a *dynamic type*, which gives the set of trajectories it may follow. Our motivation for introducing dynamic types is that this allows us to define input enabling for hybrid I/O automata: if v is an input variable of HIOA $\mathcal{A}$ then, roughly speaking, we require that $\mathcal{A}$ accepts each input signal on v, as long as it respects the dynamic type of v. Since we are in a hybrid setting where discrete transitions may change the state at any time, elements of a dynamic type may contain (countably many) "discontinuities". Formally, we assume for each variable v:

- $\mathit{type}(v)$, the *(static) type* of v. This is a set of values.
- $\mathit{dtype}(v)$, the *dynamic type* of v. This is a set of functions from left-closed intervals of T to $\mathit{type}(v)$ that is closed under the following operations:
  1. (Time shift) For each $f \in \mathit{dtype}(v)$ and $t \in \mathsf{T}$, $f + t \in \mathit{dtype}(v)$. Here $f + t$ is the function given by $(f + t)\ (t') = f(t' - t)$.
  2. (Subinterval) For each $f \in \mathit{dtype}(v)$ and each left-closed interval $J \subseteq \mathit{dom}(f)$, $f \lceil J \in \mathit{dtype}(v)$. Here $f \lceil J$ is the function obtained by restricting the domain of f to J.
  3. (Pasting) For each sequence $f_0, f_1, f_2, \ldots$ of functions in $\mathit{dtype}(v)$ such that (a) the domain of each f_i, except possibly for the last one, is right-closed, (b) for each nonfinal index i, $\max(\mathit{dom}(f_i)) = \min(\mathit{dom}(f_{i+1}))$, the function f given by $f(t) \triangleq f_i(t)$, where i is the smallest index with $t \in \mathit{dom}(f_i)$, is in $\mathit{dtype}(v)$.

Example 1. For any variable v, the set C of constant functions from a left-closed interval to $\mathit{type}(v)$ is closed under time shift and subintervals. If the dynamic type of v is obtained by closing C under the pasting operation, then v is called a *discrete* variable, as in [19]. If we take $\mathsf{T} = \mathsf{R}$ and $\mathit{type}(v) = \mathsf{R}$, then other examples of dynamic types can be obtained by taking the pasting closure of the set of continuous or smooth functions, the set of integrable functions, or the set of measurable locally essentially bounded functions. The set of all functions from left-closed intervals of R to R is also a dynamic type.

In practice, dynamic types are often defined via pasting closure of a class of continuous functions. In these cases the elements of dynamic types are continuous from the left. Elsewhere in the literature on hybrid systems one often encounters functions that are continuous from the right (see, e.g., [8]). To some extent, the choice of how to define function values at discontinuities is arbitrary. An advantage of our choice is a nice correspondence between concatenation and prefix ordering of trajectories (see Lemma 2). In the rest of this paper, when we say that the dynamic type of a variable v equals S, we actually mean that the dynamic type of v is obtained by applying the above closure operations to S.

2.2 Trajectories

In this subsection, we define the notion of a *trajectory*, define operations on trajectories, and prove simple properties of trajectories and their operations. A trajectory is used to model the evolution of a collection of variables over an interval of time.

Basic definitions Let V be a set of variables, that is, a subset of V. A *valuation* $\mathbf{v}$ for V is a function that associates to each variable $v \in V$ a value in $\mathit{type}(v)$. We write $\mathit{val}(V)$ for the set of valuations for V. Let J be a left-closed interval of T with left endpoint equal to 0. Then a J*-trajectory* for V is a function $\tau : J \to \mathit{val}(V)$, such that for each $v \in V$, $\tau \downarrow v \in \mathit{dtype}(v)$. Here $\tau \downarrow v$ is the function with domain J defined by $(\tau \downarrow v)(t) = \tau(t)(v)$.

We say that a J-trajectory is *finite* if J is a finite interval, *closed* if J is a (finite) closed interval, and *full* if $J = \mathsf{T}^{\geq 0}$. A *trajectory* for V is a J-trajectory for V, for any J. We write $\mathit{trajs}(V)$ for the set of all trajectories for V. For T a set of trajectories, $\mathit{finite}(T)$, $\mathit{closed}(T)$ and $\mathit{full}(T)$ denote the subsets of finite, closed and full trajectories in T, respectively. A trajectory with domain $[0, 0]$ is called a *point* trajectory. If $\mathbf{v}$ is a valuation then $\wp(\mathbf{v})$ denotes the point trajectory that maps 0 to $\mathbf{v}$.

If τ is a trajectory then $\tau.\mathit{ltime}$, the *limit time* of τ, is the supremum of $\mathit{dom}(\tau)$. Similarly, we define $\tau.\mathit{fval}$, the *first valuation* of τ, to be $\tau(0)$, and if τ is closed, we define $\tau.\mathit{lval}$, the *last valuation* of τ, to be $\tau(\tau.\mathit{ltime})$. For τ a trajectory and $t \in \mathsf{T}^{\geq 0}$, we define $\tau \trianglelefteq t \triangleq \tau \lceil [0,t]$, $\tau \triangleleft t \triangleq \tau \lceil [0,t)$, and $\tau \trianglerighteq t \triangleq (\tau \lceil [t,\infty)) - t$. Note that the result of applying the above operations is always a trajectory, except when the result is a function with an empty domain. By convention, $\tau \trianglelefteq \infty \triangleq \tau$ and $\tau \triangleleft \infty \triangleq \tau$.

Prefix ordering Trajectory τ is a *prefix* of trajectory υ, denoted by $\tau \leq \upsilon$, if τ can be obtained by restricting υ to a non-empty, downward closed subset of its domain. Formally, $\tau \leq \upsilon$ iff $\tau = \upsilon \lceil \mathit{dom}(\tau)$. For T a set of trajectories for V, $\mathit{pref}(T)$ denotes the *prefix closure* of T. We say that T is *prefix closed* if $T = \mathit{pref}(T)$.

The following lemma gives a simple domain theoretic characterization of the set of trajectories over a given set V. (See [7] for basic definitions and results on complete partially ordered sets, (cpo's)).

Lemma 1. *Let V be a set of variables. Then the set $\mathit{trajs}(V)$ of trajectories for V, together with the prefix ordering $\leq$, is an algebraic cpo whose compact elements are the closed trajectories.*

Concatenation The concatenation of two trajectories is obtained by taking the union of the first trajectory and the function obtained by shifting the domain of the second trajectory until the start time agrees with the limit time of the first trajectory; the last valuation of the first trajectory, which may not be the same as the first valuation of the second trajectory, is the one that appears in the concatenation. Formally, let τ, υ be trajectories, with τ closed. Then the *concatenation* is the function given by $\tau \frown \upsilon \ \triangleq \ \tau \cup (\upsilon \lceil (0,\infty) + \tau.\mathit{ltime})$. Using the closure of dynamic types under time shift and pasting, it follows that $\tau \frown \upsilon$ is a trajectory. Observe that $\tau \frown \upsilon$ is finite (resp. closed, full) iff υ is finite (resp. closed, full). Observe also that concatenation is associative.

The following lemma, which is easy to prove, shows the close connection between concatenation and the prefix ordering.

Lemma 2. *Let τ, υ be trajectories with τ closed. Then $\tau \leq \upsilon$ iff there exists a trajectory τ' such that $\tau \frown \tau'$.*

Note that if $\tau \leq \upsilon$, then the trajectory τ' such that $\upsilon = \tau \frown \tau'$ is unique except that it has an arbitrary value for $\tau'.\mathit{fval}$. Note also that the "$\Leftarrow$" implication would not hold if the first valuation of the second argument, rather than the last valuation of the first argument, were used in the concatenation.

Using a limit construction, we can generalize the definition of concatenation for any (finite or countably infinite) number of arguments. Let $\tau_0, \tau_1, \tau_2, \ldots$ be a (finite or infinite) sequence of trajectories, such that τ_i is closed for each nonfinal index i. Define trajectories $\tau'_0, \tau'_1, \tau'_2, \ldots$ by $\tau'_i \triangleq \tau_0 \frown \tau_1 \frown \cdots \frown \tau_i$. We define the *concatenation* $\tau_0 \frown \tau_1 \frown \tau_2 \ldots$ to be $\lim_{i \to \infty} \tau'_i$. It is easy to prove that $\tau_0 \frown \tau_1 \frown \tau_2 \ldots$ is a trajectory.

2.3 Hybrid sequences

In this subsection, we introduce the notion of a *hybrid sequence*, which is used to model a combination of changes that occur instantaneously and changes that occur over intervals of time. Our definition is parameterized by a set A of *actions*, which are used to model instantaneous changes and instantaneous synchronization with the environment, and a set V of *variables*, which are used to model changes over intervals and continuous interaction. We also define some special kinds of hybrid sequences and operations on hybrid sequences.

Basic definitions An (A, V)-*sequence* is a finite or infinite alternating sequence $\alpha = \tau_0\ a_1\ \tau_1\ a_2\ \tau_2 \cdots$, where (1) each τ_i is a trajectory in $\mathit{trajs}(V)$, (2) each a_i is an action in A, (3) if α is a finite sequence then it ends with a trajectory, and (4) if τ_i is not the last trajectory in α then $\mathit{dom}(\tau_i)$ is closed. We define a *hybrid sequence* to be an (A, V)-sequence for some A and V.

Since the trajectories in a hybrid sequence can be point trajectories, our notion of hybrid sequence allows a sequence of discrete actions to occur at the same real time, with corresponding changes of state.

If α is a hybrid sequence, with notation as above, then we define the *first valuation* of α, $\alpha.\mathit{fval}$, to be $\tau_0.\mathit{fval}$, and we define the *limit time* of α, $\alpha.\mathit{ltime}$, to be $\sum_i \tau_i.\mathit{ltime}$. A hybrid sequence α is defined to be:

- *time-bounded* if $\alpha.\mathit{ltime}$ is finite.
- *admissible* if $\alpha.\mathit{ltime} = \infty$.
- *closed* if α is a finite sequence and the domain of its final trajectory is a closed interval. In this case we define the *last valuation* of α, $\alpha.\mathit{lval}$, to be $\mathit{last}(\alpha).\mathit{lval}$.
- *Zeno* if α is neither closed nor admissible, that is, if α is time-bounded and is either an infinite sequence, or else a finite sequence ending with a trajectory whose domain is right-open.

Prefix ordering We say that (A, V)-sequence $\alpha = \tau_0 a_1 \tau_1 \ldots$ is a *prefix* of (A, V)-sequence $\alpha' = \tau'_0 a'_1 \tau'_1 \ldots$, denoted by $\alpha \leq \alpha'$, if either $\alpha = \alpha'$, or α is a finite sequence ending in some τ_k; $\tau_i = \tau'_i$, and $a_{i+1} = a'_{i+1}$ for every i, $0 \leq i < k$; and $\tau_k \leq \tau'_k$. Like the set of trajectories over V, the set of (A, V)-sequences is a cpo.

Lemma 3. *The set of (A, V)-sequences together with the prefix ordering $\leq$ is an algebraic cpo with as compact elements the set of closed (A, V)-sequences.*

Restriction Let A, A' be sets of actions and V, V' sets of variables. The (A', V')-restriction of an (A, V)-sequence is obtained by projecting the trajectories on the variables in V', removing the actions not in A', and concatenating the adjacent trajectories.

Lemma 4. *Restriction is a continuous operation with respect to prefix ordering.*

Concatenation Suppose α and α' are (A, V)-sequences, with α closed. Then the *concatenation* is the (A, V)-sequence given by

$$\alpha \frown \alpha' \triangleq \mathit{init}(\alpha)\ (\mathit{last}(\alpha) \frown \mathit{head}(\alpha'))\ \mathit{tail}(\alpha').$$

(If σ is a nonempty sequence then $\mathit{head}(\sigma)$ denotes the first element of σ and $\mathit{tail}(\sigma)$ denotes σ with its first element removed; if σ is finite, then $\mathit{last}(\sigma)$ denotes the last element of σ and $\mathit{init}(\sigma)$ denotes σ with its last element removed.)

Lemma 5. *Let α, α' be (A, V)-sequences with α closed. Then $\alpha \leq \alpha'$ iff there exists and (A, V)-sequence α'' such that $\alpha' = \alpha \frown \alpha''$.*

Note that if $\alpha \leq \alpha'$, then the (A, V)-sequence α'' such that $\alpha' = \alpha \frown \alpha''$ is unique except that it has an arbitrary value in $\mathit{val}(V)$ for $\alpha''.\mathit{fval}$.

Based on Lemma 5 and Lemma 3, we can extend concatenation to infinitely many (A, V)-sequences as follows. Let $\alpha_1, \alpha_2, \ldots$ be an infinite sequence of closed (A, V)-sequences. Then define the *concatenation* $\alpha_1 \frown \alpha_2 \frown \cdots$ to be $\lim_{i \to \infty} \alpha'_i$, where $\alpha'_i = \alpha_1 \frown \alpha_2 \frown \cdots \frown \alpha_i$.

3 Hybrid Automata

As a preliminary step toward defining hybrid I/O automata, we first define a slightly more general *hybrid automaton* model. Hybrid automata classify actions as external and internal, but do not further subdivide the external actions into input and output actions. Likewise, they classify variables as external and internal. The input/output distinction is added in Section 4. In addition to defining hybrid automata, we here define an implementation relation between hybrid automata and a composition operation.

3.1 Definition of Hybrid Automata

A *hybrid automaton (HA)* $\mathcal{A} = (W, X, \Theta, E, H, \mathcal{D}, \mathcal{T})$ consists of:

- A set W of *external variables* and a set X of *internal variables*, disjoint from each other. We call a valuation $\mathbf{x}$ for X a *state*, and we refer to $val(X)$ as the set of states of $\mathcal{A}$. We write $V \triangleq W \cup X$. Given a valuation $\mathbf{v}$ for V, we denote by $state(\mathbf{v})$ the state $\mathbf{v} \lceil X$.
- A nonempty set $\Theta \subseteq val(X)$ of *start states*.
- A set E of *external actions* and a set H of *internal actions*, disjoint from each other. We write $A \triangleq E \cup H$ and let $a, b, \ldots$ range over A.
- A set $\mathcal{D} \subseteq val(X) \times A \times val(X)$ of *discrete transitions*. We use $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ as shorthand for $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$. We sometimes drop the subscript, and write $\mathbf{x} \xrightarrow{a} \mathbf{x}'$, when $\mathcal{A}$ should be clear from the context.
- A set $\mathcal{T}$ of trajectories for V. Given a trajectory $\tau \in \mathcal{T}$ we denote $\tau.fval \lceil X$ by $\tau.fstate$, and, if τ is closed, $\tau.lval \lceil X$ by $\tau.lstate$. We require that the following axioms hold:
  - **T1** (Prefix closure) For every $\tau \in \mathcal{T}$ and every $\tau' \leq \tau$, $\tau' \in \mathcal{T}$.
  - **T2** (Suffix closure) For every $\tau \in \mathcal{T}$ and every $t \in dom(\tau)$, $\tau \trianglerighteq t \in \mathcal{T}$.
  - **T3** (Concatenation closure) Let $\tau_0, \tau_1, \tau_2, \ldots$ be a sequence of trajectories in $\mathcal{T}$ such that, for each nonfinal index i, τ_i is closed and $\tau_i.lstate = \tau_{i+1}.fstate$. Then $\tau_0 \frown \tau_1 \frown \tau_2 \ldots \in \mathcal{T}$.

Axioms **T1-3** express some natural closure properties on the set of trajectories that we need for our results about parallel composition. In a composed system, any trajectory of any component may be interrupted at any moment by a discrete transition of another component. Axiom **T1** ensures that the part of the trajectory up to the discrete transition is a trajectory, and axiom **T2** ensures the remainder is a trajectory. Axiom **T3** is required because the environment of a hybrid automaton, as a result of internal discrete transitions, may change its continuous dynamics repeatedly, and the automaton must be able to follow this behavior. Even without performing discrete transitions itself, a hybrid automaton must be able to follow this type of behavior of its environment. In the earlier definition of hybrid automata presented in [15, 16], we used a special stuttering action e in place of axiom **T3**; this gave rise to technical complications.

Another major difference between our new definition and the earlier one is that the external variables are no longer considered to be part of the state; thus, for instance, the discrete transitions do not depend on the values of these variables. Analogous to the way in which external actions can be used to model synchronization of discrete transitions of different components, external variables allow us to model synchronization of continuous activity ("flow") between components. Because the external actions and external variables are not part of the state, we think of them as "ephemeral".

We often denote the components of a HA $\mathcal{A}$ by $W_{\mathcal{A}}$, $X_{\mathcal{A}}$, $\Theta_{\mathcal{A}}$, $E_{\mathcal{A}}$, etc, and the components of a HA $\mathcal{A}_i$ by W_i, X_i, Θ_i, E_i, etc. We sometimes omit these subscripts, where no confusion seems likely.

3.2 Executions and traces

We now define execution fragments, executions, trace fragments, and traces, which are used to describe automaton behavior.

An *execution fragment* of a HA $\mathcal{A}$ is an (A, V)-sequence $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \cdots$, where (1) each τ_i is a trajectory in $\mathcal{T}$, and (2) if τ_i is not the last trajectory in α then $\tau_i.\mathit{lstate} \overset{a_{i+1}}{\rightarrow} \tau_{i+1}.\mathit{fstate}$. An execution fragment records all the instantaneous, discrete state changes that occur during a specific evolution of a system, as well as the state changes and external variable changes that occur while time advances. We write $\mathit{frags}_{\mathcal{A}}$ for the set of all execution fragments of $\mathcal{A}$.

If α is an execution fragment, with notation as above, then we define the *first state* of α, $\alpha.\mathit{fstate}$, to be $\mathit{state}(\alpha.\mathit{fval})$, or equivalently, $\tau_0.\mathit{fstate}$. An execution fragment α is defined to be an *execution* if $\alpha.\mathit{fstate}$ is a start state, that is, is in Θ. We write $\mathit{execs}_{\mathcal{A}}$ for the set of all executions of $\mathcal{A}$.

If α is a closed execution fragment then we define the *last state* of α, $\alpha.\mathit{lstate}$, to be $\mathit{state}(\alpha.\mathit{lval})$, or equivalently, $\mathit{last}(\alpha).\mathit{lstate}$. A state of $\mathcal{A}$ is *reachable* if it is the last state of some closed execution of $\mathcal{A}$.

Lemma 6. *Let α and α' be execution fragments of $\mathcal{A}$ with α closed, and such that $\alpha.\mathit{lstate} = \alpha'.\mathit{fstate}$. Then $\alpha ^\frown \alpha'$ is an execution fragment of $\mathcal{A}$.*

Lemma 7. *Let α and α' be execution fragments of $\mathcal{A}$ with α closed. Then $\alpha \leq \alpha'$ iff there is an execution fragment α'' such that $\alpha' = \alpha ^\frown \alpha''$.*

The trace of an execution fragment records the external actions and the evolution of external variables. Formally, if α is an execution fragment, then the *trace* of α, denoted by $\mathit{trace}(\alpha)$, is the (E, W)-restriction of α. A *trace fragment* of a hybrid automaton $\mathcal{A}$ *from* a state $\mathbf{x}$ of $\mathcal{A}$ is a trace that arises from an execution fragment of $\mathcal{A}$ whose first state is $\mathbf{x}$. We write $\mathit{tracefrags}_{\mathcal{A}}(\mathbf{x})$ for the set of trace fragments of $\mathcal{A}$ from $\mathbf{x}$. Also, we define a *trace* of $\mathcal{A}$ to be a trace fragment from an initial state, that is, a trace that arises from an execution of $\mathcal{A}$, and write $\mathit{traces}_{\mathcal{A}}$ for the set of traces of $\mathcal{A}$.

Hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *comparable* if they have the same external actions and variables, that is, if $W_1 = W_2$ and $E_1 = E_2$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable then we say that $\mathcal{A}_1$ *implements* A_2, denoted by $\mathcal{A}_1 \leq \mathcal{A}_2$, if the traces of $\mathcal{A}_1$ are included among those of $\mathcal{A}_2$, that is, if $\mathit{traces}_{\mathcal{A}_1} \subseteq \mathit{traces}_{\mathcal{A}_2}$.

3.3 Simulation relations

Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs. A *simulation* from $\mathcal{A}$ to $\mathcal{B}$ is a relation $R \subseteq \mathit{val}(X_{\mathcal{A}}) \times \mathit{val}(X_{\mathcal{B}})$ satisfying the following conditions, for all states $\mathbf{x}_A$ and $\mathbf{x}_B$ of $\mathcal{A}$ and $\mathcal{B}$, respectively:

1. If $\mathbf{x}_A \in \Theta_A$ then there exists a state $\mathbf{x}_B \in \Theta_B$ such that $\mathbf{x}_A \mathrel{R} \mathbf{x}_B$.
2. If $\mathbf{x}_A \mathrel{R} \mathbf{x}_B$, $\mathbf{x}_A \overset{a}{\rightarrow}_{\mathcal{A}} \mathbf{x}'_A$ and $\tau = \mathit{trace}(\wp(\mathbf{x}_A)\ a\ \wp(\mathbf{x}'_A))$, then $\mathcal{B}$ has a closed execution fragment α with $\alpha.\mathit{fstate} = \mathbf{x}_B$, $\mathit{trace}(\alpha) = \mathit{trace}(\tau)$, and $\mathbf{x}'_A \mathrel{R} \alpha.\mathit{lstate}$.
3. If $\mathbf{x}_A \mathrel{R} \mathbf{x}_B$ and τ is a closed trajectory of $\mathcal{A}$ with $\mathbf{x}_A = \tau.\mathit{fstate}$ and $\mathbf{x}'_A = \tau.\mathit{lstate}$, then $\mathcal{B}$ has a closed execution fragment α with $\alpha.\mathit{fstate} = \mathbf{x}_B$, $\mathit{trace}(\alpha) = \mathit{trace}(\tau)$, and $\mathbf{x}'_A \mathrel{R} \alpha.\mathit{lstate}$.

Lemma 8. *Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs, and let R be a simulation from $\mathcal{A}$ to $\mathcal{B}$. Let $\mathbf{x}_A$ and $\mathbf{x}_B$ be states of $\mathcal{A}$ and $\mathcal{B}$, respectively, such that $\mathbf{x}_A \mathrel{R} \mathbf{x}_B$. Then $\mathit{tracefrags}_{\mathcal{A}}(\mathbf{x}_A) \subseteq \mathit{tracefrags}_{\mathcal{B}}(\mathbf{x}_B)$.*

Theorem 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs, and let R be a simulation from $\mathcal{A}$ to $\mathcal{B}$. Then $\mathit{traces}_{\mathcal{A}} \subseteq \mathit{traces}_{\mathcal{B}}$.*

3.4 Composition

We now introduce the operation of composition for hybrid automata, which allows an automaton representing a complex system to be constructed by composing automata representing individual system components. We prove that the composition operation respects our implementation relationship (inclusion of sets of traces). Our composition operation identifies actions and variables with the same name in different component automata. When any component automaton performs a step involving an action a, so do all component automata that have a in their signatures. Common variables are shared among the components.

We define composition as a partial, binary operation on hybrid automata. Since internal actions of an automaton $\mathcal{A}_1$ are intended to be unobservable by any other automaton $\mathcal{A}_2$, we do not allow $\mathcal{A}_1$ to be composed with $\mathcal{A}_2$ unless the internal actions of $\mathcal{A}_1$ are disjoint from the actions of $\mathcal{A}_2$. Also, we require disjointness of the internal variables of $\mathcal{A}_1$ and the variables of $\mathcal{A}_2$. Formally, we say that hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if for $i \neq j$, $X_i \cap V_j = H_i \cap A_j = \emptyset$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible then their *composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is defined to be the structure $\mathcal{A} = (W, X, \Theta, E, H, \mathcal{D}, \mathcal{T})$ where

- $W = W_1 \cup W_2$, $X = X_1 \cup X_2$, $E = E_1 \cup E_2$, $H = H_1 \cup H_2$.
- $\Theta = \{\mathbf{x} \in \mathit{val}(X) \mid \mathbf{x} \lceil X_1 \in \Theta_1 \wedge \mathbf{x} \lceil X_2 \in \Theta_2\}$.
- For each $\mathbf{x}, \mathbf{x}' \in \mathit{val}(X)$ and each $a \in A$, $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ iff for $i = 1, 2$, either (1) $a \in A_i$ and $\mathbf{x} \lceil X_i \xrightarrow{a}_i \mathbf{x}' \lceil X_i$, or (2) $a \notin A_i$ and $\mathbf{x} \lceil X_i = \mathbf{x}' \lceil X_i$.
- $\mathcal{T} \subseteq \mathit{trajs}(V)$ is given by $\tau \in \mathcal{T} \Leftrightarrow \tau \downarrow V_1 \in \mathcal{T}_1 \ \wedge \ \tau \downarrow V_2 \in \mathcal{T}_2$.

Proposition 1. *$\mathcal{A}_1 \| \mathcal{A}_2$ is a hybrid automaton.*

Theorem 2. *Suppose $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}$ are HAs with $\mathcal{A}_1 \leq \mathcal{A}_2$, and suppose that each of $\mathcal{A}_1$ and $\mathcal{A}_2$ is compatible with $\mathcal{B}$. Then $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$.*

In the full version of this paper, we define two natural hiding operations on HAs, which hide external actions and external variables, respectively, and prove that these operations also respect the implementation preorder.

4 Hybrid I/O Automata

In this section we specialize the hybrid automaton model of Section 3 by adding a distinction between input and output.

4.1 Definition of Hyybrid I/O Automata

A *hybrid I/O automaton (HIOA)* $\mathcal{A}$ is a tuple $(\mathcal{H}, U, Y, I, O)$ where

- $\mathcal{H} = (W, X, \Theta, E, H, \mathcal{D}, \mathcal{T})$ is a hybrid automaton.
- U and Y partition W into *input* and *output* variables, respectively. Variables in $Z \triangleq X \cup Y$ are called *locally controlled*; as before we write $V \triangleq W \cup X$.

- I and O partition E into *input* and *output actions*, respectively. Actions in $L \triangleq H \cup O$ are called *locally controlled*; as before we write $A \triangleq E \cup H$.
- The following additional axioms are satisfied:
  E1 (Input action enabling)
  For all $\mathbf{x} \in \mathit{val}(X)$ and all $a \in I$ there exists $\mathbf{x}'$ such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$.
  E2 (Input flow enabling)
  For all $\mathbf{x} \in \mathit{val}(X)$ and $\upsilon \in \mathit{trajs}(U)$, there exists $\tau \in \mathcal{T}$ such that $\tau.\mathit{fstate} = \mathbf{x}, \tau \downarrow U \leq \upsilon$, and either
    1. $\tau \downarrow U = \upsilon$, or
    2. there exist $t \in \mathit{dom}(\tau)$ and $l \in L$ such that l is enabled from $\tau(t)$.

Input action enabling is the input enabling condition of ordinary I/O automata. Input flow enabling is a new corresponding condition for continuous interaction. It says that an HIOA should be able to accept any continuous input flow, either by letting time advance for the entire duration of the input flow, or by reacting with a locally controlled action after some part of the input flow has occurred.

An *execution* of an HIOA $\mathcal{A}$ is an execution of $\mathcal{H}_{\mathcal{A}}$. Similarly, a *trace* of $\mathcal{A}$ is a trace of $\mathcal{H}_{\mathcal{A}}$. Two HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *comparable* if their inputs and outputs coincide, that is, if $I_1 = I_2$, $O_1 = O_2$, $U_1 = U_2$, and $Y_1 = Y_2$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable, then $\mathcal{A}_1 \leq \mathcal{A}_2$ is defined to mean that the traces of $\mathcal{A}_1$ are included among those of $\mathcal{A}_2$: $\mathcal{A}_1 \leq \mathcal{A}_2 \triangleq \mathit{traces}_{\mathcal{A}_1} \subseteq \mathit{traces}_{\mathcal{A}_2}$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable HIOAs then $\mathcal{H}_1$ and $\mathcal{H}_2$ are comparable and $\mathcal{A}_1 \leq \mathcal{A}_2$ iff $\mathcal{H}_1 \leq \mathcal{H}_2$.

The definition of simulation for HIOAs is the same as for HAs, and the soundness result carries over immediately to the enriched setting.

4.2 Composition

The definition of composition for HIOAs builds on the corresponding definition for HAs, but also takes the input/output structure into account. Just as in the definition of compatibility for HAs, we do not allow an HIOA $\mathcal{A}_1$ to be composed with an HIOA $\mathcal{A}_2$ unless the internal actions and variables of $\mathcal{A}_1$ are disjoint from the actions and variables, respectively, of $\mathcal{A}_2$. In addition, in order that the composition operation might satisfy nice properties (such as Theorem 7), we require that at most one component automaton "controls" any given action or variable; that is, we do not allow $\mathcal{A}_1$ and $\mathcal{A}_2$ to be composed unless the sets of output actions of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint and the sets of output variables of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint.

If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible then their *composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is defined to be the tuple $\mathcal{A} = (\mathcal{H}, U, Y, I, O)$ where $\mathcal{H} = \mathcal{H}_1 \| \mathcal{H}_2$, $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $Y = Y_1 \cup Y_2$, $I = (I_1 \cup I_2) - (O_1 \cup O_2)$, and $O = O_1 \cup O_2$.

The definition of compatibility given above is not quite strong enough to imply that the composition of two HIOAs is actually an HIOA. Thus, we define a stronger notion and say that compatible HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *strongly compatible* if $\mathcal{A}_1 \| \mathcal{A}_2$ satisfies axiom **E2**. Strong compatibility implies that the reaction of the composed automaton to any input flow v must be the result of a deliberate reaction by either $\mathcal{A}_1$ or $\mathcal{A}_2$. That is, either both $\mathcal{A}_1$ and $\mathcal{A}_2$ accept v in its entirety, or one of the two reacts with a locally controlled action. No "time deadlock" is allowed due to incompatible reactions of $\mathcal{A}_1$ and $\mathcal{A}_2$.

Proposition 2. *The composition of two strongly compatible HIOAs is an HIOA.*

Theorem 3. *Suppose* $\mathcal{A}_1, \mathcal{A}_2$ *and* $\mathcal{B}$ *are HIOAs with* $\mathcal{A}_1 \leq \mathcal{A}_2$*, and each of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *is strongly compatible with* $\mathcal{B}$*. Then* $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$*.*

5 Receptive Hybrid I/O Automata

In this section we adapt the notion of receptiveness [20] to our new framework. Informally speaking, a system is receptive provided that it admits a strategy for resolving its nondeterministic choices that never generates infinitely many locally controlled actions in finite time. An important consequence of this definition is that a receptive HIOA has some response defined for any sequence of discrete and continuous input. We show that receptiveness is closed under composition. Because of the improvements in our new model, the treatment of receptiveness in this paper is simpler than that in [20]; however, we only address admissibility here, and not general liveness properties as in [20].

An execution fragment of an HIOA is *locally-Zeno* if it is Zeno and contains infinitely many locally controlled actions. An HIOA $\mathcal{A}$ is *locally-Zeno* if it has at least one locally-Zeno execution fragment. In the rest of the paper we will be interested mainly in *non-locally-Zeno* HIOAs, that is, HIOAs that are not locally-Zeno. We use non-locally-Zeno HIOAs as the basis for defining receptiveness.

Theorem 4. *Let* $\mathcal{A}_1$*,* $\mathcal{A}_2$ *be strongly compatible non-locally-Zeno HIOAs. Then* $\mathcal{A}_1 \| \mathcal{A}_2$ *is also non-locally-Zeno.*

Theorem 5. *Let $\mathcal{A}$ be a non-locally-Zeno HIOA. Then, for each (I, U)-sequence β and each state $\mathbf{x}$, there is an execution fragment α of $\mathcal{A}$ such that (1) $\alpha.fstate = \mathbf{x}$, (2) $\alpha \lceil (I, U) = \beta$.*

The property stated in Theorem 5 is known in the literature as *I/O feasibility* [17]; it implies that any finite execution can be extended to an admissible execution, no matter what the environment does.

A *strategy* for an HIOA $\mathcal{A}$ is an HIOA $\mathcal{A}'$ that differs from $\mathcal{A}$ only in that $\mathcal{D}' \subseteq \mathcal{D}$ and $\mathcal{T}' \subseteq \mathcal{T}$. A strategy $\mathcal{A}'$ for an HIOA $\mathcal{A}$ can be viewed as a nondeterministic memoryless strategy in the sense of [5, 20] that chooses some of the evolutions that are possible from each of the states of $\mathcal{A}$. The fact that the states of $\mathcal{A}$ and $\mathcal{A}'$ are the same ensures that $\mathcal{A}'$ chooses evolutions for every state $\mathbf{x}$ of $\mathcal{A}$.

We say that an HIOA is *receptive* if it has a non-locally-Zeno strategy.

Theorem 6. *A receptive HIOA is I/O feasible.*

Theorem 7. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible receptive HIOAs with two strongly compatible non-locally-Zeno strategies $\mathcal{A}'_1$ and $\mathcal{A}'_2$, respectively. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is a receptive HIOA with non-locally-Zeno strategy $\mathcal{A}'_1 \| \mathcal{A}'_2$.*

6 Sufficient conditions for strong compatibility

In order to apply Theorem 7, one has to establish that two strategies are strongly compatible. This is difficult in general since it requires checking compatibility between the continuous dynamics of two systems. However, for certain restricted classes of HIOAs, strong compatibility follows directly from compatibility.

6.1 HIOAs with restrictions on input variables

Our first example is the class of HIOAs without input variables. It is routine to verify that two HIOAs without input variables are strongly compatible iff they are compatible. From the perspective of classical control theory a system without input variables is uninteresting because it cannot be controlled; in a hybrid setting, however, a system without input variables can still interact with its environment via discrete input actions. *Linear hybrid automata* [1], for instance, have no input variables.

Another example is the class of *autistic* HIOAs—those for which the values of output variables do not depend on the values of input variables. Formally, an HIOA $\mathcal{A}$ is called *autistic* if for all $\tau \in \mathcal{T}$ and all $\upsilon \in \mathit{trajs}(U)$ such that $\mathit{dom}(\tau) = \mathit{dom}(\upsilon)$ there exists $\tau' \in T$ such that $\tau' \downarrow U = \upsilon$ and $\tau' \downarrow Y = \tau \downarrow Y$.

6.2 Lipschitz HIOAs

In this section, we define *Lipschitz HIOAs*, based on systems of differential equations using Lipschitz functions. We give examples of conditions on classes of Lipschitz HIOAs that imply strong compatibility. The ideas are derived from methods in the literature on control theory [21]. In control theory, continuous system behavior is typically defined using differential equations of the form:

$$D \triangleq \begin{cases} \dot{x} = f(x, u) \\ y = g(x) \end{cases}$$

where u, y, and x are the vectors of input, output, and state variables, respectively, together with a starting condition of the form $x(0) = x_0$.

To ensure that the system's behavior is defined, the differential equations must admit a solution for each possible starting condition. The following theorem from calculus gives sufficient conditions for a solution to exist.

Theorem 8 (Local existence). *If f is globally Lipschitz and u is $\mathcal{C}^1$, then for each starting condition $x(0) = x_0$ there is a unique solution to the equations of D, defined on a maximal neighborhood of 0, such that $x(0) = x_0$.*

Observe that, since the set of globally Lipschitz functions is closed under composition, the local existence theorem is valid also when the variables u are the result of a globally Lipschitz function applied to a C^1 function.

Suppose two interacting systems are described by sets of equations D_1 and D_2 of the form given above. Then their combined behavior can be described by the union of the sets of equations D_1 and D_2. It is easy to show that, if the functions occurring in D_1 and D_2 are globally Lipschitz, and D_1 and D_2 do not have any common output and state variables, then the union of these two sets of equations is expressible in the same form with functions that are globally Lipschitz. Thus, in this case no additional machinery is needed to prove that the behavior of the interacting systems is well defined. We define a set D of equations to be Lipschitz if functions f and g are globally Lipschitz.

To extend the above ideas to the hybrid case we define the notion of a Lipschitz HIOA. An HIOA $\mathcal{A}$ is *Lipschitz* if there is a subset M of its state variables (we call these the *mode variables*) such that:

L1 The dynamic type of each variable in M is piecewise constant.

L2 The dynamic type of each variable not in M is a subset of the set of real-valued functions defined on left-closed intervals of the reals that can be expressed in the form $h(c(\cdot))$ where h is a globally Lipschitz function and c is a C^1 function, closed under pasting.

L3 The values of the M variables are constant in each trajectory of $\mathcal{T}$.

L4 For each valuation $\mathbf{m}$ of M there is a Lipschitz system of equations $D_{\mathbf{m}}$ with input variables U, output variables Y, and state variables $X - M$ such that the following holds: If trajectory τ of $\mathcal{T}$ starts from a state $\mathbf{x}$ with $\mathbf{x} \lceil M = \mathbf{m}$, then $\tau \lceil V - M$ is expressible as the concatenation of countably many trajectories $\tau_0, \tau_1, \ldots$, where each τ_i is a solution to $D_{\mathbf{m}}$.

Define a Lipschitz HIOA to be *input bounded* if for each input variable u there exists a positive real value B such that every function in the dynamic type of u has range in $[-B, B]$.

Lemma 9. *Compatible input-bounded Lipschitz HIOAs are strongly compatible.*

Theorem 9. *The composition of two compatible input-bounded Lipschitz HIOAs is a Lipschitz HIOA.*

Theorem 10. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be compatible receptive HIOAs with non-locally-Zeno, input-bounded, Lipschitz strategies. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is a receptive HIOA with a non-locally-Zeno input-bounded Lipschitz strategy.*

Theorem 11. *The composition of two compatible receptive input-bounded Lipschitz HIOAs is a receptive input-bounded Lipschitz HIOA.*

The conclusion that we derive from Theorem 11 is that compatibility implies strong compatibility if we describe the continuous behaviors of HIOAs by means of differential equations of the form of D with functions f and g globally Lipschitz. In general, any choice of conditions on f, g, and u that guarantees local existence of unique solutions, continuity of solutions, and that is preserved by interaction between systems, can be used to define a class of automata for which strong compatibility follows from compatibility.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

2. R. Alur and T.A. Henzinger. Reactive modules. In *Proc. LICS'96*, pages 207–218, 1996.
3. R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *Proc. of CONCUR'97*, pages 74–88. LNCS 1243, 1997.
4. D.J.B. Bosscher, I. Polak, and F.W. Vaandrager. Verification of an audio control protocol. In *Proc. of FTRTFT'94*, pages 170–192. LNCS 863, 1994.
5. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1988.
6. E. Dolginova and N.A. Lynch. Safety verification for automated platoon maneuvers: A case study. In *Proc. of HART'97*, pages 154–170. LNCS 1201, 1997.
7. C.A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, Cambridge, Massachusetts, 1992.
8. A. Kapur, T.A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In *Proc. of FTRTFT'94*, pages 431–454. LNCS 863, 1994.
9. C. Livadas, J. Lygeros, and N.A. Lynch. High-level modelling and analysis of tcas. In *Proc. of RTSS'99*, 1999.
10. C. Livadas and N.A. Lynch. Formal verification of safety-critical hybrid systems. In *Proc. of HSCC'98*, pages 253–272. LNCS 1386, 1998.
11. J. Lygeros and N.A. Lynch. On the formal verification of the tcas conflict resolution algorithms. In *Proc. of 36th IEEE Conference on Decision and Control*, pages 1829–1834, 1997. Extended Abstract.
12. J. Lygeros and N.A. Lynch. Strings of vehicles: Modeling and safety conditions. In *Proc. of HSCC'98*, pages 273–288. LNCS 1066, 1998.
13. N.A. Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In *Hybrid Systems III.* LNCS 1066, 1996.
14. N.A. Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proc. of 3rd AMAST Workshop on Real-Time Systems*, pages 1–22, 1996.
15. N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid i/o automata. In *Hybrid Systems III*, pages 496–510. LNCS 1066, 1996.
16. N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid i/o automata. Technical Report CSI-R9907, Computing Science Institute, University of Nijmegan, 1999.
17. N.A. Lynch and F.W. Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing*, 8(5):499–538, 1996.
18. N.A. Lynch and H.B. Weinberg. Proving correctness of a vehicle maneuver: Deceleration. In *Proc. of 2nd European Workshop on Real-Time and Hybrid Systems*, 1995.
19. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Proc. of REX Workshop on Real-Time: Theory in Practice*, pages 447–484. LNCS 600, 1992.

20. R. Segala, R. Gawlick, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141(2):119–171, March 1998.
21. E.D. Sontag. *Mathematical Control Theory — Deterministic Finite Dimensional Systems*, volume 6 of *Texts in Applied Mathematics*. Springer-Verlag, 1990.
22. H.B. Weinberg and N.A. Lynch. Correctness of vehicle control systems: A case study. In *Proc. of RTSS'96*, pages 62–72, 1996.
23. H.B. Weinberg, N.A. Lynch, and N. Delisle. Verification of automated vehicle protection systems. In *Hybrid Systems III*, pages 101–113. LNCS 1066, 1996.

Proceedings of the 2001 SIGPLAN Conference on Programming Language Design and Implementation, Snowbird, Utah, June 2001.

A Unified Framework for Schedule and Storage Optimization

William Thies[1], Frédéric Vivien[2], Jeffrey Sheldon[1]
and Saman Amarasinghe[1]*

[1] MIT Laboratory for Computer Science
`{thies, jeffshel, saman}@lcs.mit.edu`
[2] Université Louis Pasteur, Strasbourg, France
`vivien@icps.u-strasbg.fr`

Abstract. We present a unified mathematical framework for analyzing the tradeoffs between parallelism and storage allocation within a parallelizing compiler. Using this framework, we show how to find a good storage mapping for a given schedule, a good schedule for a given storage mapping, and a good storage mapping that is valid for all legal schedules. We consider storage mappings that collapse one dimension of a multi-dimensional array, and programs that are in a single assignment form with a one-dimensional schedule. Our technique combines affine scheduling techniques with occupancy vector analysis and incorporates general affine dependences across statements and loop nests. We formulate the constraints imposed by the data dependences and storage mappings as a set of linear inequalities, and apply numerical programming techniques to efficiently solve for the shortest occupancy vector. We consider our method to be a first step towards automating a procedure that finds the optimal tradeoff between parallelism and storage space.

1 Introduction

It remains an important and relevant problem in computer science to automatically find an efficient mapping of a sequential program onto

---

[*] This research was done while Frédéric Vivien was a Visiting Professor in the MIT Laboratory for Computer Science. More information on this project can be found at http://compiler.lcs.mit.edu/aov.

a parallel architecture. Though there are many heuristic algorithms in practical systems and partial or suboptimal solutions in the literature, a theoretical framework that can fully describe the entire problem and find the optimal solution is still lacking. The difficulty stems from the fact that multiple inter-related costs and constraints must be considered simultaneously to obtain an efficient executable.

While exploiting the parallelism of a program is an important step towards achieving efficiency, gains in parallelism are often overwhelmed by other costs relating to data locality, synchronization, and communication. In particular, with the widening gap between clock speed and memory latency, and with modern memory systems becoming increasingly hierarchical, the amount of storage space required by a program can have a drastic effect on its performance. Nonetheless, parallelizing compilers often employ varying degrees of array expansion [9, 5, 1] to eliminate element-level anti and output dependences, thereby adding large amounts of storage that may or may not be justified by the resulting gains in parallelism.

Thus, compilers must be able to analyze the tradeoffs between parallelism and storage requirements in order to arrive at an efficient executable. In this paper, we introduce a unifying mathematical framework that incorporates both schedule constraints (restricting *when* statements can be executed) and storage constraints (restricting *where* their results can be stored). We consider storage mappings that collapse one dimension of a multi-dimensional array, and programs that are in a single assignment form with a one-dimensional schedule. Our technique incorporates general affine dependences across statements and loop nests, making it applicable to many scientific applications

Using this technique, we present solutions to three important scheduling problems. Namely, we show how to determine 1) a good storage mapping for a given schedule, 2) a good schedule for a given storage mapping, and 3) a good storage mapping that is valid for all legal schedules. Our method is precise and practical in that it reduces to a linear program that can be efficiently solved with standard techniques. We believe that these solutions represent the first step towards automating a procedure that finds the optimal compromise between parallelism and storage space.

The rest of this paper is organized as follows. In Section 2 we motivate the problem abstractly, and in Section 3 we define it concretely. Section 4 formulates the method abstractly, and Section 5 illustrates the method with examples. Experiments are described in Section 6, related work in Section 7, and we conclude in Section 8.

2 Abstract problem

To motivate our approach, we consider simplified descriptions of the scheduling problems faced by a parallelizing compiler. We are given a directed acyclic graph $G = (V, E)$. Each vertex $v \in V$ represents a dynamic instance of an instruction; a value will be produced as a result of executing v. Each edge $(v_1, v_2) \in E$ represents a dependence of v_2 on the value produced by v_1. Thus, each edge (v_1, v_2) imposes the *schedule constraint* that v_1 be executed before v_2, and the *storage constraint* that the value produced by v_1 be stored until the execution time of v_2.

Our task is to output (Θ, m), where Θ is a function mapping each operation $v \in V$ to its execution time, and m is the maximum number of values that we need to store at a given time. Parallelism is expressed implicitly by assigning the same execution time to multiple operations. To simplify the problem, we ignore the question of how the values are mapped to storage cells and assume that live values are stored in a fully associative map of size m. How, then, might we go about choosing Θ and m?

2.1 Choosing a store given a schedule

The first problem is to find the optimal storage mapping for a given schedule. That is, we are given Θ and choose m such that 1) (Θ, m) respects the storage constraints, and 2) m is as small as possible.

This problem is orthogonal to the traditional loop parallelization problem. After selecting the instruction schedule by any of the existing techniques, we are interested in identifying the best storage allocation. That is, with schedule-specific storage optimization we can build upon the performance gains of any one of the many scheduling techniques available to the parallelizing compiler.

2.2 Choosing a schedule given a store

The second problem is to find an optimal schedule for a given size of the store, if any valid schedule exists. That is, we are given m and choose Θ such that 1) (Θ, m) respects the schedule and storage constraints, and 2) Θ assigns the earliest possible execution time to each instruction. Note that if m is too small, there might not exist a Θ that respects the constraints.

This is a very relevant problem in practice because of the stepwise, non-linear effect of storage size on execution time. For example, when the storage required cannot be accommodated within the register file

```
A[][] = new int[n][m]
...
for j = 1 to m
  for i = 1 to n
    A[i][j] = f(A[i-2][j-1], A[i][j-1], A[i+1][j-1])
```

Fig. 1. Original code for Example 1.

or the cache, and has to resort to the cache or the external DRAM, respectively, the cost of storage increases dramatically. Further, since there are only a few discrete storage spaces in the memory hierarchy, and their size is known for a given architecture, the compiler can adopt the strategy of trying to restrict the store to successively smaller spaces until no valid schedule exists. Once the storage is at the lowest possible level, the schedule could then be shortened, having a more continuous and linear effect on efficiency than the storage optimization. In the end, we end up with a near-optimal storage allocation and instruction schedule.

2.3 Choosing a store for all schedules

The final problem is to find the optimal storage mapping that is valid for all legal schedules. That is, we are given a (possibly infinite) set $\Psi = \{\Theta_1, \Theta_2, \ldots\}$, where each Θ in Ψ respects the schedule constraints. We choose m such that 1) $\forall \Theta \in \Psi,\ (\Theta, m)$ respects the storage constraints, and 2) m is as small as possible.

A solution to this problem allows us to have the minimum storage requirements without sacrificing any flexibility of our scheduling. For instance, we could first apply our storage mapping, and then arrange the schedule to optimize for data locality, synchronization, or communication, without worrying about violating the storage constraints.

Such flexibility could be critical if, for example, we want to apply loop tiling [10] in conjunction with storage optimization. If we optimize storage too much, tiling could become illegal; however, we sacrifice efficiency if we don't optimize storage at all. Thus, we optimize storage as much as we can without invalidating a schedule that was valid under the original storage mapping.

More generally, if our analysis indicates that certain schedules are undesirable by any measure, we could add edges to the dependence graph and solve again for the smallest m sufficient for all the remaining candidate schedules. In this way, m provides the best storage option that is legal across the entire set of schedules under consideration.

```
A[] = new int[n]
...
for j = 1 to m
  for ALL i = 1 to n
    A[i] = f(A[i-2], A[i], A[i+1])
```

Fig. 2. Transformed code for Example 1. The occupancy vector is (0,1).

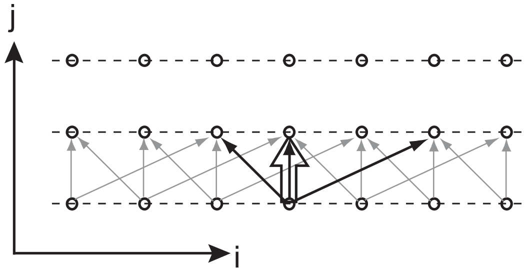


Fig. 3. Iteration space diagram for Example 1. Given the schedule where each row is executed in parallel, our method identifies $(0, 1)$ as the shortest valid occupancy vector.

3 Concrete problem

Unfortunately, the domain of real programs does not lend itself to the simple DAG representation as presented above. Primarily, loop bounds in programs are often specified by symbolic expressions instead of constants, thereby yielding a parameterized and infinite dependence graph. Furthermore, even when the constants are known, the problem sizes are too large for schedule and storage analysis on a DAG, and the executable grows to an infeasible size if a static instruction is generated for every node in the DAG.

Accordingly, we make two sets of simplifying assumptions to make our analysis tractable. The first concerns the nature of the dependence graph G and the scheduling function Θ. Instead of allowing arbitrary edge relationships and execution orderings, we restrict our attention to affine dependences and affine schedules. The second assumption concerns our approach to the optimized storage mapping. Instead of allowing a fully associative map of size m, as above, we employ the *occupancy vector* as a mechanism of storage reuse. In the following sections, we discuss these assumptions in the context of an example.

3.1 Program domain

Primarily, we require an affine description of the dependences of the program. This formulation gives an accurate description of the depen-

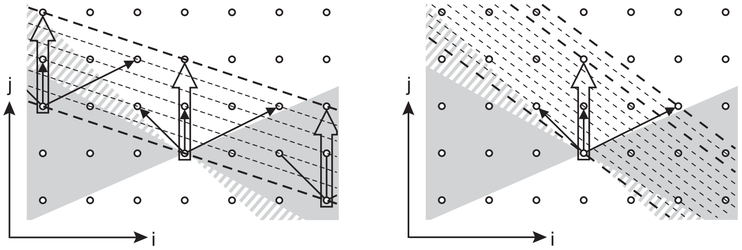


Fig. 4. Iteration space diagram for Example 1. Given an occupancy vector of $(0, 2)$, our method identifies the range of valid schedules. An affine schedule will sweep across the space, executing a line of iterations at once. If this line falls within the gray region (as on the left), then the schedule is valid for the occupancy vector of $(0, 2)$. If this line falls within the striped region (as on the right) then the schedule is valid for some occupancy vector other than $(0, 2)$. The schedule at right is invalid because the operation at the tip of the occupancy vector $(0, 2)$ overwrites a value before the operation at $(2, 1)$ can consume it.

dences of programs with static control flow and affine index expressions [6] and can be estimated conservatively for others. As will become clear below, restricting our attention to affine dependences allows us to model the infinite dependence graph as a finite set of parameters, which is central to the method.

In this paper, we further assume a single-assignment form where the iteration space of each statement exactly corresponds with the data space of the array written by that statement. That is, for array references appearing on the left hand side of a statement, the expression indexing the i'th dimension of the array is the index variable of the i'th enclosing loop (this is formalized below). While techniques such as array expansion [5] can be used to convert programs with affine dependences into this form, our analysis will be most useful in cases where an expanded form was obtained for other reasons (e.g., to detect parallelism) and one now seeks to reduce storage requirements.

We will refer to the example in Figure 1, borrowed from [17]. It clearly falls within our input domain, as the dependences have constant distance, and iteration (i, j) assigns to $A[i][j]$. This example represents a computation where a one-dimensional array $A[i]$ is being updated over a time dimension j, and the intermediate results are being stored. We assume that only the element $A[n][m]$ is used outside the loop; the other values are only temporary.

3.2 Occupancy vectors

To arrive at a simple model of storage reuse, we borrow the notion of an *occupancy vector* from Strout et al. [17]. The strategy is to reduce storage requirements by defining equivalence classes over the locations of an array. Following a storage transformation, all members of a given equivalence class in the original array will be mapped to the same location in the new array. The equivalence relation is:

$$R_{\boldsymbol{v}} = \{(\boldsymbol{l}_1, \boldsymbol{l}_2) \mid \exists k \in \mathbb{Z} \ s.t. \ \boldsymbol{l}_1 = \boldsymbol{l}_2 + k \cdot \boldsymbol{v}\}$$

and we refer to $\boldsymbol{v}$ as the *occupancy vector*. We say that A' is the result of *transforming* A under the occupancy vector $\boldsymbol{v}$ if, for all pairs of locations $(\boldsymbol{l}_1, \boldsymbol{l}_2)$ in A:

$$R_{\boldsymbol{v}}(\boldsymbol{l}_1, \boldsymbol{l}_2) \iff \boldsymbol{l}_1 \ \textit{and} \ \boldsymbol{l}_2 \ \textit{are stored in same location in} \ A'$$

We say that an occupancy vector $\boldsymbol{v}$ is *valid* for an array A with respect to a given schedule Θ if transforming A under $\boldsymbol{v}$ everywhere in the program does not change the semantics when the program is executed according to Θ.

Given an occupancy vector, we implement the storage transformation using the technique of [17] in which the original data space is projected onto the hyperplane perpendicular to the occupancy vector. If an occupancy vector intersects multiple (integral) points of the data space, then modulation must be used to distinguish these points in the transformed array.

Occupancy vector transformations are useful for reducing storage requirements when many of the values stored in the array are temporary. Generally, shorter occupancy vectors lead to smaller storage requirements because more elements of the original array are coalesced into the same storage location. However, the shape of the array also has the potential to influence the transformed storage requirements. Throughout this paper, we assume that the shapes of arrays have second-order effects on storage requirements, and we refer to the "best" occupancy vector as that which is the shortest.

We are now in a position to consider our occupancy vector analysis as applied to Example 1. First, assume that we have chosen to execute each row in parallel so as to have the shortest schedule. What is the best storage mapping for this schedule? Our method can identify $(0, 1)$ as the shortest occupancy vector for this schedule (see Figure 3), yielding the code in Figure 2.

Secondly, consider the case where we become interested in adding some flexibility to our scheduling. If we lengthen the occupancy vector

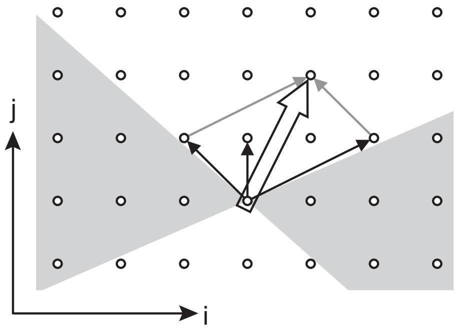


Fig. 5. Iteration space diagram for Example 1. Here the hollow arrow denotes an Affine Occupancy Vector that is valid for all legal affine schedules. The gray region indicates the slopes at which a legal affine schedule can sweep across the iteration domain.

```
A[] = new int[2*n+m]
...
for j = 1 to m
  for i = 1 to n
    A[2*i-j+m] = f(A[2*(i-2)-(j-1)+m],
                   A[2*i-(j-1)+m],
                   A[2*(i+1)-(j-1)+m])
```

Fig. 6. Transformed code for Example 1. The AOV is (1,2).

to $(0, 2)$, what is the range of schedules that we can consider? As illustrated in Figure 4, our method can identify all legal affine schedules for the occupancy vector of $(0, 2)$. We could then use affine scheduling techniques [7] to choose amongst these schedules according to other criteria.

3.3 Affine occupancy vectors

Finally, we might inquire as to the shortest occupancy vector that is valid for all affine schedules in Example 1. An affine schedule is one where each dynamic instance of a statement is executed at a time that is an affine expression of the loop indices, loop bounds, and compile-time constants. To address the problem, then, we need the notion of an Affine Occupancy Vector:

Definition 1 *An occupancy vector* $\boldsymbol{v}$ *for array* A *is an Affine Occupancy Vector (AOV) if it is valid with respect to every affine schedule* Θ *that respects the schedule constraints of the original program.*

Note that, in contrast to the Universal Occupancy Vector of [17], an AOV need not be valid for *all* schedules; rather, it only must be valid for affine ones. Almost all the instruction schedules found in practice are affine, since any FOR loop with constant increment and bounds defines a schedule that is affine in its loop indices. (This is independent of the *array references* found in practice, which are sometimes non-affine.) In this paper, we further relax the definition of an AOV to those occupancy vectors which are valid for all *one-dimensional*[1] affine schedules.

We also observe that, if tiling is legal in the original program, then tiling is legal after transforming each array in the program under one of its AOV's. This follows from the fact that two loops are tilable if and only if they can be permuted without affecting the semantics of the program [10]. Since each permutation of the loops corresponds to a given affine schedule and the AOV is valid with respect to both schedules, the AOV transformation is also valid with respect to a tiled schedule.

Returning to our example, we find using our method that $(1, 2)$ is a valid AOV (see Figure 5). Any affine one-dimensional schedule that respects the dependences in the original code will give the same result when executed with the transformed storage.

4 The Method

4.1 Notation

We adopt the following notation:

- An iteration vector $\boldsymbol{i}$ contains the values of surrounding loop indices at a given point in the execution of the program.
- The structural parameters $\boldsymbol{n}$, of domain $\mathcal{N}$, represent loop bounds and other parameters that are unknown at compile time, but that are fixed for any given execution of the program.
- There are n_s statements $S_1 \ldots S_{n_s}$ in the program. Each statement S has an associated polyhedral domain $\mathcal{D}_S$, such that $\forall \boldsymbol{i} \in \mathcal{D}_S$, there is a dynamic instance $S(\boldsymbol{i})$ of statement S at iteration $\boldsymbol{i}$ during the execution of the program.
- With each statement S is associated a scheduling function θ_S which maps the instance of S on iteration $\boldsymbol{i}$ to a scalar execution time. By assumption, θ_S is an affine function of the iteration vector and the

---

[1] A one-dimensional affine schedule assigns a scalar execution time to each operation as an affine function of the enclosing loop indices and symbolic constants. Multi-dimensional schedules assign vector-valued execution times, which are ordered lexicographically; certain programs require multi-dimensional schedules. See [7, 8, 4] for details.

structural parameters: $\theta_S(\boldsymbol{i}, \boldsymbol{n}) = \boldsymbol{a}_S \cdot \boldsymbol{i} + \boldsymbol{b}_S \cdot \boldsymbol{n} + c_S$. The schedule for the entire program is denoted by $\Theta \in \mathcal{E}$, where $\mathcal{E}$ is the space of all the scheduling parameters $(\boldsymbol{a}_{S_1}, \boldsymbol{b}_{S_1}, \boldsymbol{c}_{S_1}), \ldots, (\boldsymbol{a}_{S_{n_s}}, \boldsymbol{b}_{S_{n_s}}, \boldsymbol{c}_{S_{n_s}})$.

- There are n_p dependences $P_1 \ldots P_{n_p}$ in the program. Each dependence P_j is a 4-tuple $(R_j, T_j, \mathcal{P}_j, h_j)$ where R_j and T_j are statements, h_j is a vector-valued affine function, and $\mathcal{P}_j \subseteq \mathcal{D}_{R_j}$ is a polyhedron such that:

$$\forall \boldsymbol{i} \in \mathcal{P}_j, R_j(\boldsymbol{i}) \textit{ depends on } T_j(\boldsymbol{h}_j(\boldsymbol{i}, \boldsymbol{n})) \tag{1}$$

  The dependences P_j are determined using an array dataflow analysis, e.g., [6] or the Omega test [15].
- There are n_a arrays $A_1 \ldots A_{n_a}$ in the program, and $A(S)$ denotes the array assigned to by statement S. Our assumption that the data space corresponds with the iteration space implies that for all statements S, $S(\boldsymbol{i})$ writes to location $\boldsymbol{i}$ of $A(S)$, and S is the only statement writing to A. However, each array A may still appear on the right hand side of any number of statements, where its indices can be arbitrary affine expressions of $\boldsymbol{i}$ and $\boldsymbol{n}$.
- With each array A we will associate an occupancy vector $\boldsymbol{v}_A$ that specifies the storage reuse within A. The locations $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ in the original data space of A will be stored in the same location following our storage transform if and only if $\boldsymbol{l}_1 = \boldsymbol{l}_2 + k * \boldsymbol{v}_A$, for some integer k. Given our assumption about the data space, we can equivalently state that the values produced by iterations $\boldsymbol{i}_1$ and $\boldsymbol{i}_2$ will be stored in the same location following our storage transform if and only if $\boldsymbol{i}_1 = \boldsymbol{i}_2 + k * \boldsymbol{v}_A$, for some integer k.

4.2 Schedule constraints

According to dependence P_j (Equation (1)), for any value of $\boldsymbol{i}$ in $\mathcal{P}_j$, operation $R_j(\boldsymbol{i})$ depends on the execution of operation $T_j(\boldsymbol{h}_j(\boldsymbol{i}, \boldsymbol{n}))$. Therefore, in order to preserve the semantics of the original program, in any new order of the computations, $T_j(\boldsymbol{h}_j(\boldsymbol{i}, \boldsymbol{n}))$ must be scheduled at a time strictly *earlier* than $R_j(\boldsymbol{i})$, for all $\boldsymbol{i} \in \mathcal{P}_j$. We express this constraint in terms of the scheduling function. We must have, for each dependence P_j, $j \in [1, n_p]$:

$$\forall \boldsymbol{n} \in \mathcal{N}, \forall \boldsymbol{i} \in \mathcal{P}_j, \theta_{R_j}(\boldsymbol{i}, \boldsymbol{n}) - \theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{i}, \boldsymbol{n}), \boldsymbol{n}) - 1 \geq 0 \tag{2}$$

These dependence constraints can be solved using Farkas' lemma as shown by Feautrier [7, 8, 4]. The result can be expressed as a polyhedron $\mathcal{R}$: the set of all the legal schedules Θ in the space of scheduling parameters $\mathcal{E}$. Note that Equation (2) does not always have a solution [7]. In such a case, one needs to use multidimensional schedules [8]. However, in this paper, we assume that Equation (2) has a solution.

Refer to Section 5.1 for an example of the schedule constraints.

4.3 Storage constraints

The occupancy vectors induce some storage constraints. We consider any array A. Because we assume that the data space corresponds with the iteration space, and by definition of the occupancy vectors, the values computed by iterations $\boldsymbol{i}$ and $\boldsymbol{i}+\boldsymbol{v}_A$ are both stored in the same location $\boldsymbol{l}$. For an occupancy vector $\boldsymbol{v}_A$ to be valid for a given data object A, every operation depending on the value stored at location $\boldsymbol{l}$ by iteration $\boldsymbol{i}$ must execute *no later than* iteration $\boldsymbol{i}+\boldsymbol{v}_A$ stores a new value at location $\boldsymbol{l}$. Otherwise, following our storage transformation, a consumer expecting to reference the contents of $\boldsymbol{l}$ produced by iteration $\boldsymbol{i}$ could reference the contents of $\boldsymbol{l}$ written by iteration $\boldsymbol{i}+\boldsymbol{v}_A$ instead, thereby changing the semantics of the program. We assume that, at a given time step, all the reads precede the writes, such that an operation consuming a value can be scheduled for the same execution time as an operation overwriting the value. (This choice is arbitrary and unimportant to the method; under the opposite assumption, we would instead require that the consumer execute at least one step before its value is overwritten.)

Let us consider a dependence $P=(R,T,h,\mathcal{P})$. Then operation $T(\boldsymbol{h}(\boldsymbol{i},\boldsymbol{n}))$ produces a value which will be later on read by $R(\boldsymbol{i})$. This value will be overwritten by $T(\boldsymbol{h}(\boldsymbol{i},\boldsymbol{n})+\boldsymbol{v}_{A(T)})$. The storage constraint imposes that $T(\boldsymbol{h}(\boldsymbol{i},\boldsymbol{n})+\boldsymbol{v}_{A(T)})$ is scheduled no earlier than $R(\boldsymbol{i})$. Therefore, any schedule Θ and any occupancy vector $\boldsymbol{v}_{A(T)}$ respects the dependence P if:

$$\forall \boldsymbol{n}\in\mathcal{N}, \forall \boldsymbol{i}\in\mathcal{Z}, \theta_T(\boldsymbol{h}(\boldsymbol{i},\boldsymbol{n})+\boldsymbol{v}_{A(T)},\boldsymbol{n})-\theta_R(\boldsymbol{i},\boldsymbol{n})\geq 0 \qquad (3)$$

where $\mathcal{Z}$ represents the domain over which the storage constraint applies. That is, the storage constraint applies for all iterations $\boldsymbol{i}$ where $\boldsymbol{i}$ is in the domain of the dependence, and where $\boldsymbol{h}(\boldsymbol{i},\boldsymbol{n})+\boldsymbol{v}_{A(T)}$ is in the domain of statement T. Formally, $\mathcal{Z}=\{\boldsymbol{i}\mid \boldsymbol{i}\in\mathcal{P}\wedge\boldsymbol{h}(\boldsymbol{i},\boldsymbol{n})+\boldsymbol{v}_{A(T)}\in\mathcal{D}_T\}$. This definition of $\mathcal{Z}$ is not problematic, since the intersection of two polyhedra is defined simply by the union of the affine inequalities describing each, which obviously is a polyhedron. Note, however, that $\mathcal{Z}$ is parameterized by both $\boldsymbol{v}_{A(T)}$ and $\boldsymbol{n}$, and not simply by $\boldsymbol{n}$.

Equation (3) expresses the constraint on an occupancy vector for a given dependence and a given schedule. For an occupancy vector to be an AOV, however, it must respect all dependences across all legal schedules. Thus, the following constraint defines a valid AOV $\boldsymbol{v}_A$ for each object A in the program:

$$\forall \Theta \in \mathcal{R}, \forall \boldsymbol{n} \in \mathcal{N}, \forall j \in [1, n_p], \forall \boldsymbol{i} \in \mathcal{Z}_j,$$
$$\theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{i}, \boldsymbol{n}) + \boldsymbol{v}_{A(T_j)}, \boldsymbol{n}) - \theta_{R_j}(\boldsymbol{i}, \boldsymbol{n}) - 1 \geq 0 \tag{4}$$

See Section 5.1 for an illustration of the storage constraints.

4.4 Linearizing the constraints

Equations (3) and (4) represent a possibly infinite set of constraints, because of the parameters. Therefore, we need to rewrite them so as to obtain an equivalent but finite set of affine equations and inequalities, which we can easily solve. Meanwhile, we seek to express the schedule (2) and storage (4) constraints in forms affine in the scheduling parameters Θ. This step is essential for constructing a linear program that minimizes the length of the AOV's.

Section 5.2 contains an illustrative example of the constraint linearization.

Reduction using the vertices of polyhedra Any nonempty polyhedron is fully defined by its vertices, rays and lines [16], which can be computed even in the case of parameterized polyhedra [13]. The following theorem explains how we can use these vertices, rays and lines to reduce the size of our sets of constraints.

Theorem 1. *Let $\mathcal{D}$ be a nonempty polyhedron. $\mathcal{D}$ can be written $\mathcal{D} = P + C$, where P is a polytope (bounded polyhedron) and C is a cone. Then any affine function h defined over $\mathcal{D}$ is nonnegative on $\mathcal{D}$ if and only if 1) h is nonnegative on each of the vertices of P and 2) the linear part of h is nonnegative (resp. null) on the rays (resp. lines) of C.*

Although the domain of structural parameters $\mathcal{N}$ is an input of this analysis and may be unbounded, all the polyhedra produced by the dependence analysis of programs are in fact polytopes, or bounded polyhedra. Therefore, in order to simplify the equations, we now assume that all the polyhedra we manipulate are polytopes, except when stated otherwise. Then, according to Theorem 1, an affine function is nonnegative on a polyhedron if and only if it is nonnegative on the vertices of this polyhedron. We successively use this theorem to eliminate the iteration vector and the structural parameters from Equation (3).

Eliminating the tteration vector Let us consider any fixed values of Θ in $\mathcal{R}$ and $\boldsymbol{n}$ in $\mathcal{N}$. Then, for all $j \in [1, n_p]$, $\boldsymbol{v}_{A(T_j)}$ must satisfy:

$$\forall \boldsymbol{i} \in \mathcal{Z}_j, \theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{i}, \boldsymbol{n}) + \boldsymbol{v}_{A(T_j)}, \boldsymbol{n}) - \theta_{R_j}(\boldsymbol{i}, \boldsymbol{n}) - 1 \geq 0 \tag{5}$$

which is an affine inequality in $\boldsymbol{i}$ (as $\boldsymbol{h}_j$, θ_{T_j}, and θ_{R_j} are affine functions). Thus, according to Theorem 1, it takes its extremal values on the vertices of the polytope $\mathcal{Z}_j$, denoted by $\boldsymbol{z}_{1,j}, \ldots, \boldsymbol{z}_{n_z,j}$. Note that $\mathcal{Z}_j$ is parameterized by $\boldsymbol{n}$ and $\boldsymbol{v}_{A(T_j)}$. Therefore, the number of its vertices might change depending on the domain of values of $\boldsymbol{n}$ and $\boldsymbol{v}_{A(T_j)}$. In this case we decompose the domains of $\boldsymbol{n}$ and $\boldsymbol{v}_{A(T_j)}$ into subdomains over which the number and definition of the vertices do not change [13], we solve our problem on each of these domains, and we take the "best" solution.

Thus, we evaluate (5) at the extreme points of $\mathcal{Z}_j$, yielding the following:

$$\begin{aligned}\forall k \in [1, n_z], \theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{n}), \boldsymbol{n}) + \boldsymbol{v}_{A(T_j)}, \boldsymbol{n})& \\ -\theta_{R_j}(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{n}), \boldsymbol{n}) - 1 \geq 0&\end{aligned} \tag{6}$$

According to Theorem 1, Equations (5) and (6) are equivalent. However, we have replaced the iteration vector $\boldsymbol{i}$ with the vectors $\boldsymbol{z}_{k,j}$, each of which is an affine form in $\boldsymbol{n}$ and $\boldsymbol{v}_{A(T_j)}$.

Eliminating the structural parameters Suppose $\mathcal{N}$ is also a bounded polyhedron. We eliminate the structural parameters the same way we eliminated the iteration vector: by only considering the extremal vertices of their domain $\mathcal{N}$. Thus, for any fixed value of Θ in $\mathcal{R}$, j in $[1, n_p]$, and k in $[1, n_z]$ we must have:

$$\begin{aligned}\forall \boldsymbol{n} \in \mathcal{N}, \theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{n}), \boldsymbol{n}) + \boldsymbol{v}_{A(T_j)}, \boldsymbol{n})& \\ -\theta_{R_j}(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{n}), \boldsymbol{n}) - 1 \geq 0&\end{aligned} \tag{7}$$

Denoting the vertices of $\mathcal{N}$ by $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{n_w})$, the above equation is equivalent to:

$$\begin{aligned}\forall l \in [1, n_w], \theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{w}_l), \boldsymbol{w}_l) + \boldsymbol{v}_{A(T_j)}, \boldsymbol{w}_l)& \\ -\theta_{R_j}(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{w}_l), \boldsymbol{w}_l) - 1 \geq 0&\end{aligned} \tag{8}$$

Case of unbounded domain of parameters. It might also be the case that $\mathcal{N}$ is not a polytope but an unbounded polyhedron, perhaps corresponding to a parameter that is input from the user and can be arbitrarily large. In this case, we use the general form of Theorem 1. Let $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_r}$ be the rays defining the unbounded portion of $\mathcal{N}$ (a line being coded by two opposite rays). We must ensure that the linear part of Equation (8) is nonnegative on these rays. For example, given a single structural parameter $n_1 \in [5, \infty)$, we have the following constraint for the vertex $n_1 = 5$:

$$\begin{aligned}\theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, 5), 5) + \boldsymbol{v}_{A(T_j)}, 5)\\ -\theta_{R_j}(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, 5), 5) - 1 \geq 0\end{aligned}$$

and the following constraint for the positive ray of value 1:

$$\begin{aligned}\theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, 1), 1) + \boldsymbol{v}_{A(T_j)}, 1)\\ -\theta_{R_j}(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, 1), 1)\\ -\theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, 0), 0) + \boldsymbol{v}_{A(T_j)}, 0)\\ +\theta_{R_j}(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, 0), 0) \geq 0\end{aligned} \tag{9}$$

Though this equation may look complicated, in practice it leads to simple formulas since all the constant parts of Equation (7) are going away. We assume in the rest of this paper that $\mathcal{N}$ is a polytope. This changes nothing in our method, but greatly improves the readability of the upcoming systems of constraints!

4.5 Finding a solution

After removing the structural parameters, we are left with the following set of storage constraints:

$$\begin{aligned}\forall j \in [1, n_p], \forall k \in [1, n_z], \forall l \in [1, n_w],\\ \theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{w}_l), \boldsymbol{w}_l) + \boldsymbol{v}_{A(T_j)}, \boldsymbol{w}_l)\\ -\theta_{R_j}(\boldsymbol{z}_{k,j}(\boldsymbol{v}_{A(T_j)}, \boldsymbol{w}_l), \boldsymbol{w}_l) - 1 \geq 0\end{aligned} \tag{10}$$

which is a set of affine inequalities in the coordinates of the schedule Θ, with the occupancy vectors $\boldsymbol{v}_{A(T_j)}$ as unknowns. Note that the vertices $\boldsymbol{z}_{k,j}$ of the iteration domain, the vertices $\boldsymbol{w}_l$ of the structural parameters, and the components $\boldsymbol{h}_j$ of the affine functions, all have fixed and known values.

Similarly, we can linearize the schedule constraints to arrive at the following equations:

$$\begin{aligned}\forall j \in [1, n_p], \forall k \in [1, n_y], \forall l \in [1, n_w],\\ \theta_{R_j}(\boldsymbol{y}_{k,j}(\boldsymbol{w}_l), \boldsymbol{w}_l) - \theta_{T_j}(\boldsymbol{h}_j(\boldsymbol{y}_{k,j}(\boldsymbol{w}_l), \boldsymbol{w}_l), \boldsymbol{w}_l) - 1 \geq 0\end{aligned} \tag{11}$$

Where $y_{1,j}, \ldots, y_{n_y,j}$ denote the vertices of $\mathcal{P}_j$.

Finding an occupancy vector given a schedule At this point we have all we need to determine which occupancy vectors (if any) are valid for a given schedule Θ: we simply substitute into the simplified storage constraints (10) the value of the given schedule. Then we obtain a set of affine inequalities where the only unknowns are the components of the occupancy vector. This system of constraints fully and exactly defines the set of the occupancy vectors valid for the given schedule. We can search this space for solutions with any Linear Programming solver.

To find the shortest occupancy vectors, we can use as our objective function the sum of the lengths[2] of the components of the occupancy vector. This metric minimizes the "Manhattan" length of each occupancy vector instead of minimizing the Euclidean length. However, minimizing the Euclidean length would require a non-linear objective function.

We improve our heuristic slightly by minimizing the difference between the lengths of the occupancy vector components as a second-order term in the objective function. That is, the objective function is

$$obj(\boldsymbol{v}) = k * \sum_{i=1}^{dim(v)} |v_i| + \sum_{i=1}^{dim(v)} \sum_{j=1}^{dim(v)} |v_i - v_j|$$

where k is large enough that the first term dominates, thereby selecting our vector first by the length of its components and then by the distribution of those lengths across its dimensions (a more "even" distribution having a shorter Euclidean distance.) It has been our experience that this linear objective function also finds the occupancy vector of the shortest Euclidean distance.

For an example of this procedure, refer to Section 5.1.

Finding a schedule given an occupancy vector At this point, we also have all we need to determine which schedules (if any) exist for a given set of occupancy vectors. Given an occupancy vector $\boldsymbol{v}_A$ for each array A in the program, we substitute into the linearized storage constraints (10) to obtain a set of inequalities where the only unknowns are the scheduling parameters. These inequalities, in combination with the linearized schedule constraints (11) completely define the space of valid affine schedules valid for the given occupancy vectors. Once again,

---

[2] To minimize $|x|$, set $x = w - z$, $w \geq 0$, $z \geq 0$, and then minimize $w + z$. Either w or z will be zero in the optimum, leaving $w + z = |x|$.

```
A[][] = new int[n][m]
B[][] = new int[n][m]
...
for i = 1 to n
  for j = 1 to m
    A[i][j] = f(B[i-1][j])                 (S1)
    B[i][j] = g(A[i][j-1])                 (S2)
```

Fig. 7. Original code for Example 2.

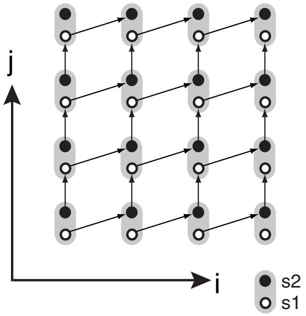


Fig. 8. Dependence diagram for Example 2.

we can search this space for solutions with any Linear Programming solver, selecting the "best" schedule as in [7].

See Section 5.1 for an example.

Finding the AOV's Solving for the AOV's is more involved (follow Section 5.1 for an example.) To find a set of AOV's, we need to satisfy the storage constraints (10) for any value of the schedule Θ within the polyhedron $\mathcal{R}$ defined by the schedule constraints. To do this, we apply the Affine Form of Farkas' Lemma [16, 7, 4].

Theorem 2. *(Affine Form of Farkas' Lemma) Let $\mathcal{D}$ be a nonempty polyhedron defined by p affine inequalities*

$$\boldsymbol{a}_j \cdot \boldsymbol{x} + b_j \geq 0,\ j \in [1, p],$$

in a vector space $\mathcal{E}$. Then an affine form Ψ is nonnegative everywhere in $\mathcal{D}$ if and only if it is an affine combination of the affine forms defining $\mathcal{D}$:

$$\forall \boldsymbol{x} \in \mathcal{E},\ \Psi(\boldsymbol{x}) \equiv \lambda_0 + \sum_j (\lambda_j(\boldsymbol{a}_j \cdot \boldsymbol{x} + b_j)),\ \lambda_0 \ldots \lambda_p \geq 0$$

The nonnegative constants λ_j are referred to as Farkas multipliers.

```
A[] = new int[m+n]
B[] = new int[m+n]
...
for i = 1 to n
  for j = 1 to m
    A[i-j+m] = f(B[(i-1)-j+m])              (S1)
    B[i-j+m] = g(A[i-(j-1)+m])              (S2)
```

Fig. 9. Transformed code for Example 2. Each array has an AOV of (1,1).

To apply the lemma, we note that the storage constraints are affine inequalities in Θ which are nonnegative over the polyhedron $\mathcal{R}$. Thus, we can express each storage constraint as a nonnegative affine combination of the schedule constraints defining $\mathcal{R}$.

To simplify our notation, let *STORAGE* be the set of expressions that are constrained to be nonnegative by the linearized storage constraints (10). That is, *STORAGE* contains the left hand side of each inequality in (10). Naively, $|STORAGE| = n_p \times n_z \times (n_w + n_r)$; however, several of these expressions might be equivalent, thereby reducing the size of *STORAGE* in practice.

Similarly, let *SCHEDULE* be the set of expressions that are constrained to be nonnegative by the linearized schedule constraints (11). The size of *SCHEDULE* is at most $n_p \times n_y \times (n_w + n_r)$.

Then, the application of Farkas' Lemma yields these identities across the vector space $\mathcal{E}$ of scheduling parameters in which Θ lives:

$$STORAGE_i(\boldsymbol{x}) = \lambda_{i,0} + \sum_{j=1}^{|SCHEDULE|} (\lambda_{i,j} \cdot SCHEDULE_j(\boldsymbol{x}))$$

$$\lambda_{i,j} \geq 0, \quad \forall \boldsymbol{x} \in \mathcal{E}, \forall i \in [1, |STORAGE|]$$

These equations are valid over the whole vector space $\mathcal{E}$. Therefore, we can collect the terms for each of the components of x, as well as the constant terms, setting equal the respective coefficients of these terms from opposite sides of a given equation (cf. [7, 4] for full details). We are left with $|STORAGE| \times (3 \times n_s + 1)$ linear equations where the only variables are the λ's and the occupancy vectors $\boldsymbol{v}_A$.

The set of valid AOV's is completely and exactly determined by this set of equations and inequalities. To find the shortest AOV, we proceed as in Section 4.5.

5 Examples

We present four examples to illustrate applications of the method described above.

5.1 Example 1: Simple Stencil

First we derive the solutions presented earlier for the 3-point stencil in Example 1.

Constraints Let θ denote the scheduling function for the statement writing to array A. We assume that θ is an affine form as follows:

$$\theta(i, j, n, m) = a * i + b * j + c * n + d * m + e$$

There are three dependences in the stencil, each from the statement unto itself. The access functions describing the dependences are $\boldsymbol{h}_1(i, j, n, m) = (i-2, j-1)$, $\boldsymbol{h}_2(i, j, n, m) = (i, j-1)$, and $\boldsymbol{h}_3(i, j, n, m) = (i+1, j-1)$. Because these dependences are uniform–that is, they do not depend on the iteration vector–we can simplify our analysis by considering the dependence domains to be across all values of i and j. Thus, the schedule constraints are:

$$\begin{array}{l} \theta(i, j, n, m) - \theta(i-2, j-1, n, m) - 1 \geq 0 \\ \theta(i, j, n, m) - \theta(i, j-1, n, m) - 1 \geq 0 \\ \theta(i, j, n, m) - \theta(i+1, j-1, n, m) - 1 \geq 0 \end{array}$$

However, substituting the definition of θ into these equations, we find that i, j, n, and m are eliminated. This is because the constraints are uniform. Thus, we obtain the following simplified schedule constraints, which are affine in the scheduling parameters:

$$\begin{array}{l} 2 * a + b - 1 \geq 0 \\ b - 1 \geq 0 \\ -a + b - 1 \geq 0 \end{array}$$

Now let $\boldsymbol{v}_A = (v_i, v_j)$ denote the AOV that we are seeking for array A. Then the storage constraints are as follows:

$$\begin{array}{l} \theta(i - 2 + v_i, j - 1 + v_j, n, m) - \theta(i, j, n, m) \geq 0 \\ \theta(i + v_i, j - 1 + v_j, n, m) - \theta(i, j, n, m) \geq 0 \\ \theta(i + 1 + v_i, j - 1 + v_j, n, m) - \theta(i, j, n, m) \geq 0 \end{array}$$

Simplifying the storage constraints as we did the schedule constraints, we obtain the linearized storage constraints:

$$\begin{array}{l} a * v_i + b * v_j - 2 * a - b \geq 0 \\ a * v_i + b * v_j - b \geq 0 \\ a * v_i + b * v_j + a - b \geq 0 \end{array}$$

```
imax = a.length
jmax = b.length
kmax = c.length
D[][][] = new int[imax][jmax][kmax]
...
for i = 1 to imax
  for j = 1 to jmax
    for k = 1 to kmax
      if (i==1) or (j==1) or (k==1) then
        D[i][j][k] = f(i,j,k)                          (S1)
      else
        D[i][j][k] =                                   (S2)
          min(D[i-1][j-1][k-1] + w(a[i],b[j],c[k]),
              D[i][j-1][k-1] + w(GAP,b[j],c[k]),
              D[i-1][j][k-1] + w(a[i],GAP,c[k]),
              D[i-1][j-1][k] + w(a[i],b[j],GAP),
              D[i-1][j][k] + w(a[i],GAP,GAP),
              D[i][j-1][k] + w(GAP,b[j],GAP),
              D[i][j][k-1] + w(GAP,GAP,c[k]))
```

Fig. 10. Original code for Example 3, for multiple sequence alignment. Here f computes the initial gap penalty and w computes the pairwise alignment cost.

Finding an occupancy vector To find the shortest occupancy vector for the schedule that executes the rows in parallel, we substitute $\theta(i, j, n, m) = j$ into the linearized schedule and storage constraints. Minimizing $|v_i + v_j|$ with respect to these constraints gives the occupancy vector of $(0, 1)$ (see Figure 3).

Finding a schedule To find the set of schedules that are valid for the occupancy vector of $(0, 2)$, we substitute $v_i = 0$ and $v_j = 2$ into the linearized schedule and storage constraints. Simplifying the resulting constraints yields:

$$\begin{aligned} b &\geq 1 - 2 * a \\ b &\geq 1 + a \\ b &\geq 2 * a \end{aligned}$$

Inspection of these inequalities reveals that the ratio a/b has a minimum value of $-1/2$ and a maximum value that asymptotically approaches $1/2$, thus corresponding to the set of legal affine schedules depicted in Figure 5 (note that in the frame of the figure, however, the schedule's slope is $-a/b$.)

Finding an AOV To find an AOV for A, we apply Farkas' Lemma to rewrite each of the linearized storage constraints as a non-negative

```
imax = a.length
jmax = b.length
kmax = c.length
D[][] = new int[imax+jmax][imax+kmax]
...
for i = 1 to imax
  for j = 1 to jmax
    for k = 1 to kmax
      if (i==1) or (j==1) or (k==1) then
        D[jmax+i-j][kmax+i-k] = f(i,j,k)              (S1)
      else
        D[jmax+i-j][kmax+i-k] =                       (S2)
          min(D[jmax+(i-1)-(j-1)][kmax+(i-1)-(k-1)] + w(a[i],b[j],c[k]),
              D[jmax+i-(j-1)][kmax+i-(k-1)] + w(GAP,b[j],c[k]),
              D[jmax+(i-1)-j][kmax+(i-1)-(k-1)] + w(a[i],GAP,c[k]),
              D[jmax+(i-1)-(j-1)][kmax+(i-1)-k] + w(a[i],b[j],GAP),
              D[jmax+(i-1)-j][kmax+(i-1)-k] + w(a[i],GAP,GAP),
              D[jmax+i-(j-1)[kmax+i-k] + w(GAP,b[j],GAP),
              D[jmax+i-j][kmax+i-(k-1)] + w(GAP,GAP,c[k]))
```

Fig. 11. Transformed code for Example 3, using the AOV of (1,1,1). The new array has dimension [imax+jmax][imax+kmax], with each reference to [i][j][k] mapped to [jmax+i-j][kmax+i-k].

affine combination of the linearized schedule constraints:

$$\begin{bmatrix} a * v_i + b * v_j - 2 * a - b \\ a * v_i + b * v_j - b \\ a * v_i + b * v_j + a - b \end{bmatrix} =$$

$$\begin{bmatrix} \lambda_{1,1} & \lambda_{1,2} & \lambda_{1,3} & \lambda_{1,4} \\ \lambda_{2,1} & \lambda_{2,2} & \lambda_{2,3} & \lambda_{2,4} \\ \lambda_{3,1} & \lambda_{3,2} & \lambda_{3,3} & \lambda_{3,4} \end{bmatrix} \begin{bmatrix} 1 \\ 2 * a + b - 1 \\ b - 1 \\ -a + b - 1 \end{bmatrix}$$

$$\lambda_{i,j} \geq 0, \ \forall i \in [1,3], \forall j \in [1,4]$$

Minimizing $|v_i + v_j|$ subject to these constraints yields an AOV $(v_i, v_j) = (1, 2)$, which is smaller than the shortest UOV of $(0, 3)$ [17].

To transform the data space of array A according to this AOV $\boldsymbol{v}$, we follow the approach of [17] and project the original data space onto the line perpendicular to $\boldsymbol{v}$. Choosing $\boldsymbol{v}_\perp = (2, -1)$ so that $\boldsymbol{v} \cdot \boldsymbol{v}_\perp = 0$, we transform the original indices of (i, j) into $\boldsymbol{v}_\perp \cdot (i, j) = 2 * i - j$. Finally, to ensure that all data accesses are non-negative, we add m to the new index, such that the final transformation is from $A[i][j]$ to $A[2 * i - j + m]$. Thus, we have reduced storage requirements from $n * m$ to $2 * n + m$. The modified code corresponding to this mapping is shown in Figure 6.

```
A[][] = new int[n][m]
B[] = new int[n]
...
for i = 1 to n
  for j = 1 to n
    A[i][j] = B[i-1]+j              (S1)
  B[i] = A[i][n-i]                  (S2)
```

Fig. 12. Original code for Example 4.

5.2 Example 2: Two-statement stencil

We now consider an example adapted from [12] where there is a uniform dependence between statements in a loop (see Figures 7 and 8). Letting θ_1 and θ_2 denote the scheduling functions for statements 1 and 2, respectively, we have following schedule constraints:

$$\begin{array}{l}\theta_1(i,j,n,m) - \theta_2(i-1,j,n,m) - 1 \geq 0 \\ \theta_2(i,j,n,m) - \theta_1(i,j-1,n,m) - 1 \geq 0\end{array}$$

and the following storage constraints:

$$\begin{array}{l}\theta_2(i-1+v_{B,i}, j+v_{B,j}, n, m) - \theta_1(i,j,n,m) \geq 0 \\ \theta_1(i+v_{A,i}, j-1+v_{A,j}, n, m) - \theta_2(i,j,n,m) \geq 0\end{array}$$

We now demonstrate how to linearize the schedule constraints. We observe that the polyhedral domain of the iteration parameters (i,j) has vertices at $(1,1), (n,1), (1,m), (n,m)$, so we evaluate the schedule constraints at these points to eliminate (i,j):

$$\begin{array}{l}\theta_1(1,1,n,m) - \theta_2(0,1,n,m) - 1 \geq 0 \\ \theta_2(1,1,n,m) - \theta_1(1,0,n,m) - 1 \geq 0 \\ \theta_1(n,1,n,m) - \theta_2(n-1,1,n,m) - 1 \geq 0 \\ \theta_2(n,1,n,m) - \theta_1(n,0,n,m) - 1 \geq 0 \\ \theta_1(1,m,n,m) - \theta_2(1-1,m,n,m) - 1 \geq 0 \\ \theta_2(1,m,n,m) - \theta_1(1,m-1,n,m) - 1 \geq 0 \\ \theta_1(n,m,n,m) - \theta_2(n-1,m,n,m) - 1 \geq 0 \\ \theta_2(n,m,n,m) - \theta_1(n,m-1,n,m) - 1 \geq 0\end{array}$$

Next, we eliminate the structural parameters (n,m). Assuming n and m are positive but arbitrarily large, the domain of these parameters is an unbounded polyhedron:
$(n,m) = (1,1) + j * (0,1) + k * (1,0)$, for positive integers j and k. We must evaluate the above constraints at the vertex $(1,1)$, as well as the linear part of the constraints for the rays $(1,0)$ and $(0,1)$. Doing

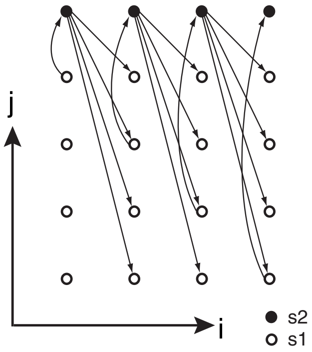


Fig. 13. Dependence diagram for Example 4.

```
A[] = new int[n]
B = new int
...
for i = 1 to n
  for j = 1 to n
    A[i] = B+j                        (S1)
  B = A[i]                            (S2)
```

Fig. 14. Transformed code for Example 4. The AOV's for A and B are (1,0) and 1, respectively.

so yields 24 equations, of which we show the first 3 (which result from substituting into the first of the equations above):

$$\begin{aligned}
&\theta_1(1,1,1,1) - \theta_2(0,1,1,1) - 1 \geq 0\\
&\theta_1(1,1,1,0) - \theta_2(0,1,1,0) - \theta_1(1,1,0,0) + \theta_2(0,1,0,0) \geq 0\\
&\theta_1(1,1,0,1) - \theta_2(0,1,0,1) - \theta_1(1,1,0,0) + \theta_2(0,1,0,0) \geq 0
\end{aligned}$$

Expanding the scheduling functions as $\theta_x(i,j,n,m) = a_x + b_x * i + c_x * j + d_x * n + e_x * m$, the entire set of 24 equations can be simplified to:

$$\begin{aligned}
&d_1 = d_2\\
&e_1 = e_2\\
&a_1 + b_1 + c_1 - a_2 - c_2 + (b_1 - b_2)n - 1 \geq 0\\
&a_1 + 2b_1 + c_1 - a_2 - b_2 - c_2 - 1 \geq 0\\
&a_2 + b_2 + 2c_2 - a_1 - b_1 - c_1 - 1 \geq 0\\
&a_2 + 2c_2 - a_1 - c_1 + (b_2 - b_1)n - 1 \geq 0
\end{aligned}$$

These equations constitute the linearized schedule constraints. In a similar fashion, we could linearize the storage constraints, and then apply Farkas' lemma to find the shortest AOV's of $\boldsymbol{v}_A = \boldsymbol{v}_B = (1,1)$. Due to space limitations, we do not derive the entire solution here. The code that results after transformation by these AOV's is shown in Figure 9.

5.3 Example 3: Multiple sequence alignment

We now consider a version of the Needleman-Wunch sequence alignment algorithm [14] to determine the cost of the optimal global alignment of three strings (see Figure 10). The algorithm utilizes dynamic programming to determine the minimum-cost alignment according to a cost function w that specifies the cost of aligning three characters, some of which might represent gaps in the alignment.

Using θ_1 and θ_2 to represent the scheduling functions for statements 1 and 2, respectively, we have the following schedule constraints (we enumerate only three constraints for each pair of statements since the other dependences follow by transitivity):

$$\begin{array}{l}
\theta_2(i,j,k,x,y,z) - \theta_1(i-1,j,k,x,y,z) - 1 \geq 0 \\
\quad for\ i = 2, j \in [2,y], k \in [2,z] \\
\theta_2(i,j,k,x,y,z) - \theta_1(i,j-1,k,x,y,z) - 1 \geq 0 \\
\quad for\ i \in [2,x], j = 2, k \in [2,z] \\
\theta_2(i,j,k,x,y,z) - \theta_1(i,j,k-1,x,y,z) - 1 \geq 0 \\
\quad for\ i \in [2,x], j \in [2,y], k = 2 \\
\theta_2(i,j,k,x,y,z) - \theta_2(i-1,j,k,x,y,z) - 1 \geq 0 \\
\quad for\ i \in [3,x], j \in [2,y], k \in [2,z] \\
\theta_2(i,j,k,x,y,z) - \theta_2(i,j-1,k,x,y,z) - 1 \geq 0 \\
\quad for\ i \in [2,x], j \in [3,y], k \in [2,z] \\
\theta_2(i,j,k,x,y,z) - \theta_2(i,j,k-1,x,y,z) - 1 \geq 0 \\
\quad for\ i \in [2,x], j \in [2,y], k \in [3,z]
\end{array}$$

Note that each constraint is restricted to the subset of the iteration domain under which it applies. That is, S_2 depends on S_1 only when i, j, or k is equal to 2; otherwise, S_2 depends on itself. This example illustrates the precision of our technique for general dependence domains.

The storage constraints are as follows:

$$\begin{array}{l}
\theta_2(i-1+v_i, j+v_j, k+v_k, x,y,z) - \theta_2(i,j,k,x,y,z) \geq 0 \\
\quad for\ i \in [3,x], j \in [2,y], k \in [2,z] \\
\theta_2(i+v_i, j-1+v_j, k+v_k, x,y,z) - \theta_2(i,j,k,x,y,z) \geq 0 \\
\quad for\ i \in [2,x], j \in [3,y], k \in [2,z] \\
\theta_2(i+v_i, j+v_j, k-1+v_k, x,y,z) - \theta_2(i,j,k,x,y,z) \geq 0 \\
\quad for\ i \in [2,x], j \in [2,y], k \in [3,z]
\end{array}$$

There is no storage constraint corresponding to the dependence of S_2 on S_1 because the domain $\mathcal{Z}$ of the constraint is empty for occupancy vectors with positive components, and occupancy vectors with a non-positive component do not satisfy the above constraints. That is, for the first dependence of S_2 on S_1, the dependence domain is $\mathcal{P} = \{(2, j, k) \mid j \in [2, y] \wedge k \in [2, z]\}$ while the existence domain of S_1 is $\mathcal{D}_{S_1} = \{(i, j, k) \mid i \in [1, x] \wedge j \in [1, y] \wedge k \in [1, z] \wedge (i = 1 \vee j = 1 \vee k = 1)\}$. Then, the domain of the first storage constraint is $\mathcal{Z} = \{(i, j, k) \mid (i, j, k) \in \mathcal{P} \wedge (i-1, j, k) + \boldsymbol{v}_A \in \mathcal{D}_{S_1}\}$. Now, $\mathcal{Z}$ is empty given that $\boldsymbol{v}_A$ has positive components, because if $(i, j, k) \in \mathcal{P}$ then $i = 2$, but if $(i-1, j, k) + \boldsymbol{v}_A \in \mathcal{D}_{S_1}$ then $i - 1 + v_{A,i} = 1$, or equivalently $i + v_{A,i} = 2$. Thus for $\mathcal{Z}$ to be non-empty, we would have $2 + v_{A,i} = 2$, which contradicts the positivity assumption on $v_{A,i}$. The argument is analogous for other dependences of S_2 on S_1.

Applying our method for this example yields an AOV of $(1, 1, 1)$. The transformed code under this occupancy vector is just like the original, except that the array is of dimension [imax+jmax][imax+kmax] and element [i][j][k] is mapped to [jmax+i-j][kmax+i-k].

5.4 Example 4: Non-uniform dependences

Our final example is constructed to demonstrate the application of our method to non-uniform dependences (see Figures 12 and 13). Let θ_1 and θ_2 denote the scheduling functions for statements S_1 and S_2, respectively. Then we have the following schedule constraints:

$$\begin{array}{l} \theta_1(i, j, n) - \theta_2(i-1, n) - 1 \geq 0 \\ \theta_2(i, n) - \theta_1(i, n-i, n) - 1 \geq 0 \end{array}$$

and the following storage constraints:

$$\begin{array}{l} \theta_2(i - 1 + v_B, n) - \theta_1(i, j, n) \geq 0 \\ \theta_1(i + v_{A,i}, n - i + v_{A,j}, n) - \theta_2(i, n) \geq 0 \end{array}$$

Applying our method to these constraints yields the AOV's $\boldsymbol{v}_A = (1, 0)$ and $v_B = 1$. The transformed code is shown in Figure 14.

6 Experiments

We performed preliminary experiments that validate our technique as applied to two of our examples. The tests were carried out on an SGI Origin 2000, which uses MIPS R10000 processors with 4MB L2 caches.

For Example 2, the computation was divided into diagonal strips. Since there are no data dependences between strips, the strips can be assigned to processors without requiring any synchronization [12]. Figure 15 shows the speedup gained on varying numbers of processors using both the original and the transformed array. Both versions show

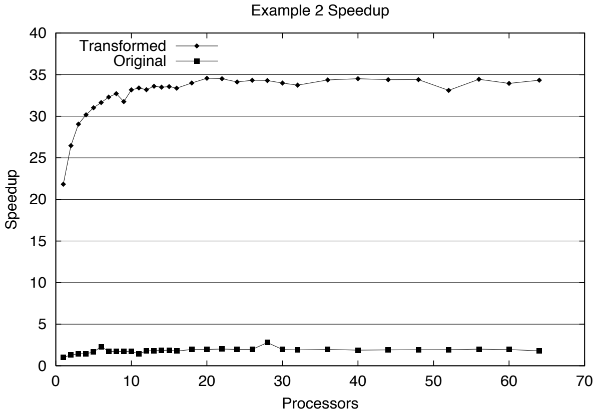


Fig. 15. Speedup vs. number of processors for Example 2.

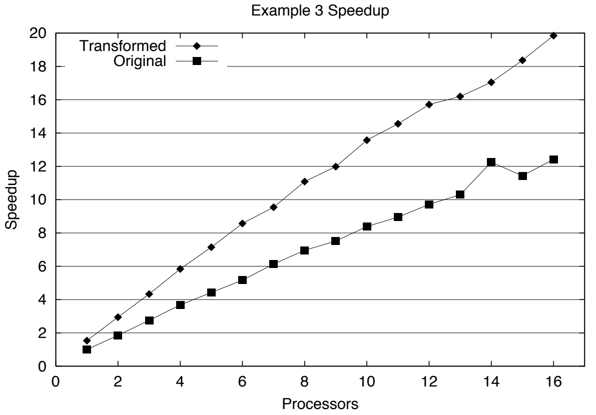


Fig. 16. Speedup vs. number of processors for Example 3.

the same trend and do not significantly improve past 16 processors, but the transformed code has an advantage by a sizable constant factor.

Example 3 was parallelized by blocking the computation, and assigning rows of blocks to each processor. As shown in Figure 16, the transformed code again performs substantially better than the original code. With the reduced working set of data in the transformed code, the speedup is super-linear in the number of processors due to improved caching.

7 Related work

The work most closely related to ours is that of [17], which considers schedule-independent storage mappings using the Universal Occupancy Vector (UOV). While an AOV is valid only for affine schedules, a UOV is valid for any legal execution ordering. Consequently, sometimes there exist AOV's that are shorter than any UOV since the AOV must be valid for a smaller range of schedules. While the analysis of [17] is limited to a stencil of dependences involving only one statement within a perfectly nested loop, our method applies to general affine dependences across statements and loop nests. Moreover, our framework goes beyond AOV's to unify the notion of occupancy vectors with known affine scheduling techniques.

Another related approach to storage management for parallel programs is that of [3, 2, 11]. Given an affine schedule, [11] optimizes storage first by restricting the size of each array dimension and then by combining distinct arrays via renaming. This work is extended in [3, 2] to consider storage mappings for a *set* of schedules, towards the end of capturing the tradeoff between parallelism and storage.

However, these techniques utilize a storage mapping where, in an assignment, each array dimension is indexed by a loop counter and is modulated independently (e.g. `A[i mod n][j mod m]`). This is distinct from the occupancy vector mapping, where the data space of the array is projected onto a hyperplane before modulation (if any) is introduced. The former mapping–when applied to all valid affine schedules–does not enable any storage reuse in Examples 2 and 3, where the AOV did. However, with a single occupancy vector we can only reduce the dimensionality of an array by one, whereas the other mapping can introduce constant bounds in several dimensions. In the future, we hope to extend our method to find multiple occupancy vectors, thereby enabling storage reuse along multiple array dimensions.

Memory reuse in the context of the polyhedral model is also considered in [18]. This approach uses yet another storage mapping, which utilizes array transformations on the data space to achieve the effect of multiple occupancy vectors applied at once. However, the mapping does not have any modulation, so it could not duplicate the effect of an occupancy vector that intersects multiple integral points of the iteration space. Also, the technique assumes that the schedule is given.

8 Conclusion

We have presented a mathematical framework that unifies the techniques of affine scheduling and occupancy vector analysis. Within this framework, we showed how to determine a good storage mapping for a given schedule, a good schedule for a given storage mapping, and a good storage mapping that is valid for all legal schedules. Our technique is general and precise, allowing inter-statement affine dependences and efficiently solving for the shortest occupancy vector using standard numerical programming methods.

We consider this research to be the first step towards automating a procedure that finds the optimal tradeoff between parallelism and storage space. This question is very relevant in the context of array expansion, where the cost of extra array dimensions must be weighed against the scheduling freedom that they provide. Additionally, our framework could be applied to single-assignment functional languages where all storage reuse must be orchestrated by the compiler. In both of these applications, and even for compiling to uniprocessor systems, understanding the interplay between scheduling and storage is crucial for achieving good performance.

However, since finding an exact solution for the “best” occupancy vector is a very complex problem, our method relies on several assumptions to make the problem tractable. We ignore the shape of the data space and assume that the shortest occupancy vector is the best; further, we minimize the Manhattan length of the vector, since minimizing the Euclidean length is nonlinear. Also, we restrict the input domain to programs where 1) the data space matches the iteration space, 2) only one statement writes to each array, 3) the schedule is one-dimensional and affine, and 4) there is an affine description of the dependences. It is with these qualifications that our method finds the “best” solution.

In future work, we aim to relax some of the assumptions about the input domain. Perhaps most relevant is the case of arbitrary affine references on the left hand side, since it would not only widen the input domain, but would allow the reduction of multiple array dimensions via application of successive occupancy vectors. Many of these extensions are difficult because, in their straightforward formulations, the constraints become nonlinear. We consider it to be an open question to formulate these extensions as linear programming problems.

It will also be valuable to consider more general storage mappings. The occupancy vector method as it stands now can only decrease the dimensionality of an array by one, and the irregular shape of the resulting data space could be hard to embed in a rectilinear array in a storage-efficient way. However, other storage mappings [11, 18] we discussed also have their limitations. The perfect storage mapping would allow variations in the number of array dimensions, while still capturing the directional and modular reuse of the occupancy vector and having an efficient implementation; it should also lend itself to efficient storage reuse between distinct arrays.

9 Acknowledgements

We would like to thank Kath Knobe for her helpful comments and suggestions. We appreciate the support of the MARINER project at Boston University for giving us access to its Scientific Computing Facilities. This work was partly supported by NSF Grant CCR0073510, DARPA grant DBT63-96-C-0036, and a graduate fellowship from Siebel Systems.

References

1. D. Barthou, A. Cohen, and J.F. Collard. Maximal static expansion. In *Principles of Programming Languages*, pages 98–106, San Diego, CA, January 1998.
2. A. Cohen. Parallelization via constrained storage mapping optimization. *Lecture Notes in Computer Science*, 1615:83–94, 1999.
3. A. Cohen and V. Lefebvre. Optimization of storage mappings for parallel programs. Technical Report 1998/46, PRiSM, U. of Versailles, 1988.
4. Alain Darte, Yves Robert, and Frédéric Vivien. *Scheduling and Automatic Parallelization*. Birkhäuser Boston, 2000.
5. Paul Feautrier. Array expansion. In *ACM Int. Conf. on Supercomputing*, pages 429–441, 1988.
6. Paul Feautrier. Dataflow analysis of array and scalar references. *Int. J. of Parallel Programming*, 20(1):23–51, 1991.
7. Paul Feautrier. Some efficient solutions to the affine scheduling problem. part I. one-dimensional time. *Int. J. of Parallel Programming*, 21(5):313–347, October 1992.
8. Paul Feautrier. Some efficient solutions to the affine scheduling problem. part II. multidimensional time. *Int. J. of Parallel Programming*, 21(6):389–420, December 1992.
9. Paul Feautrier, Jean-Francois Collard, Michel Barreteau, Denis Barthou, Albert Cohen, and Vincent Lefebvre. The interplay of expansion and scheduling in paf. Technical Report 1998/6, PRiSM, U. of Versailles, 1988.
10. Franois Irigoin and Rémy Triolet. Supernode partitioning. In *Proc. 15th Annual ACM Symp. Principles of Prog. Languages*, pages 319–329, San Diego, CA, January 1988.

11. Vincent Lefebvre and Paul Feautrier. Automatic storage management for parallel programs. *Parallel Computing*, 24(3–4):649–671, May 1998.
12. A. Lim and M. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Prog. Languages*, January 1997.
13. Vincent Loechner and Doran K. Wilde. Parameterized polyhedra and their vertices. *Int. J. of Parallel Programming*, 25(6):525–549, December 1997.
14. S. B. Needleman and C. D. Wunsch. A general method applicable to the search of similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.
15. William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–114, August 1992.
16. Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1986.
17. Michelle Mills Strout, Larry Carter, Jeanne Ferrante, and Beth Simon. Schedule-independent storage mapping for loops. In *Architectural Support for Programming Languages and Operating Systems*, pages 24–33, 1998.
18. D. Wilde and S. Rajopadhye. Memory reuse analysis in the polyhedral model. *Parallel Processing Letters*, 7(2):203–215, June 1997.

International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2001), Atlanta, GA, November 2001, to appear.

Heads and Tails: A Variable-Length Instruction Format Supporting Parallel Fetch and Decode

Heidi Pan and Krste Asanović

MIT Laboratory for Computer Science
{`xoxo, krste`}`@lcs.mit.edu`

Abstract. Existing variable-length instruction formats provide higher code densities than fixed-length formats, but are ill-suited to pipelined or parallel instruction fetch and decode. This paper presents a new variable-length instruction format that supports parallel fetch and decode of multiple instructions per cycle, allowing both high code density and rapid execution for high-performance embedded processors. In contrast to earlier schemes that store compressed variable-length instructions in main memory then expand them into fixed-length in-cache formats, the new format is suitable for direct execution from the instruction cache, thereby increasing effective cache capacity and reducing cache power. The new head-and-tails (HAT) format splits each instruction into a fixed-length head and a variable-length tail, and packs heads and tails in separate sections within a larger fixed-length instruction bundle. The heads can be easily fetched and decoded in parallel as they are a fixed distance apart in the instruction stream, while the variable-length tails provide improved code density. A conventional MIPS RISC instruction set is re-encoded in a variable-length HAT scheme, and achieves an average static code compression ratio of 75% and a dynamic fetch ratio (new-bits-fetched/old-bits-fetched) of 75%.

1 Introduction

Many embedded systems have severe cost, power consumption, and space constraints. Reducing code size is a critical factor in meeting these constraints. Program code is often the largest consumer of memory in control-intensive applications, affecting both system cost and size. Also, instruction fetches are responsible for a significant fraction of system power and memory bandwidth.

Architects of CISC instruction sets had similar motivations for reducing program size and instruction fetch bandwidth, because early systems had small, slow magnetic core memories with no caches. These variable-length CISC instructions tend to give greater code density than fixed-length instructions. However, fixed-length RISC-style instruction sets became popular after inexpensive DRAMs reduced the cost of main memory and large semiconductor instruction caches became feasible to reduce memory bandwidth demands. Fixed-length instructions simplify high performance implementations because the address of the next instruction can be determined before decoding the current instruction (ignoring branches and other changes in control flow). Therefore, they allow fetch and decode to be easily pipelined or performed in parallel for superscalar machines.

Although embedded processors have traditionally had simple single-issue pipelines, newer designs have deeper pipelines or superscalar issue [5, 16, 19] to meet higher performance requirements. Fixed-length ISAs reduce the complexity of pipelined and superscalar fetch and decode, but incur a significant code size penalty.

In this paper, we present a new *heads-and-tails* (HAT) format, which allows compressed variable-length instructions to be held in the cache yet remain easily indexable for parallel fetch and decode. Therefore, we take advantage of the high code density of variable-length instructions while enabling deeply pipelined or superscalar machines.

The paper is structured as follows. In Section 2, we review related work in instruction compression and superscalar variable-length instruction decoding. Section 3 gives a general overview of the HAT instruction format and describes a straightforward hardware implementation. In Section 4, we present an example that packs MIPS RISC [12] instructions into the HAT format using a simple compression scheme. Using MIPS-HAT as a concrete example, we also describe more sophisticated hardware schemes that remove branch penalties. Section 5 presents results for MIPS-HAT using programs taken from the Mediabench benchmark suite. Section 6 concludes the paper.

2 Related work

The ARM Thumb [18] and MIPS16 [13] instruction sets provide alternate 16-bit versions of the base fixed-length RISC ISA (ARM and MIPS respectively) to improve code density. Decompression is a straightforward mapping from the short instruction format to the wider instruction format in the decode stage of the pipeline. Both ISAs allow dynamic switching between full-width and half-width instruction formats at subroutine boundaries. The half-width formats reduce static code size by around 30–40%. However, since they can only encode a limited subset of operations and operand addressing modes, more dynamic instructions are required to execute a given task. The reduced fetch bandwidth can compensate for the increased instruction count when running directly from a 16-bit memory system, but for systems with an instruction cache, performance is reduced by around 20% [18]. Although they are fixed length, the reduced performance makes these short instruction formats unattractive for a superscalar implementation, as a simpler approach to boosting performance would be to revert back to the higher-performing wider format.

An alternative technique that reduces the static code size of a RISC ISA while allowing parallel fetch and decode is to hold instruction cache lines compressed in main memory but then expand them into fixed-length instruction lines when refilling the cache. This idea was introduced with the CCRP scheme [20], and a variety of similar techniques have subsequently been developed and commercialized [10, 15]. Earlier techniques developed for VLIW machines [8] only removed NOP fields within a VLIW instruction, reducing code size to about that of a RISC ISA. The processor remains unchanged with these techniques, as it sees regular easy-to-decode fixed-length instructions in the cache. Caching the uncompressed instructions avoids the additional latency and energy consumption of the decompression unit on cache hits, but decreases the effective capacity of the primary cache and increases the energy used to fetch cached instructions. Cache miss latencies increase for two reasons. First, because the processor uses regular program counter (PC) addresses to index the cache, cache miss addresses must be translated through an additional memory-resident lookup table (the Line Address Table [20]) to locate the corresponding compressed block in main memory, although a miss address translation cache can be added to reduce this penalty (the CLB in [20]). Second, the missing block is often encoded in a form that must be decompressed sequentially, increasing refill latency particularly when the requested word is not the first word in the cache line. For systems with limited memory bandwidth, however, the compressed format can actually reduce total miss latency by reducing the amount of data read from memory [20].

Dictionary-based compression schemes have also been used on instruction streams, where fixed-length code words in the instruction stream point to a dictionary holding commonly occuring instruction sequences [2, 7, 9]. The program code is scanned to determine the commonly occuring strings, which are replaced with codewords pointing into a dictionary. Branch addresses must also be modified to point to locations in the compressed instruction stream. The dictionary is preloaded before program execution starts and forms an additional component of the process state, although it could potentially be managed as a separate cache. The main advantage of these techniques is that decompression is just a fast table lookup. On the other hand, these schemes have several disadvantages. Preloading the table before each program is executed complicates multi-programmed systems, and the table fetch adds latency into the instruction pipeline increasing branch mispredict penalties. Many dictionary schemes interleave variable length code words with uncompressed instructions, severely complicating a highly pipelined or superscalar implementation. Although it might be possible to have parallel fetch and decode from the sequences stored in the dictionary, the common strings tend to be short — often only a single instruction [2, 3, 7]. Dictionary schemes fetch full-size instructions from the dictionary RAM, which is often comparable in size to a primary instruction cache, adding additional instruction fetch energy overhead on top of the fetch of codeword bits from the primary instruction stream.

Of course, the complexity of dynamically compressing instructions can be avoided by adopting a more compact base instruction set. Legacy CISC ISAs, including VAX and x86, provide denser encoding but were intended for microcoded implementations that interpret the instruction format sequentially. Parallel fetch and decode is complicated by the need to examine multiple bytes of an instruction before the start address of the next sequential instruction is known. Nevertheless, the economic importance of legacy CISC instruction sets, such as x86, has resulted in several high-performance superscalar variable-length CISC designs [1, 4, 6, 11]. These all convert complex variable-length instructions into fixed-length RISC-like internal "micro-ops". The Intel P6 microarchitecture can decode three variable-length x86 instructions in parallel, but the second and third instructions must be simple [6]. The P6 takes a brute-force strategy by performing speculative decodes at each byte position, then muxing out the correctly decoded instructions once the lengths of the first and second instructions are determined (further described below). The AMD Athlon design predecodes instructions during cache refill to mark the boundaries between instructions and the locations of opcodes, but still requires several cycles after instruction fetch to scan and align multiple variable-length instructions

[1]. The Pentium-4 design [4] improves on the P6 family by caching decoded fixed-length micro-ops in a trace cache, but similar to the CCRP scheme, cache hits require full-size fixed-length micro-op fetches and cache misses have longer latency due to the decoding process.

These legacy CISC ISAs were not designed with parallel fetch and decode in mind. In this paper, we introduce a new heads-and-tails (HAT) format designed to support parallel fetch and decode of compact variable-length instruction sets directly from cache. The HAT format helps an implementation deliver multiple, variable-sized, randomly-accessible instruction units to the CPU in a single cycle or alternatively enables a deeply-pipelined fetch of such units. This capability can be used in several ways. The HAT format can be used to hold variable-length instructions generated by other compression schemes, or alternatively hold a new ISA developed to take advantage of the format. The example evaluated in this paper uses HAT to hold a quickly-decodable variable-length re-encoding of the MIPS instruction set.

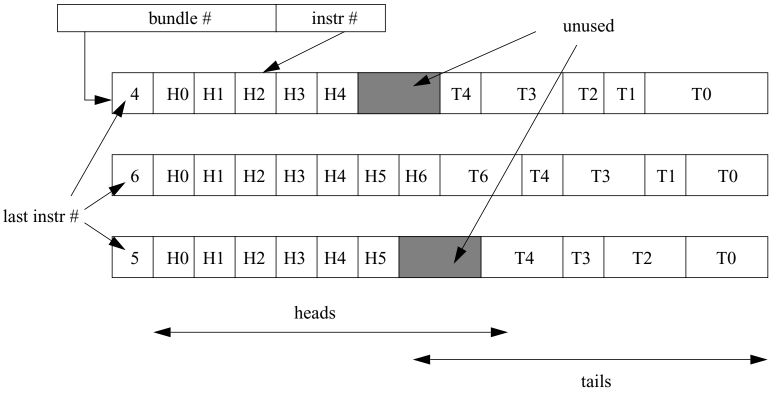


Fig. 1. Overview of heads-and-tails format.

3 Heads and tails format

The HAT format packs multiple variable-length instructions into fixed-length bundles as shown in Figure 1. The HAT format is used both in main memory and cache, although additional information might be added to the cached version to improve performance as described below. A cache line could contain one or more bundles. Bundles contain varying numbers of instructions, so each bundle begins with a small fixed-length field holding the number of the last instruction in the bundle, i.e. a bundle holding N instructions has $N-1$ in this field. The remainder of the bundle is used to hold instructions.

Each instruction is split into a fixed-length head portion and a variable-length tail portion. The fixed-length heads are packed together in program order at the start of the bundle, while the variable-length tails are packed together in reverse program order at the end (i.e., the first tail is at the end of the bundle). Not all heads necessarily have a tail, though this can simplify some hardware implementations. The granularity of the tails is independent of the size of the heads, i.e., the heads could be 11-bits long while the tails are multiples of 5 bits, though there can be hardware advantages to making the head length a multiple of the tail granularity as discussed below. When packing compressed instructions into bundles, there can be internal fragmentation if the next instruction doesn't fit into the remaining space in a bundle, in which case the space is left empty and a new bundle is started.

The program counter (PC) in a HAT scheme is split into a bundle number held in the high bits and an instruction offset held in the low bits. During sequential execution, the PC is incremented as usual, but after fetching the last instruction in a bundle (as given by the instruction count stored in the bundle), it will skip to the next bundle by incrementing the bundle number and reseting the instruction offset to zero. All branches into a bundle have their target instruction offset field checked against the instruction count, and a PC error is generated if the offset is larger than the instruction count.

A PC value points directly to the head portion of an instruction and, because they are fixed-length, multiple sequential instruction heads can be fetched and decoded in parallel. The tails are still variable-length, however, and so the heads must contain enough information to locate the correct tail. One approach would be for each head to have a pointer to its tail, but this would usually require a large number of bits. Fewer bits are needed if the head just encodes the presence and length of a tail. This length information can often be folded into the opcode information to further reduce code size, as described below in the MIPS-HAT scheme. Similar to a conventional variable-length scheme, the tail size information in the head of one instruction must be decoded to ascertain the location of the start of the tail of the next instruction. But in the HAT format the length information for each instruction is held at a fixed spacing in the head instruction stream, independent of the length of the whole instruction. This makes the critical path to determine tail alignment for multiple parallel instructions much shorter than in a conventional variable-length scheme, where the *location* of the length information in the next instruction depends on the length of the current instruction.

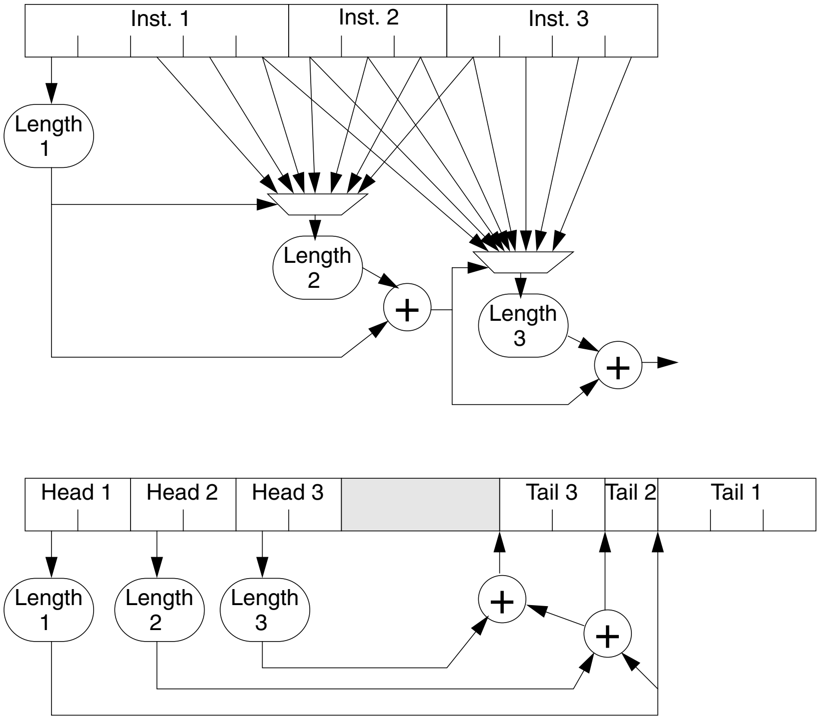


Fig. 2. Comparison of variable-length decoding in a conventional variable-length scheme and a HAT scheme.

This difference between a regular variable-length scheme and a HAT scheme is illustrated in Figure 2. The Figure shows a three-issue superscalar length decoder for a conventional variable-length ISA and a HAT ISA scheme. In both cases, instructions vary from 2–8 bytes and length information is encoded in the first byte. In the conventional scheme, the length decoder for the second instruction cannot produce a value until the first length decoder drives the mux to steer the correct byte into the second length decoder. Similarly, the third length decoder has to wait for the first two to complete before its input settles. The output of the third decoder is needed to determine the correct amount to shift the instruction input buffer for the next cycle. This scheme scales poorly, as $O(W^2)$ area and delay for issue width W, because the number of inputs to the length byte muxes grows linearly with the number of parallel instructions. The Intel P6 family reduces this critical path by replicating simple decoders at every byte position, then muxing out the correct instructions. This requires considerable die area and additional power, and still scales as $O(W^2)$ albeit with a smaller constant for delay. In contrast, the HAT scheme operates all the length decoders in parallel, and then sums their outputs to determine tail alignments. This addition can be organized as a parallel prefix sum using a carry-save adder tree, and so delay scales logarithmically with issue width $O(\log W)$, and hardware costs grow as $O(W \log W)$.

The tails in a HAT scheme are delayed relative to the heads, but the head and tail fetches can be pipelined independently. The performance impact of the additional latency for the tails can be partly hidden if more latency-critical instruction information is located in the head portions.

3.1 Handling branches in HAT

While fetching sequentially within a bundle, the HAT instruction decoder is consuming head bits from one end of the bundle and tail bits from the other end. To avoid having to fetch and decode a new bundle before locating its first instruction's tail bits, we place tails in reverse order starting at the end of the bundle. When execution moves sequentially on to a new bundle, the initial head and tail data can be simply found at either end of the new bundle.

Branches create the biggest potential problems for the HAT scheme. Whereas a branch target address points at the entire target instruction in a conventional scheme, it only locates the head within a bundle in a HAT scheme. One simplistic approach to locate the tail of a branch target is to scan all earlier heads from the beginning of the target bundle, summing their tail lengths to get a pointer to the start of the branch target's tail. Although correct, this scheme would add a substantial delay and energy penalty to taken branch instructions. Next, we describe three different approaches to finding branch target tails in a HAT scheme: tail-start bit vectors, tail pointers, and an enhanced branch target buffer.

Tail-start bit vector We can reduce branch penalties for locating the target tail by storing auxiliary data structures in the cache alongside each bundle. These data structures do not impact static code size as they are only present in the cache, but they increase cache area and the number of dynamic bits fetched from the cache, potentially increasing cache hit energy. The simplest scheme would be to hold a separate tail pointer for each possible instruction in a bundle, but this incurs a large overhead of $H \log(T)$ bits per bundle, where H is the maximum number of heads and T is the number of possible tail positions. A more compact approach is to store a single bit per tail position (T bits total per bundle), each bit indicating the possible start of a tail. A branch into a bundle would then read the bit vector to find the start of the Nth tail. This bit vector approach handles both fixed and indirect jumps, but adds some additional latency to taken branches to process the bit vector. This scheme also requires that every instruction has a tail, otherwise a second bit vector would be required to determine which instructions had tails.

Tail pointers A different approach to finding branch target tails is to change branch and jump instruction encodings to include an additional tail pointer field pointing to the tail portion of the branch target. This is filled in by the linker at link time. The tail pointer removes all latency penalties for fixed-target branch instructions, but increases code size slightly. This approach, however, cannot be used for indirect jumps where the target address is not known until run time.

There are two schemes that can be used to handle indirect jumps with tail pointers. The first scheme is to expand all PC values to contain a tail pointer in addition to the bundle and instruction offset numbers. Jump-to-subroutine instructions would then write these expanded PCs into the link register as return PC values, and jump indirect instructions would expect tail pointers in the PC values held in registers as jump targets. A minor disadvantage of this scheme is that it reduces the virtual address space available for user code by the number of bits taken for the target tail pointer ($\log(T)$ bits). Another disadvantage is that it becomes possible to branch to the middle of a tail if the user manipulates the target tail pointer directly.

The second scheme treats each type of indirect jump separately. There are three main uses of indirect jumps: indirect function calls (e.g., virtual functions in C++), switch statement tables, and subroutine returns. We can eliminate penalties on function calls and switch tables by noting that a branch to the start of a bundle can always find the tail bits of the first instruction at the end of the bundle. Therefore by simply placing function entry points and case statement entry points at the start of a bundle (which might be desirable for cache performance in any case), we eliminate branch penalties for these indirect jumps.

Subroutine returns cannot be handled as easily because the subroutine call could be anywhere within a bundle. One simple approach is to only allow instructions without tails between the subroutine call and the end of the current bundle, as a tail-less instruction does not need the tail pointer to be restored correctly after the subroutine returns. This is likely to reduce performance and waste code space, as NOPS will have to be inserted if an instruction with tail is required. Another approach is to store the return PC tail pointer on the subroutine return address stack, if the microrachitecture has one to predict subroutine returns. If the return address stack prediction fails, execution falls back to the naive algorithm that scans heads from the beginning of the target bundle.

BTB for HAT branches The third general approach to handling branches in a HAT scheme stores target tail pointer information in the branch target buffer (BTB). This can handle both fixed and indirect jumps. Again, if the prediction fails, the target bundle can be scanned from the beginning to locate a tail in the middle of the bundle. This approach does not increase static code size, but increases BTB size and branch mispredict penalty.

3.2 HAT advantages

To summarize, the HAT scheme has a number of advantages over conventional variable-length schemes.

- Fetch and decode of multiple variable-length instructions can be pipelined or parallelized.
- Unlike conventional variable-length formats, it is impossible to jump into the middle of an instruction (except if PCs are expanded to include a tail pointer field as described above).
- The PC granularity is always in units of a single instruction, and is independent of the granularity at which the instruction length can be varied. This allows branch offsets to be encoded with fewer bits than a conventional variable-length ISA, where PC granularity and instruction length granularity are identical (e.g., in bytes). This helps counteract the code size increase if tail pointers are added to branch target specifiers.
- The variable alignment muxes needed are smaller than in a conventional variable-length scheme, because they only have to align bits from the tail and not from the entire instruction length. The fixed-length heads are handled using a much simpler and faster mux.

- The HAT format guarantees that no variable-length instruction straddles a cache line or page boundary, simplifying instruction fetch and handling of page faults.

4 MIPS-HAT

In this section, we demonstrate the HAT format using a compressed variable-length re-encoding of the MIPS RISC ISA [12] as an example.

4.1 MIPS-HAT compression techniques

The MIPS compression scheme we use is based partly on a previous scheme by Panich [17]. To keep instruction decoding simple, we choose to never split MIPS register specifier fields, and so use a 5-bit granularity for our tail encoding. Our minimum size instruction is 15 bits and the maximum size is 40 bits. As discussed later in the hardware section, tail lookup can be simplified if every instruction has a tail and so we chose heads that are 10 bits long but always with a tail, giving a minimum instruction size of 15 bits. The following techniques were used to compress the MIPS instructions.

1. Use the minimum number of 5-bit fields to encode immediates.
2. Eliminate unused register and operand fields.
3. Certain instructions often use a specific value for a register or immediate, for example, the BEQ instruction often (≈90%) has zero as one operand. We provide new opcodes for these cases.
4. We provide two-address versions of instructions that frequently have a source register the same as the destination register.
5. We re-encode some common instruction sequences as a single instruction. We re-encode only the simplest but most common two types of instruction sequence: branch instructions with a NOP in the delay slot and multiple sequential loads. New opcodes for branches and jumps indicate that they are followed by a NOP. The multiple load instructions are used by subroutines to restore saved registers from consecutive offsets from the stack pointer and can be combined into a single instruction by specifying the initial register, initial offset, and the number of load instructions in the sequence. We considered a multiple store instruction but this did not provide sufficient savings to be justified (we believe this asymmetry was because the compiler often interleaves code from the start of a function with the register save code in the prologue whereas the register restore in the function epilogue is not polluted in the same way).

Each instruction can be one of six sizes, ranging from 15–40 bits. One way to specify the size would be to attach three overhead bits per instruction. However, each instruction type, e.g., ADDI (add-immediate), typically only uses a few sizes, so we fold instruction sizes into new opcodes, e.g. ADDI10b for a 10-bit add-immediate.

This substantially increases the number of possible opcodes, but only a small subset of these new opcodes is frequently used. We select the most popular opcodes, together with several different "escape" opcodes, and encode these in a 5-bit primary opcode field in the head. The escape opcodes indicate that a secondary opcode is placed in the tail, but also includes critical information required for decode, such as the size of the instruction and its general category (e.g., arithmetic versus branch). Table 1 and Table 2 show the most popular primary opcodes and escape opcodes together with the frequency that they occur across the Mediabench benchmarks. The "Break" escape opcode is used for all instructions that will cause opcode traps, including SYSCALL and BREAK.

Table 1. The 32 MIPS-HAT primary opcodes.

Instruction	Size	Freq	Instruction	Size	Freq
Specific Primary Opcodes					
addu(rt=0)	15	8.7%	lw(imm=0)	15	2.2%
sw	25	5.2%	sw	20	1.9%
lw	25	4.7%	addu	20	1.8%
addiu	25	4.5%	lw	20	1.7%
noop	15	4.3%	addiu(-1)	15	1.6%
lui	30	3.6%	jr	15	1.5%
addiu(+1)	15	3.2%	bne(rt=0)	15	1.4%
jal	25	3.2%	beq(rt=0)	15	1.3%
addu(rs=rd)	15	2.6%	addiu(rs=rd)	15	1.2%
sw(rw=r2)	20	2.6%	addiu(rs=rd)	20	1.2%
addiu	20	2.4%	addiu	30	1.1%
j	25	2.2%			
Escape Opcodes					
I-Load/Store	30	10.0%	I-Arithmetic	40	1.5%
R	25	7.2%	I-Load/store	40	0.4%
I-Branch	30	6.7%	I-Branch	40	0.0%
I-Arithmetic	30	5.4%	J	40	0.0%
Break	35	3.3%			

Table 2. MIPS-HAT primary opcodes by category.

Instruction	Size	Freq	Instruction	Size	Freq
R					
addu(rt=0)	15	8.7%	addu(rs=rd)	15	2.6%
ESC	25	7.2%	addu	20	1.8%
noop	15	4.3%	jr	15	1.5%
I-Arithmetic					
ESC	30	5.4%	addiu(-1)	15	1.6%
addiu	25	4.5%	ESC	40	1.5%
lui	30	3.6%	addiu(rs=rd)	15	1.2%
addiu(+1)	15	3.2%	addiu(rs=rd)	20	1.2%
addiu	20	2.4%	addiu	30	1.1%
I-Branch					
ESC	30	6.7%	beq(rt=0)	15	1.3%
bne(rt=0)	15	1.4%	ESC	40	0.0%
I-Load/Store					
ESC	30	10.0%	lw(imm=0)	15	2.2%
sw	25	5.2%	sw	20	1.9%
lw	25	4.7%	lw	20	1.7%
sw(rw=r2)	20	2.6%	ESC	40	0.4%
J					
jal	25	3.2%	ESC	40	0.0%
j	25	2.2%			
Break					
ESC	35	3.3%			

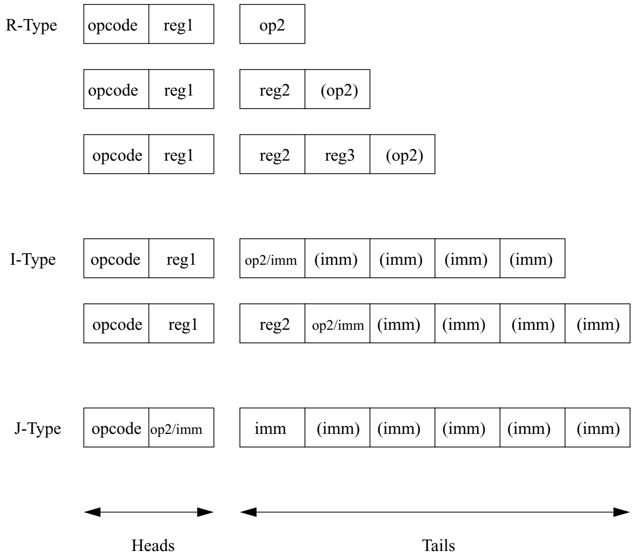


Fig. 3. Compressed MIPS instruction formats.

Figure 3 shows the formats of the three types of MIPS-HAT instructions — register (R), immediate (I), and jump (J). All fields are five bits wide. The fields in parenthesis are optional, depending on the instruction length.

4.2 Bundle format

We evaluated use of both 128-bit and 256-bit bundles for MIPS-HAT. The 128b bundle is split into a three-bit instruction count field and 25×5b units, holding up to 8×10b heads and up to 16×5b tail units. The 256b bundle has a four-bit instruction count field, two empty bits, and 50×5b units which can hold up to 16×10b heads and up to 32×5b tail units.

Note that we restrict the size of the head and tail regions to reduce the number of bits needed for the instruction count field and for the tail-start bit vector if present. Neither the head nor tail region completely spans the bundle, although the boundary between the regions is flexible. In practice, it is rare for bundle packing to be affected by this restriction.

4.3 HAT cache implementation

MIPS-HAT is designed to be directly executed from cache, and instructions remain in the same format after being fetched from memory to cache, avoiding additional cache miss latencies. The new format is only slightly more complex than regular MIPS to decode, and the decompression is just folded into the decoder.

A conventional variable-length ISA would fetch words of data sequentially from the cache into fetch buffers that can rotate the data to the correct alignment for the instruction decoder. MIPS-HAT would use the same scheme for the tails, but in addition would be fetching a second stream for the fixed-length heads which would not require an alignment circuit. The cache RAM does not require a second read port to provide the head data stream, as the heads are always from the same bundle as the tails and hence would be on the same cache line. The cache RAM sense-amps just need a separate set of bus drivers onto the head data bus.

Table 3. Static Compression Ratios

Input	128b	256b	128b BrTail	256b BrTail
adpcm-dec	78.5%	75.5%	82.6%	82.1%
adpcm-enc	78.6%	75.6%	82.6%	82.0%
epic-dec	77.1%	74.0%	80.4%	79.5%
epic-enc	78.7%	75.5%	81.6%	80.8%
g721-dec	78.0%	75.0%	82.3%	81.6%
g721-enc	78.0%	75.0%	82.3%	81.6%
gsm-dec	79.8%	76.8%	85.2%	84.4%
gsm-enc	79.8%	76.8%	85.2%	84.4%
jpeg-dec	74.3%	71.5%	78.4%	77.5%
jpeg-enc	74.2%	71.5%	78.7%	77.9%
mpeg2-dec	80.6%	77.9%	85.5%	85.2%
mpeg2-enc	81.6%	79.1%	86.4%	86.4%
pegwit-dec	80.0%	76.6%	84.4%	84.4%
pegwit-enc	80.0%	76.6%	84.4%	84.4%
Average	78.5%	75.5%	82.8%	82.3%

Because head information is needed to extract the tails, the tail instruction bits always lag the heads. To reduce the impact of this additional latency on the execution pipeline, MIPS-HAT places the instruction category in the head so that the instruction can be steered

Table 4. Instruction Size Distribution

	15b	20b	25b	30b	35b	40b
Average (w/o BrTail)	22.1%	13.0%	47.5%	3.8%	3.3%	10.4%
Cumulative	22.1%	35.1%	82.6%	86.4%	89.6%	100.0%
Average (w/ BrTail)	19.8%	16.1%	35.1%	17.3%	3.3%	8.4%
Cumulative	19.8%	35.9%	70.9%	88.2%	91.6%	100.0%

to an appropriate functional unit before the tail arrives, allowing the tail to be sent directly to the appropriate unit for further decoding.

5 Experimental results

To test the effectiveness of the MIPS-HAT scheme, we selected benchmarks from the Mediabench [14] benchmark suite, reencoded the MIPS binaries generated by a `gcc` cross-compiler (`egcs-1.0.3a -O2`), and took static and dynamic measurements. For the dynamic measurements, the Mediabench programs were run to completion on the provided input sets.

5.1 Static compression ratios

Table 3 gives the static compression ratios (compressed-size/original-size) for 128b and 256b versions of MIPS-HAT. The bundle ratios for the two sizes includes the overhead bits to count the instructions in each bundle and any wasted space due to fragmentation.

The average bundle compression ratio is 78.5% for the 128b bundle and 75.5% for the 256b bundle. The smaller bundle incurs relatively more overhead and has more internal fragmentation. If we adopt the scheme that adds target tail links to speed taken branches, the static code size increases, to a compression ratio of 82.8% for 128b bundles and 82.3% for the 256b bundles.

Table 4 shows the distribution of static instruction sizes averaged over the benchmark set, with and without the tail pointer scheme. Without target tails, over 80% of instructions are 25 bits or less.

5.2 Dynamic measures

We measured the reduction in dynamic bits fetched from the instruction cache using the MIPS-HAT scheme. We report this number as a dynamic fetch ratio (new-bits-fetched/original-bits-fetched). We evaluated several different schemes to avoid taken branch penalties

Tables 5 and 6 show the dynamic fetch ratios for 128b and 256b bundles, respectively, for a variety of implementations. The baseline column shows the ratios including the cost of fetching the instruction count on every access to a new bundle. The 256b scheme has a slightly lower fetch ratio (75.0% versus 75.5%) as relatively fewer overhead bits are fetched.

The BrBV column shows the large increase in dynamic fetch ratio when a tail-start bit vector (Section 3.1) is used to reduce branch taken penalties. The increase is less for the 128b bundles which have a 16b vector per line, such that these now have lower fetch ratios than 256b bundles, which must fetch a 32b vector on every taken branch.

The BrTail columns shows the fetch ratio for the tail pointer scheme, where branch instruction encodings include a tail pointer. These fetch ratios are much lower than for the BrBV approach, but this technique has a higher static code size.

Table 5. Dynamic Compression Ratios - 128b

Input	Line Ratio	BrBV	BrTail
adpcm-dec	72.0%	79.8%	75.0%
adpcm-enc	74.5%	84.0%	76.9%
epic-dec	75.2%	83.4%	77.7%
epic-enc	85.5%	89.3%	87.8%
g721-dec	75.3%	82.2%	78.4%
g721-enc	75.3%	82.2%	78.5%
gsm-dec	75.5%	79.6%	76.0%
gsm-enc	72.0%	74.1%	74.5%
jpeg-dec	68.2%	71.0%	69.1%
jpeg-enc	72.9%	79.9%	73.9%
mpeg2-dec	80.1%	85.3%	82.0%
mpeg2-enc	74.0%	79.1%	75.7%
pegwit-dec	79.1%	83.2%	80.8%
pegwit-enc	78.0%	82.3%	79.8%
average	75.5%	81.1%	77.6%

A BTB aproach to locating target tails would add nothing to the static code size, and would have a dynamic fetch ratio similar to the BrTail scheme, except now some of these bits would be fetched from

Table 6. Dynamic Compression Ratios - 256b

Input	Line Ratio	BrBV	BrTail
adpcm-dec	71.2%	86.9%	74.5%
adpcm-enc	73.5%	92.5%	76.4%
epic-dec	74.5%	91.0%	77.3%
epic-enc	85.1%	92.8%	87.1%
g721-dec	75.0%	88.9%	78.5%
g721-enc	73.8%	87.7%	78.4%
gsm-dec	74.8%	83.1%	77.5%
gsm-enc	71.3%	75.5%	72.2%
jpeg-dec	67.5%	80.7%	68.8%
jpeg-enc	72.4%	86.3%	75.3%
mpeg2-dec	79.7%	90.1%	79.7%
mpeg2-enc	76.1%	83.8%	75.1%
pegwit-dec	78.2%	86.5%	79.9%
pegwit-enc	77.1%	85.8%	78.8%
average	75.0%	86.5%	77.1%

the BTB structure. The BTB scheme will also incur additional latency penalties on BTB mispredicts.

5.3 Results discussion

The numbers show that there are tradeoffs between static code size, dynamic fetch ratio, and taken branch performance, depending on the bundle size and the branch penalty avoidance scheme. The larger bundle generally gives the best reduction in code size and bits fetched. Our dynamic results did not measure the expected increase in performance due to the effective increase in cache capacity, which should lower miss rates.

Other work has presented compression numbers for MIPS code. CCRP [20] achieves a compression ratio of 73% but has to uncompress instructions into cache to allow parallel fetch and decode. MIPS16 [13] obtains a compression ratio of around 60%, but at the cost of limiting operations and operand addressing modes which reduces performance. SAMC and SADC [15] use more complex algorithms to achieve a compression ratio nearly 50% on MIPS code but either with a long decoding delay or an added dictionary lookup step.

6 Conclusions

We have introduced a new head-and-tails (HAT) variable-length instruction format that separates instructions into fixed-length heads that can be easily indexed and variable-length tails that provide code compression. The format can provide high code density in memory and in cache, while allowing parallel fetch and decode for direct superscalar execution from cache. The HAT scheme makes it difficult to quickly locate an entire instruction at a branch target. A number of techniques are possible to reduce taken branch penalties, and these were shown to have differing effects on static code size, dynamic bits fetched, and branch penalties.

We developed a simple MIPS instruction compression scheme by re-encoding the MIPS ISA into a variable-length format, and mapping the resulting variable-length instructions into the HAT format. Our experiments showed that the MIPS-HAT format can provide a compression ratio of 75.5% and a dynamic fetch ratio reduction of 75.0% while supporting deeply pipelined or superscalar execution.

The HAT format can be applied to many other types of instruction encoding. For example, each instruction slot in a VLIW instruction could be encoded in a similar way as MIPS-HAT to give similar savings (over and above simple NOP compression). In future work, we are also investigating more aggressive instruction compression techniques tuned for the HAT format, as well as developing new instruction sets that take advantage of the HAT format to increase performance without sacrificing code density.

7 Acknowledgements

This work was funded by DARPA PAC/C award F30602-00-2-0562 and by NSF CAREER award CCR-0093354.

References

1. *AMD Athlon Processor x86 Code Optimization*, chapter Appendix A: AMD Athlon Processor Microarchitecture. AMD Inc., 220071-0 edition, September 2000.
2. C. Lefurgy et al. Improving code density using compression techniques. In *MICRO-30*, pages 194–203, Research Triangle Park, North Carolina, December 1997.
3. G. Araujo et al. Code compression based on operand factorization. In *MICRO-31*, pages 194–201, December 1998.

4. G. Hinton et al. The microarchitecture of the Pentium 4 processor. *Intel Technology Journal*, Q1 2001.
5. J. Choquette et al. High-performance RISC microprocessors. *IEEE Micro*, 19(4):48–55, July/August 1999.
6. J. Circello et al. The superscalar architecture of the MC68060. *IEEE Micro*, 15(2):10–21, April 1995.
7. L. Benini et al. Selective instruction compression for memory energy reduction in embedded systems. In *ISLPED*, pages 206–211, August 1999.
8. R. P. Colwell et al. A VLIW architecture for a trace scheduling compiler. *IEEE Trans. Computers*, 37(8):967–979, August 1988.
9. S. Liao et al. Code optimization techniques for embedded dsp microprocessors. In *DAC*, 1995.
10. T. M. Kemp et al. A decompression core for PowerPC. *IBM J. Res. & Dev.*, 42(6):807–812, November 1998.
11. L. Gwennap. Intel's P6 uses decoupled superscalar design. *Microprocessor Report*, 9(2):9–15, February 1995.
12. G. Kane. *MIPS RISC Architecture (R2000/R3000).* Prentice Hall, 1989.
13. Kevin D. Kissell. MIPS16: High-density MIPS for the embedded market. In *Proceedings RTS97*, 1997.
14. C. Lee, M. Potkanjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. In *Micro-30*, pages 330–335, December 1997.
15. H. Lekatsas and W. Wolf. Code compression for embedded systems. In *DAC*, pages 516–521, San Francisco, CA, June 1998.
16. IBM Microelectronics. PowerPC 440GP embedded processor: High performance SOP for networked applications. Presentation from Embedded Processor Forum, June 2000.
17. M. Panich. Reducing instruction cache energy using gated wordlines. Master's thesis, Massachusetts Institute of Technology, August 1999.
18. S. Segars, K. Clarke, and L. Goudge. Embedded control problems, Thumb, and the ARMT7TDMI. *IEEE Micro*, 15(5):22–30, October 1995.
19. SiByte, Inc. SB-1 CPU fact sheet. at www.sibyte.com, October 2000. rev. 0.1.
20. A. Wolfe and A. Chanin. Executing compressed programs on an embedded RISC architecture. In *MICRO 25*, pages 81–91, Portland, Oregon, December 1992.

Proceedings of the 15th International
Conference on Supercomputing, 2001.

Analytical Cache Models with Applications to Cache Partitioning

G. Edward Suh, Srinivas Devadas and Larry Rudolph

MIT Laboratory for Computer Science
{`suh, devadas, rudolph`}`@mit.edu`

Abstract. An accurate, tractable, analytic cache model for time-shared systems is presented, which estimates the overall cache miss-rate of a multiprocessing system with any cache size and time quanta. The input to the model consists of the isolated miss-rate curves for each process, the time quanta for each of the executing processes, and the total cache size. The output is the overall miss-rate. Trace-driven simulations demonstrate that the estimated miss-rate is very accurate. Since the model provides a fast and accurate way to estimate the effect of context switching, it is useful for both understanding the effect of context switching on caches and optimizing cache performance for time-shared systems. A cache partitioning mechanism is also presented and is shown to improve the cache miss-rate up to 25% over the normal LRU replacement policy.

1 Introduction

This paper presents an analytical model for the behavior of a cache in a multiprocessing system that can accurately estimate overall miss-rate for any cache size and any time quantum. An evaluation method for miss-rate is essential to optimize cache performance. Traditional cache performance evaluation is done by simulations [25, 16, 12], which provide accurate results, but simulation time is often long. Hardware monitoring can dramatically speed up the process [26], however, it is limited to the particular cache configuration. As a result, both simulations and hardware monitoring can only be used to evaluate the effect of context switches [14, 10]. Moreover, simulations and monitoring rarely provide intuitive understanding making it difficult to improve cache performance. To provide both performance prediction and insight into improving performance, analytical cache models are required.

We use our model to determine the best cache partitioning so as to improve performance. Partitioning is needed to mitigate the effects of conflicts among concurrently executing processes, especially for large caches. In the past, caches were small and it was best to let each process consume the entire cache space, since process footprints were much larger than the cache. In modern microprocessors, caches are much larger; some Level 1 (L1) caches range up to one MB [5], and L2 caches are up to several MB [2, 13]. Large caches allow potential performance improvement by partitioning. Since each process may not need the entire cache space, the effect of context switches can be mitigated by keeping useful data in the cache over context switches. It is crucial for modern microprocessors to minimize inter-process conflicts by proper cache partitioning [21, 9] or scheduling [17, 23].

Our model requires information that is relatively easy to acquire. The characteristics for each process are given by the miss-rate as a function of cache size when the process is isolated, which can be easily obtained either on-line or off-line. The time quantum for each process and cache size are also given as inputs to the model. With this information, the model estimates the overall miss-rate for a given cache size when an arbitrary combination of processes is run. The model provides good estimates for any cache size and any time quantum, and is easily applied to real problems since the input miss-rate curves are both intuitive and easy to obtain in practice. Therefore, we believe that the model is useful for any study related to the effect of context switches on cache memory.

After describing related research in Section 2, Section 3 derives an analytical cache model for time-shared systems. Section 4 discusses cache partitioning based on the model and evaluates the model-based partitioning method by simulations. Finally, Section 5 concludes the paper.

2 Related work

Several early investigations of the effects of context switches use analytical models. Thiébaut and Stone [20] modeled the amount of additional misses caused by context switches for set-associative caches. Agarwal, Horowitz and Hennessy [1] also included the effect of conflicts between processes in their analytical cache model and showed that inter-process conflicts are noticeable for a mid-range of cache sizes that are large enough to have a considerable number of conflicts but not large enough to hold all the working sets. However, these models work only for long enough time quanta, and require information that is hard to collect on-line.

Mogul and Borg [14] studied the effect of context switches through trace-driven simulations. Using a timesharing system simulator, their research shows that system calls, page faults, and a scheduler are the main sources of context switches. They also evaluated the effect of context switches on cycles per instruction (CPI) as well as the cache miss-rate. Depending on cache parameters, the cost of a context switch appears to be in the thousands of cycles, or tens to hundreds of microseconds in their simulations.

Stone, Turek and Wolf [18] investigated the optimal allocation of cache memory between two competing processes that minimizes the overall miss-rate of a cache. Their study focuses on the partitioning of instruction and data streams, which can be thought of as multitasking with a very short time quantum. Their model for this case shows that the optimal allocation occurs at a point where the miss-rate derivatives of the competing processes are equal. The LRU replacement policy appears to produce cache allocations very close to optimal for their examples. They also describe a new replacement policy for longer time quanta that only increases cache allocation based on time remaining in the current time quantum and the marginal reduction in miss-rate due to an increase in cache allocation. However, their policy simply assumes the probability for a evicted block to be accessed in the next time quantum as a constant, which is neither validated nor is it described how this probability is obtained.

Thiébaut, Stone and Wolf applied their partitioning work [18] to improve disk cache hit-ratios [21]. The model for tightly interleaved streams is extended to be applicable for more than two processes. They also describe the problems in applying the model in practice, such as approximating the miss-rate derivative, non-monotonic miss-rate derivatives, and updating the partition. Trace-driven simulations for 32-MB disk caches show that the partitioning improves the relative hit-ratios in the range of 1% to 2% over the LRU policy.

Our analytical model and partitioning differ from previous efforts that tend to focus on some specific cases of context switches. Our model works for any specific time quanta, whereas the previous models focus only on long time quanta. Also, our partitioning works for any time quanta, whereas Thiébaut's algorithms only works for very short time quanta. Moreover, the inputs of our model (miss-rates) are much easier to obtain compared to footprints or the number of unique cache blocks that previous models require.

3 Analytical cache model

The analytical cache model estimates the overall cache miss-rate for a multi-processing system. The cache size and the time quantum length for each job is known. The cache size is given by the number of cache blocks, and the time quantum is given by the number of memory references. Both are assumed to be constants (See Figure 1 (a)). In addition, associated with each job is its miss-rate curve, i.e., the number of cache misses as a function of the cache size.

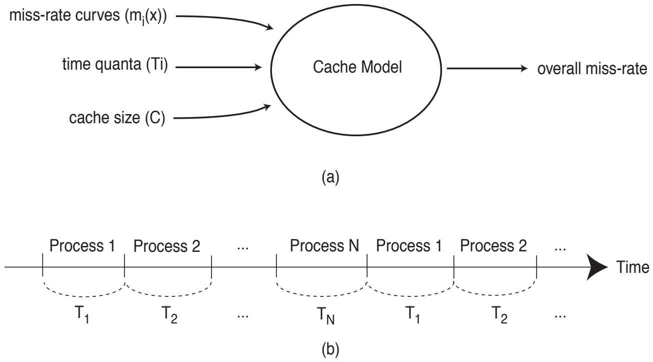


Fig. 1. (a) The overview of an analytical cache model. (b) Round-robin schedule.

This section explains the development of the model in several steps. Heavy use is made of the individual, isolated miss-rate curve (iimr). This curve is the miss-rate for a process as a function of cache size assuming no other processes are running. There is much information that can be gleaned from this equation. For example, we can compute the miss rate of a process as a function of time (Section 3.2) from the miss-rate of a process as a function of space.

Observe that as a process executes, it either references an item in the cache, in which case its footprint size remains the same, or it gets a cache miss thereby increasing its footprint size. In other words, we know how much cache is allocated to a process as a function of time: from the iimr curve, we compute the independent, isolated footprint as a function of time (iifp) (Section 3.2).

If one knows how much cache is allocated to a process when it begins executing its time quantum and how much more cache it will need during the execution of that time quantum, we can compute how much cache will be left for the next process that is about to begin its time quantum execution. In other words, from the iifp curves of all the concurrent processes, we compute the individual, dependent footprint (dfp) as a function of time (Section 3.2).

At each time step, we know how much cache is allocated to the running process (from dfp(t)) and we know the miss rate for that size (from iimr(S)) for the executing process and so we can get the dependent miss rate as a function of time (dmr(t)) (Section 3.2).

Finally, integrating or summing the dmr(t) over time, gives the overall average miss rate for a given cache size, given time quantum sizes, and a given set of concurrent processes (Section 3.2).

The following subsection gives an overview of our assumptions. The development of the cache model is then presented, following the outline given above. Finally, this section ends with experimental verification of the model.

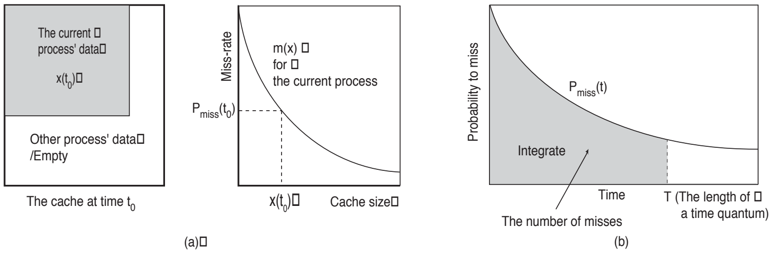


Fig. 2. (a) The probability of a miss at time t_0. (b) The number of misses from $P_{miss}(t)$ curve.

3.1 Assumptions

The memory reference pattern of each process is assumed to be represented by a miss-rate curve that is a function of the cache size. Moreover, this miss-rate curve is assumed not to change over time. Although real applications do have dynamically changing memory reference patterns, our results show that, in practice, an average miss-rate function works very well. For abrupt changes in the reference pattern, multiple miss-rate curves can be used to estimate an overall miss-rate.

There is no shared address space among processes. This assumption is true for common cases where each process has its own virtual address space and the shared memory space is negligible compared to the entire memory space that is used by a process.

Finally, a round-robin scheduling policy with a fixed time quantum for each process is assumed (see Figure 1 (b)), an LRU replacement policy is used, and the cache is fully associative. Although most real caches are set-associative, a model for fully-associative caches is very useful for understanding the effect of context switches because the model is simple. Moreover, cache partitioning experiments demonstrate that the fully-associative model can also be applied to set-associative caches in practice (Section 4). Elsewhere, we have extended the model to handle set-associative caches [19]. A model assuming many other scheduling methods and replacement policies can be similarly derived.

We make use of the following notations:

t the number of memory references from the beginning of a time quantum.
$x(t)$ the number of cache blocks that belong to a process after t memory references.
$m(x)$ the steady-state miss-rate for a process with cache size x.
T the number of memory references in a time quantum.

3.2 Cache model

The goal is to predict the average miss-rate for a multiprocess machine with a given cache size and set of processes.

Miss rate as function of time Given the independent, isolated miss-rate of a process as a function of cache size, we compute its miss-rate as a function of time. Let time t start at the beginning of a time quantum, not at the beginning of execution. Since all time quanta for a process are identical by our assumptions, we consider only one time quantum for each process.

Although the cache size is C, at certain times, it is possible that only part of the cache is filled with the current process' data (Figure 2 (a) shows a snapshot of a cache at time t_0). Therefore, the effective cache size at time t_0 can be thought of as the amount of the current process' data $x(t_0)$ in the cache at that time. The probability of a cache miss in the next memory reference is given by

$$P_{miss}(t_0) = m(x(t_0)). \tag{1}$$

Once we have $P_{miss}(t_0)$, it is easy to estimate the miss-rate over time during that time quantum. The number of misses for the process over a time quantum can be expressed as a simple integral, Figure 2 (b), where the miss-rate is expressed as the number of misses divided by the number of memory references.

$$\text{miss-rate} = \frac{1}{T}\int_0^T P_{miss}(t)dt = \frac{1}{T}\int_0^T m(x(t))dt \tag{2}$$

Footprint as a function of time We now estimate $x(t)$, the amount of a process' data, i.e. its footprint, in a cache as a function of time. Let us begin with the assumption that a process starts executing during a time quantum with an empty cache in order to estimate cache performance for cases when a cache gets flushed for every context switch. Virtual address caches without process ID are good examples of such a case. We show later how to estimate $x(t)$ when the cache is not empty at the start of a time quantum.

Consider $x^{\infty}(t)$ as the amount of the current process' data at time t for an infinite size cache. We assume that the process starts with an empty cache at time 0. There are two possibilities for $x^{\infty}(t)$ at time $t+1$. If the $(t+1)^{th}$ memory reference results in a cache miss, a new cache block is brought into the cache. As a result, the amount of the process's cache data increases by one block. Otherwise, the amount of data remains the same. Therefore, the amount of the process' data in the cache at time $t+1$ is given by

$$x^{\infty}(t+1) = \begin{cases} x^{\infty}(t)+1 & (t+1)^{th} \text{ reference misses} \\ x^{\infty}(t) & \text{otherwise.} \end{cases} \tag{3}$$

Since the probability for the $(t+1)^{th}$ memory reference to miss is $m(x^{\infty}(t))$ from Equation 1, the expected value of $x(t+1)$ can be written by

$$\begin{aligned} E[x^{\infty}(t+1)] &= E[x^{\infty}(t)\cdot(1-m(x^{\infty}(t))) \\ &\qquad + (x^{\infty}(t)+1)\cdot m(x^{\infty}(t))] \\ &= E[x^{\infty}(t)+1\cdot m(x^{\infty}(t))] \\ &= E[x^{\infty}(t)] + E[m(x^{\infty}(t))]. \end{aligned} \tag{4}$$

Assuming that $m(x)$ is convex[1], we can use Jensen's inequality [3] and rewrite the equation as a function of $E[x^{\infty}(t)]$.

$$E[x^{\infty}(t+1)] \geq E[x^{\infty}(t)] + m(E[x^{\infty}(t)]). \quad (5)$$

Usually, a miss-rate changes slowly. As a result, for a short interval such as from x to $x+1$, $m(x)$ can be approximated as a straight line. Since the equality in Jensen's inequality holds if the function is a straight line, we can approximate the amount of data at time $t+1$ as

$$E[x^{\infty}(t+1)] \simeq E[x^{\infty}(t)] + m(E[x^{\infty}(t)]). \quad (6)$$

We can calculate the expectation of $x^{\infty}(t)$ more accurately by calculating the probability for every possible value at time t [19]. However, calculating a set of probabilities is computationally expensive. Also, our experiments show that the above approximation closely matches simulation results.

If we further approximate the amount of data $x^{\infty}(t)$ to be the expected value $E[x^{\infty}(t)]$, $x^{\infty}(t)$ can be expressed with a differential equation:

$$x^{\infty}(t+1) - x^{\infty}(t) = m(x^{\infty}(t)), \quad (7)$$

which can be easily calculated in a recursive manner.

To obtain a closed form solution, we can rewrite the discrete form of the differential equation 7 to a continuous form:

$$\frac{dx^{\infty}}{dt} = m(x^{\infty}). \quad (8)$$

Solving the differential equation by separating variables, the differential equation becomes

$$t = \int_{x^{\infty}(0)}^{x^{\infty}(t)} \frac{1}{m(x')} dx'. \quad (9)$$

We define a function $M(x)$ as an integral of $1/m(x)$, which means that $dM(x)/dx = 1/m(x)$, and then $x^{\infty}(t)$ can be written as a function of t:

$$x^{\infty}(t) = M^{-1}(t + M(x^{\infty}(0))) \quad (10)$$

where $M^{-1}(x)$ represents the inverse function of $M(x)$.

Finally, for a finite size cache, the amount of data in the cache is limited by the size of the cache C. Therefore, $x^{\phi}(t)$, the amount of a process' data starting from an empty cache, is written by

$$x^{\phi}(t) = MIN[x^{\infty}(t), C] = MIN[M^{-1}(t + M(0)), C]. \quad (11)$$

---

[1] If a replacement policy is smart enough, the marginal gain of having one more cache block monotonically decreases as we increase the cache size.

Individual, dependent footprint as a function of time We now compute the amount of a process' data at time t when the cache is not flushed at a context switch, i.e., the dependent case. To distinguish between the processes, a subscript i is used to represent Process i. For example, $x_i(t)$ represents the amount of Process i's data at time t.

The estimation of $x_i(t)$ is based on round-robin scheduling (See Figure 1 (b)) and the LRU replacement policy. Process i runs for a fixed length time quantum T_i. For simplicity, processes are assumed to be of infinite length so that there is no change in the scheduling. Also, the initial startup transient from an empty cache is ignored since it is negligible compared to the steady state.

To estimate the amount of a process' data at a given time, imagine the snapshot of a cache after executing Process i for time t as shown in Figure 3. Note that time is 0 at the beginning of the process' time quantum. In the figure, the blocks on the left side show recently used data, and blocks on the right side show old data. $P_{j,k}$ represents the data of Process j, and subscript k specifies the most recent time quantum when the data are referenced. From the figure, we can obtain $x_i(t)$ once we know the size of all $P_{j,k}$ blocks.

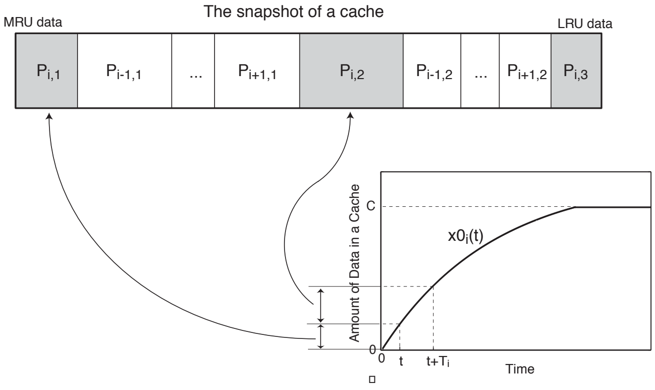


Fig. 3. The snapshot of a cache after running Process i for time t.

The size of each block can be estimated using the $x_i^{\phi}(t)$ curve from Equation 11, which is the amount of Process i's data when the process starts with an empty cache. Since $x_i^{\phi}(t)$ can also be thought of as the amount of data that are referenced from time 0 to time t, $x_i^{\phi}(T_i)$ is the amount of data that are referenced over one time quantum. Similarly, we can estimate the amount of data that are referenced over k recent time quanta to be $x_i^{\phi}(k \cdot T_i)$. As a result, the size of Block $P_{j,k}$ can be written as

$$P_{j,k} = \begin{cases} x_j^{\phi}(t + (k-1) \cdot T_j) - x_j^{\phi}(t + (k-2) \cdot T_j) & \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } j \text{ is executing} & \\ x_j^{\phi}(k \cdot T_j) - x_j^{\phi}((k-1) \cdot T_j) & \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise} & \end{cases} \tag{12}$$

where we assume that $x_j^{\phi}(t) = 0$ if $t < 0$.

$x_i(t)$ is the sum of $P_{i,k}$ blocks that are inside the cache of size C in Figure 3. If we define $l_i(t)$ as the maximum integer value that satisfies the following inequality, then $l_i(t) + 1$ represents how many $P_{i,k}$ blocks are in the cache.

$$\sum_{k=1}^{l_i(t)} \sum_{j=1}^{N} P_{j,k} = x_i^{\phi}(t + (l_i(t) - 1) \cdot T_i) + \sum_{j=1, j \neq i}^{N} x_j^{\phi}(l_i(t) \cdot T_j) \leq C \tag{13}$$

where N is the number of processes. From $l_i(t)$ and Figure 3, the estimated value of $x_i(t)$ is

$$x_i(t) = \begin{cases} x_i^{\phi}(t + l_i(t) \cdot T_i) & \text{if } x_i^{\phi}(t + l_i(t) \cdot T_i) + \\ & \displaystyle\sum_{j=1, j \neq i}^{N} x_j^{\phi}(l_i(t) \cdot T_j) \leq C \\ C - \displaystyle\sum_{j=1, j \neq i}^{N} x_j^{\phi}(l_i(t) \cdot T_j) & \text{otherwise} \end{cases} \tag{14}$$

Figure 4 illustrates the relation between $x_i^{\phi}(t)$ and $x_i(t)$. In the figure $l_i(t)$ is assumed to be 2. Unlike the cache flushing case, a process can start with some of its data left in the cache. The amount of initial data $x_i(0)$ is given by Equation 14. If the least recently used (LRU) data in a cache does not belong to Process i, $x_i(t)$ increases the same as $x_i^{\phi}(t)$. However, if the LRU data belongs to Process i, $x_i(t)$ does not increase on a cache miss since Process i's block gets replaced.

Define $t_{start}(j, k)$ as the time when the kth MRU block of Process j ($P_{j,k}$) becomes the LRU part of a cache, and $t_{end}(j, k)$ as the time when $P_{j,k}$ gets completely replaced from the cache (See Figure 3). $t_{start}(j, k)$ and $t_{end}(j, k)$ specify the flat segments in Figure 4 and can be estimated

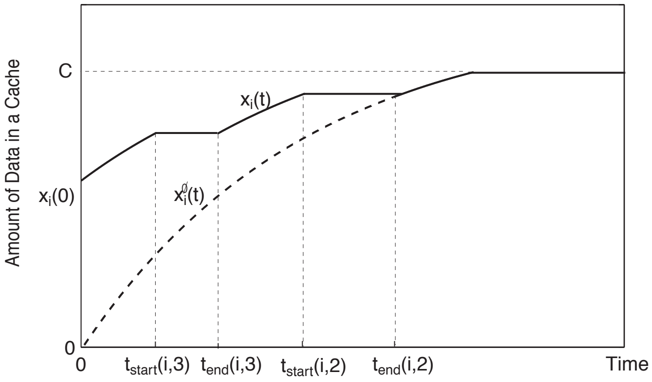


Fig. 4. The relation between $x_i^\phi(t)$ and $x_i(t)$. $x_i(0)$ is the amount of Process i's data in the cache when a time quantum starts.

from the following equations that are based on Equation 12.

$$x_j^\phi(t_{start}(j,k) + (k-1) \cdot T_j) + \sum_{p=1,p\neq j}^{N} x_p^\phi((k-1) \cdot T_p) = C. \tag{15}$$

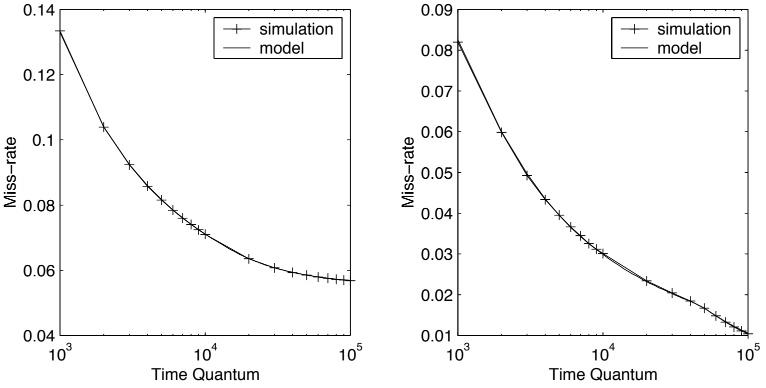
$$x_j^\phi(t_{end}(j,k) + (k-2) \cdot T_j) + \sum_{p=1,p\neq j}^{N} x_p^\phi((k-1) \cdot T_p) = C. \tag{16}$$

$t_{start}(j, l_j(t)+1)$ would be zero if Equation 15 is satisfied when $t_{start}(j, l_j(t)+1)$ is negative, which means that the $P(j, l_j(t)+1)$ block is already the LRU part of the cache at the beginning of a time quantum.

Overall miss-rate This section presents the overall miss-rate calculation. When a cache uses virtual address tags and gets flushed for every context switch, each process starts a time quantum with an empty cache. In this case, the miss-rate of a process can be estimated from the results of Section 3.2 and 3.2. From Equation 2 and 11, the miss-rate for Process i can be written by

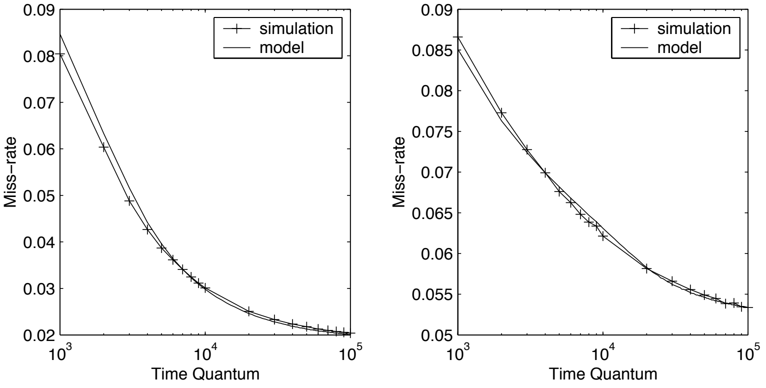
$$\text{miss-rate}_i^\phi = \frac{1}{T_i}\int_0^{T_i} m_i(MIN[M_i^{-1}(t + M_i(0)), C])dt. \tag{17}$$

If a cache uses physical address tags or has a process' ID with virtual address tags, it does not have to be flushed at a context switch. In this



(a) vpr

(b) vortex



(c) gcc

(d) bzip2

Fig. 5. The result of the cache model for cache flushing cases. (a) `vpr`. (b) `vortex`. (c) `gcc`. (d) `bzip2`.

case, the amount of data $x_i(t)$ is estimated in Section 3.2. The miss-rate for Process i can be written by

$$\text{miss-rate}_i = \frac{1}{T_i} \int_0^{T_i} m_i(x_i(t))dt \tag{18}$$

where $x_i(t)$ is given by Equation 14.

For actual calculation of the miss-rate, $t_{start}(j,k)$ and $t_{end}(j,k)$ from Equation 15 and 16 can be used. Since $t_{start}(j,k)$ and $t_{end}(j,k)$ specify the flat segments in Figure 4, the miss-rate of Process i can be rewritten by

$$\begin{aligned}\text{miss-rate}_i = \frac{1}{T_i}\{&\int_0^{T'_i} m_i(MIN[M_i^{-1}(t + M_i(x_i(0))), C])dt \\ &+ \sum_{k=d_i}^{l_i(t)+1} m_i(x_i^{\phi}(t_{start}(i,k) + (k-1)\cdot T_i)) \\ &\cdot (MIN[t_{end}(i,k), T_i] - t_{start}(i,k))\}\end{aligned} \tag{19}$$

where d_i is the minimum integer value that satisfies $t_{start}(i, d_i) < T_i$. T'_i is the time that Process i actually grows.

$$T'_i = T_i - \sum_{k=d_i}^{l_i(t)+1} (MIN[t_{end}(i,k), T_i] - t_{start}(i,k)). \tag{20}$$

As shown above, calculating a miss-rate could be complicated if we do not flush a cache at a context switch. If we assume that the executing process' data left in a cache is all in the most recently used part of the cache, we can use the equation for estimating the amount of data starting with an empty cache. Therefore, the calculation can be much simplified as follows,

$$\overline{\text{miss-rate}_i} = \frac{1}{T_i} \int_0^{T_i} m_i(MIN[M_i^{-1}(t + M_i(x_i(0))), C])dt \tag{21}$$

where $x_i(0)$ is estimated from Equation 14. The effect of this approximation is evaluated in the experiment section (cf. Section 3.3).

Once we calculate the miss-rate of each process, the overall miss-rate is straightforwardly calculated from those miss-rates.

$$\text{Overall miss-rate} = \frac{\sum_{i=1}^{N} \text{miss-rate}_i \cdot T_i}{\sum_{i=1}^{N} T_i} \tag{22}$$

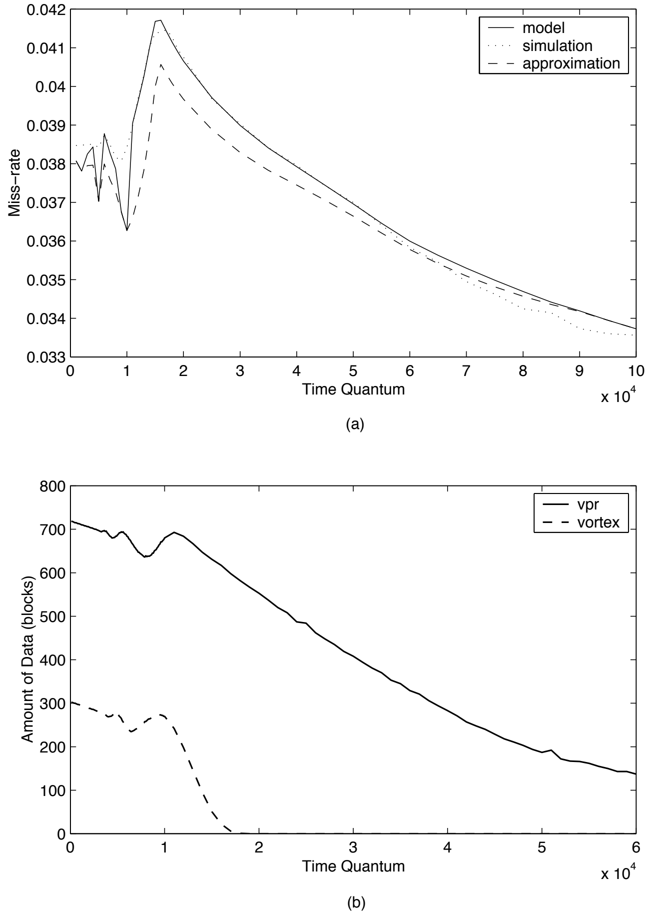


Fig. 6. The result of the cache model when two processes (`vpr`, `vortex`) are sharing a cache (32 KB fully-associative). (a) the overall miss-rate. (b) the initial amount of data $x_i(0)$.

3.3 Experimental verification

Our cache model can be validated by comparing estimated miss-rate predictions with simulation results. Several combinations of benchmarks are modeled and simulated for various time quanta. First, we simulate cases when a cache gets flushed at every context switch, and compare the results with the model's estimation. Cases without cache flushing are also tested. For the cases without cache flushing, both the complete model (Equation 19) and the approximation (Equation 21) are used to estimate the overall miss-rate. Based on the simulation results, the error caused by the approximation is discussed.

Cache flushing case The results of the cache model and simulations are shown in Figure 5 in cases when a process starts its time quantum with an empty cache. Four benchmarks from SPEC CPU2000 [7], which are `vpr`, `vortex`, `gcc` and `bzip2`, are tested. The cache is a 32-KB fully-associative cache with 32-Byte blocks. The miss-rate of a process is plotted as a function of the length of a time quantum, and shows a good agreement between the model's estimation and the simulation result.

As inputs to the cache model, the average miss-rate of each process has been obtained from simulations. Each process has been simulated for 25 million memory references, and the miss-rates of the process for various cache size have been recorded. The simulation results were also obtained by simulating benchmarks for 25 million memory references with flushing a cache every T memory references. As the result shows, the average miss-rate works very well.

General case Figure 6 shows the result of the cache model when two processes are sharing a cache. The two benchmarks are `vpr` and `vortex` from SPEC CPU2000, and the cache is a 32-KB fully-associative cache with 32-Byte blocks. The overall miss-rates are shown in Figure 6 (a). As shown in the figure, the miss-rate estimated by the model shows a good agreement with the results of the simulations.

The figure also shows an interesting fact that a certain range of time quanta could be very problematic for cache performance. For short time quanta, the overall miss-rate is relatively small. For very long time quanta, context switches do not matter since a process spends most of its time in the steady state. However, medium time quanta could severely degrade cache miss-rates as shown in the figure. This problem occurs when a time quantum is long enough to pollute the cache but not long enough to compensate for the misses caused by context switches. The problem becomes clear in Figure 6 (b). The figure shows the amount of data left in the cache at the beginning of a time quantum. Comparing Figure 6 (a) and (b), we can see that the problem occurs when the initial amount of data rapidly decreases.

The error caused by our approximation (Equation 21) method can be seen in Figure 6. In the approximation, we assume that the data left in the cache at the beginning of a time quantum are all in the MRU region of the cache. In reality, however, the data left in the cache could be the LRU cache blocks and get replaced before other process' blocks in the cache, although the current process's data are likely to be accessed in the time quantum. As a result, the approximated miss-rate is lower than the simulation result when the initial amount of data is not zero.

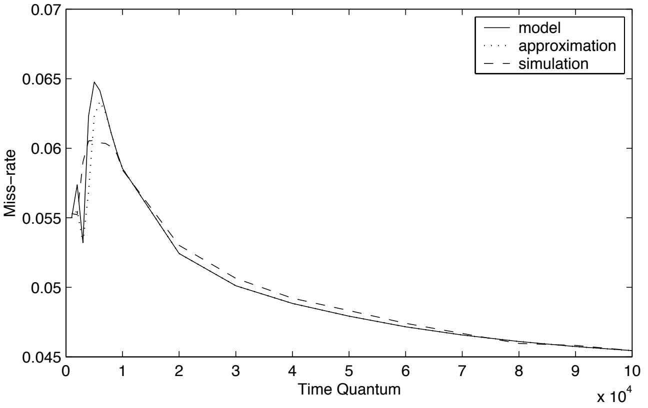


Fig. 7. The overall miss-rate when four processes (`vpr`, `vortex`, `gcc`, `bzip2`) are sharing a cache (32 KB, fully-associative).

A four-process case is also tested in Figure 7. Two more benchmarks, `gcc` and `bzip2`, from SPEC CPU2000 [7] are added to `vpr` and `vortex`, and the same cache configuration is used as the two process case. The figure also shows a very close agreement between the miss-rate estimated by the cache model and the miss-rate from simulations. The problematic time quanta and the effect of the approximation have changed. Since there are more processes polluting the cache as compared to the two process case, a process experiences an empty cache in shorter time quanta. As a result, the problematic time quanta become shorter. On the other hand, the effect of the approximation is less harmful in this case. This is because the error in one process' miss-rate becomes less important as we have more processes.

4 Cache partitioning

This section shows how the analytical cache model can be used to dynamically partition the cache. A partitioned cache allocates cache space to particular processes. This space is dedicated to the process and cannot be used to satisfy cache misses by other processes. Using trace-driven simulations, we compare partitioning with the normal LRU. The partitioning is based on the fully-associative cache model. However, simulation results demonstrate that this implementation works for both fully-associative caches and set-associative caches.

4.1 Recording memory reference patterns

The miss-rate curves for each process are generated off-line. We record the miss-rate curve for each process to represent its memory reference pattern. For various cache sizes, a single process cache simulator is applied to each process. This information can be reused for any combination of processes as long as the cache configuration is the same[2].

To incorporate the dynamically changing behavior of a process, a set of miss-rate curves, one for each time period, are produced. At run-time, the miss-rate curve is mapped to the appropriate time quantum.

4.2 The partitioning scheme

The overall flow of the partitioning scheme can be viewed as a set of four modules: off-line recording, scheduler information, allocation, and replacement (Figure 8). The scheduler provides the partition module with the set of executing processes and their start/end times. The partition module uses the miss-rate information for the processes to calculate cache partitions at the end of each time quantum. Finally, the replacement unit maps these partitions to the appropriate parts of the cache.

The partition module decides the number of cache blocks that should be dedicated to a process (D_i). The D_i most recently used cache blocks of Process i are kept in the cache over other process' time quanta, and

---

[2] Note that for our fully-associative model, only the cache block size matters

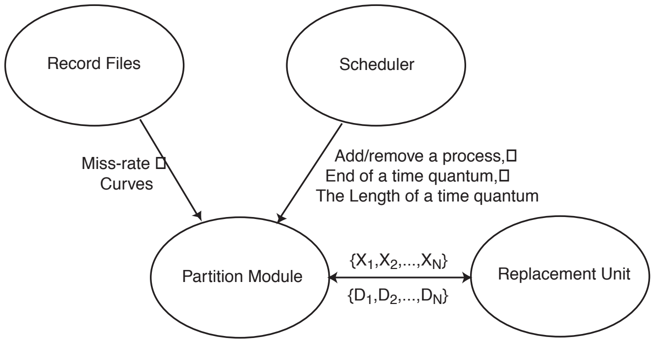


Fig. 8. The implementation of on-line cache partitioning.

Process i starts its time quantum with those cache blocks in the cache. During its own time quantum, Process i can use all cache blocks that are not reserved for other processes ($S = C - \sum_{j=1,j \neq i}^{N} D_j$).

In addition to LRU information, our replacement decision depends on the number of cache blocks that currently belong to each process (X_i), that is, the number of cache lines in the cache that currently contain memory of that process. The LRU cache block of an active process (i) is chosen if its actually allocation (X_i) is larger than or equal to the desired one ($D_i + S \leq X_i$). Otherwise, the LRU cache block of a dormant overallocated process is chosen. For set-associative caches, there may be no cache block of the desired process in the set. In this case, the LRU cache block of the set is replaced.

For set-associative caches, the fully-associative replacement policy may result in replacing recently used data to keep useless data. Imagine the case when a process starts to heavily access two or more addresses that happen to be mapped to the same set. If the process already has many cache blocks in other sets, our partitioning will allocate only a few cache blocks in the accessed set for the process, causing lots of conflict misses. To solve this problem, we can use better mapping functions [22, 6] or a victim cache [8].

When a Process i first starts, D_i is set to zero since there is no cache block that belongs to the process. At the end of Process i's time quantum, the partition module updates the information such as the miss-rate curve($m_i(x)$) and the time quantum(T_i). If there is any change, D_i is also updated based on the cache model.

A cache partition specifies the amount of data in the cache at the beginning of a process' time quantum (D_i), and the maximum cache

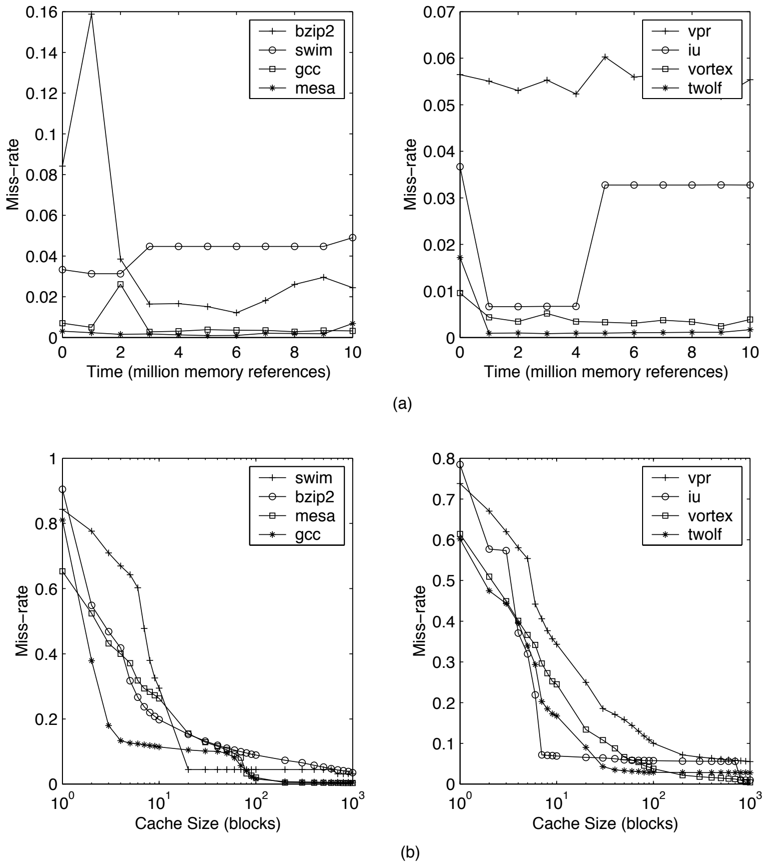


Fig. 9. The characteristics of the benchmarks. (a) The change of a miss-rate over time. (b) The miss-rate as a function of the cache size.

space the process can use ($C - \sum_{j=1,j\neq i}^{N} D_j$). Therefore, the number of misses for a process over one time quantum can be estimated from Equation 21:

$$\text{miss}_i = \int_0^{T_i} m_i(MIN[M_i^{-1}(t + M_i(D_i)), C - \sum_{j=1,j\neq i}^{N} D_j])dt \tag{23}$$

where C is cache size, and N is the number of processes sharing the cache.

The new value of D_i is the integer, in the range $[0, X_i]$, that minimizes the total number of misses that is given by the following quantity:

$$\sum_{p=1}^{N} \int_0^{T_p} m_p(MIN[M_p^{-1}(t + M_p(D_p)), C - \sum_{q=1,q\neq p}^{N} D_q])dt. \tag{24}$$

4.3 Experimental verification

The case of eight processes sharing a 32-KB cache is simulated to evaluate model-based partitioning. Seven benchmarks (`bzip2`, `gcc`, `swim`, `mesa`, `vortex`, `vpr`, `twolf`) are from SPEC CPU2000 [7], and one (the image understanding program (`iu`)) is from a data intensive systems benchmark suite [15]. The overall miss-rate with partitioning is compared to the miss-rate only using the normal LRU replacement policy.

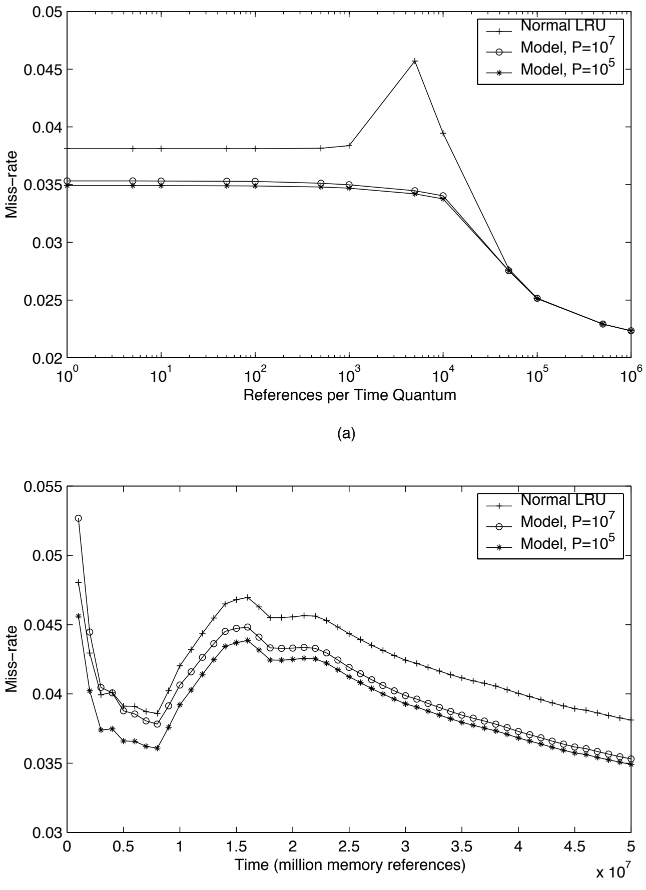
The simulations are carried out for fifty million memory references for each time quantum. Processes are scheduled in a round-robin fashion with the fixed number of memory references per time quantum. Also, the number of memory references per time quantum is assumed to be the same for the all eight processes. Finally, two record cycles (P), of ten million and one hundred thousand memory references, respectively, are used for the model-based partitioning. The record cycle represents how often the miss-rate curve is recorded for the off-line profiling. Therefore, a shorter record cycle implies more detailed information about a process' memory reference pattern.

The characteristics of the benchmarks are illustrated in Figure 9. Figure 9 (a) shows the change of a miss-rate over time. The x-axis represents simulation time. The y-axis represents the average miss-rate over one million memory references at a given time. As shown in the figure, `bzip2`, `gcc`, `swim` and `iu` show abrupt changes in their miss-rate, whereas other benchmarks have very uniform miss-rate characteristics over time. Figure 9 (b) illustrates the miss-rate as a function of the cache size. For a 32-KB fully-associative cache, benchmarks show miss-rates between 1% and 5%.

Fully-associative result The results of cache partitioning for a fully-associative cache are shown in Figure 10. In Figure 10 (a), the miss-rates are averaged over 50 million memory references and shown for various time quanta. As discussed in the cache model, the normal LRU replacement policy is problematic for a certain range of time quanta. In this case, the overall miss-rate increases dramatically for time quanta between one thousand and ten thousand memory references. For this problematic region, the model-based partitioning improves the cache miss-rate by lowering it from 4.6% to 3.4%, which is about a 25% improvement. For short time quanta, the relative improvement is about 7%. For very long time quanta, the model-based partitioning shows the exact same result as the normal LRU replacement policy. In general, it is shown by the figure that the model-based partitioning always performs at least as well as or better than the normal LRU replacement policy. Also, the partitioning with a short record cycle performs better than the partitioning with a long record cycle.

In our example of a 32-KB cache with eight processes (Figure 10), the problematic time quanta are in the order of a thousand memory references, which is very short for modern microprocessors. As a result, only systems with very fast context switching, such as simultaneous multi-threading machines [24, 11, 4], can be improved for this cache size and workload. However, longer time quanta become problematic if a cache is larger. Therefore, conventional time-shared systems with very high clock frequency can also be improved by the same technique if a cache is large.

Figure 10 (b) shows the change of a miss-rate over time rather than an average miss-rate over the entire simulation. It is clear from the figure how the short record cycle helps partitioning. In the figure, the model-based partitioning with the long record cycle ($P = 10^7$) performs worse than LRU at the beginning of a simulation, even though it outperforms the normal LRU replacement policy overall. This is because the model-based partitioning has only one average miss-rate curve for a process. As shown in Figure 9, some benchmarks such as `bzip2` and `gcc` have a very different miss-rate at the beginning. Therefore, the average miss-rate curves for those benchmarks do not work at the beginning of the simulation, which results in worse performance than the normal LRU replacement policy. The model-based partitioning with the short record cycle ($P = 10^5$), on the other hand, always outperforms the normal LRU replacement policy. In this case, the model has correct miss-rate curves for all the time quanta, and partitions the cache properly even for the beginning of processes.



(a)

(b)

Fig. 10. The results of the model-based cache partitioning for a fully-associative cache when eight processes (`bzip2`, `gcc`, `swim`, `mesa`, `vortex`, `vpr`, `twolf`, `iu`) are sharing the cache (32 KB, fully associative). (a) the average miss-rate for various time quanta. (b) the change of the miss-rate over time with ten memory references per time quantum.

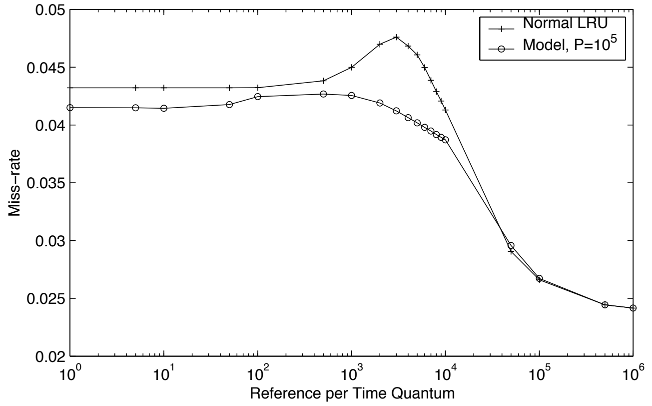


Fig. 11. The results of the model-based cache partitioning for a set-associative cache when eight processes (`bzip2`, `gcc`, `swim`, `mesa`, `vortex`, `vpr`, `twolf`, `iu`) are sharing the cache (32 KB, 8-way associative).

Set-associative result The result of cache partitioning for a set-associative cache is shown in Figure 11. The same set of benchmarks are simulated with a 32-KB 8-way set-associative cache, and the same miss-rate curves generated for a 32-KB fully-associative cache are used. In this case, a 16 entry victim cache is added. In the figure, the model-based partitioning improves the miss-rate about 4% for short time quanta and up to 15% for mid-range time quanta. The figure demonstrates that the model-based partitioning mechanism works reasonably well for set-associative caches.

5 Conclusion

An analytical cache model to estimate overall miss-rate when multiple processes are sharing a cache has been presented. The model obtains the information about each process from its miss-rate curve, and combines it with parameters that define the cache configuration and schedule of processes. Interference among processes under the LRU replacement policy is quickly estimated for any cache size and any time quantum, and the estimated miss-rate is very accurate. A more important result is that the model provides not only the overall miss-rate but also a very good understanding of the effect of context switching. For example, the model clearly shows that the LRU replacement policy is problematic for mid-range time quanta because the policy replaces the blocks of least recently executed process that are more likely to be accessed in the near future.

The analytical model has been applied to the cache partitioning problem. A model-based partitioning method has been implemented and verified by simulations. Miss-rate curves are recorded off-line and partitioning is performed on-line according to the combination of processes that are executing. Even though we have used an off-line profiling method to obtain miss-rate curves, it should not be hard to approximate the miss-rate curve on-line using a miss-rate monitoring technique. Therefore, a fully on-line cache partitioning method can be developed based on the model.

Only the cache partitioning problem has been studied in this paper. However, as shown by the study of cache partitioning, our model can be applied to any cache optimization problem that is related to the problem of context switching. For example, it can be used to determine the best combination of processes that can be run on each processor of a multi-processor system. Also, the model is useful to identify areas in which further research in improving cache performance would be fruitful since it can easily provide the maximum improvement we can expect in the area.

6 Acknowledgments

Funding for this work is provided in part by the Defense Advanced Research Projects Agency under the Air Force Research Lab contract F30602-99-2-0511, titled “Malleable Caches for Data-Intensive Computing”. Thanks also to E. Peserico, D. Chiou, and D. Chen for their comments on the cache model.

References

1. Anant Agarwal, Mark Horowitz, and John Hennessy. An analytical cache model. *ACM Transactions on Computer Systems*, 7(2), May 1989.
2. Compaq. Compaq alphastation family.
3. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley, John & Sons, Incorporated, March 1991.
4. Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, and Dean M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17(5), 1997.
5. Catherine Freeburn. The hewlett packard PA-RISC 8500 processor. Technical report, Hewlett Packard Laboratories, October 1998.

6. Antonio González, Mateo Valero, Nigel Topham, and Joan M. Parcerisa. Eliminating cache conflict misses through XOR-based placement functions. In *the 1997 international conference on Supercomputing*, 1997.
7. John L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, July 2000.
8. N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *the 17th Annual International Symposium on Computer Architecture*, 1990.
9. David B. Kirk. Process dependent static cache partitioning for real-time systems. In *Real-Time Systems Symposium*, 1988.
10. Hantak Kwak, Ben Lee, Ali R. Hurson, Suk-Han Yoon, and Woo-Jong Hahn. Effects of multithreading on cache performance. *IEEE Transactions on Computers*, 48(2), February 1999.
11. Jack L. Lo, Joel S. Emer, Henry M. Levy, Rebecca L. Stamm, Dean M. Tullsen, and S. J. Eggers. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, 15, 1997.
12. Peter Magnusson and Bengt Werner. Efficient memory simulation in SimICS. In *28th Annual Simulation Symposium*, 1995.
13. MIPS Technologies, Inc. *MIPS R10000 Microprocessor User's Manual*, 1996.
14. Jeffrey C. Mogul and Anita Borg. The effect of context switches on cache performance. In *the fourth international conference on Architectural support for programming languages and operating systems*, 1991.
15. José Muoz. *Data-Intensive Systems Benchmark Suite Analysis and Specification*. http://www.aaec.com/projectweb/dis, June 1999.
16. Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta. Complete computer system simulation: The SimOS approach. *IEEE Parallel & Distributed Technology*, 1995.
17. Mark S. Squillante and Edward D. Lazowska. Using processor-cache affinity information in shared-momory multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 4(2), February 1993.
18. Harold S. Stone, John Turek, and Joel L. Wolf. Optimal partitioning of cache memory. *IEEE Transactions on Computers*, 41(9), September 1992.
19. Gookwon Edward Suh and Larry Rudolph. Set-associative cache models for time-shared systems. Technical Report CSG Memo 433, Massachusetts Institute of Technology, 2001.
20. Dominique Thiébaut and Harold S. Stone. Footprints in the cache. *ACM Transactions on Computer Systems*, 5(4), November 1987.
21. Dominique Thiébaut, Harold S. Stone, and Joel L. Wolf. Improving disk cache hit-ratios through cache partitioning. *IEEE Transactions on Computers*, 41(6), June 1992.
22. Nigel Topham and Antonio González. Randomized cache placement for eleminating conflicts. *IEEE Transactions on Computers*, 48(2), February 1999.

23. Josep Torrellas, Andrew Tucker, and Anoop Gupta. Benefits of cache-affinity scheduling in shared-memory multiprocessors: A summary. In *the 1993 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, 1993.
24. Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture*, 1995.
25. Richard A. Uhlig and Trevor N. Mudge. Trace-driven memory simulation: A survey. *ACM Computing Surveys*, 29(2), June 1997.
26. Marco Zagha, Brond Larson, Steve Turner, and Marty Itzkowitz. Performance analysis using the MIPS R1000. In *Supercomputing'96*, 1996.

13th IASTED International Conference
on Parallel and Distributed Computing
Systems, 2001.

Dynamic Cache Partitioning for Simultaneous Multithreading Systems

G. Edward Suh, Larry Rudolph and Srinivas Devadas

MIT Laboratory for Computer Science
{`suh, rudolph, devadas`}`@mit.edu`

Abstract. This paper proposes a dynamic cache partitioning method for simultaneous multithreading systems. We present a general partitioning scheme that can be applied to set-associative caches at any partition granularity. Furthermore, in our scheme threads can have overlapping partitions, which provides more degrees of freedom when partitioning caches with low associativity.

Since memory reference characteristics of threads can change very quickly, our method collects the miss-rate characteristics of simultaneously executing threads at run-time, and partitions the cache among the executing threads. Partition sizes are varied dynamically to improve hit rates. Trace-driven simulation results show a relative improvement in the L2 hit-rate of up to 40.5% over those generated by the standard least recently used replacement policy, and IPC improvements of up to 17%. Our results show that smart cache management and scheduling is important for SMT systems to achieve high performance.

1 Introduction

Microprocessors with multiple functional units have low IPC (Instructions per Cycle) rates either because of a lack of parallelism, or because of a high incidence of data dependencies. Simultaneous Multi-Threading, [15, 10, 6] (SMT), helps in the former case but, in the latter case, it only exacerbates the stress on the memory subsystem, especially since the standard LRU replacement scheme treats all references in the same way. Thus, a single thread can easily "pollute" the cache with its data, causing higher miss rates for other threads, and resulting in low overall performance.

This paper presents a dynamic cache partitioning algorithm that minimizes the overall cache miss rate for simultaneous multithreading systems. Rather than relying on the standard least recently used (LRU) cache replacement policy, our algorithm dynamically allocates parts of the cache to the most needy threads using on-line estimates of individual thread miss rates. The cache is assumed to be large enough to support multiple contexts, but not large enough to hold all of the working sets of the simultaneously executing threads. Although a 1997 study has shown that a 256-KB L2 cache, which is reasonable size for modern microprocessors [8, 5, 11], is large enough for a particular set of workloads [10], we believe that workloads have become much larger and diverse; multimedia programs such as video or audio processing software often consume hundreds of MB and many SPEC CPU2000 benchmarks now have memory footprints larger than 100 MB [9].

We propose a novel cache partitioning scheme wherein a cache miss will only allocate a new cache block to a thread if its current allocation is below its limit. To implement this scheme, we require counters in the cache that provide on-line estimates of individual thread miss rates. Based on these counters, we can augment LRU replacement to better allocate cache resources to threads, or we can use Column caching [3], which allows threads to be assigned to overlapping partitions, to partition the cache. Simulation shows that the partitioning algorithm can significantly improve both the miss-rate and the instructions per cycle (IPC) of the overall workload.

In conventional time-shared systems, cache partitioning depends not only on the active thread, but also on the memory reference pattern of inactive threads which have run in the past, and will run again in the near future. On the other hand, in SMT systems, multiple threads are active at the same time, collectively stressing the memory system. Since these threads very quickly use up cache resources once they start running, partitioning depends only on the memory reference characteristics of the set of active threads. This differs from traditional time sharing systems where one must also consider the length of the time quantum and the characteristics of the ready, but not executing threads. Since the memory references from each thread are interleaved very tightly, one can consider an SMT system to be a traditional time-sharing system with a context switch at each memory reference.[1]

---

[1] In many systems, each page fault or disk access causes a context switch and so disk cache partitioning schemes are somewhat relevant to SMT cache partitioning.

This paper is organized as follows. In Section 2, we describe related work. In Section 3, we first study the optimal cache partitioning problem for the ideal case of fully associative caches that are partitionable on a cache-block basis. We then extend our method to the more realistic set-associative cache case. Section 4 evaluates the partitioning method by simulations. Finally, Section 5 concludes the paper.

2 Related work

Stone, Turek and Wolf [12] investigated the optimal allocation of cache memory between two competing processes that minimizes the overall miss-rate of a cache. Their study focuses on the partitioning of instruction and data streams, which can be thought of as multitasking with a very short time quantum, and shows that the optimal allocation occurs at a point where the miss-rate derivatives of the competing processes are equal. The LRU replacement policy appears to produce cache allocations very close to optimal for their examples.

In previous work [13] we proposed an analytical cache model for multitasking, and also studied the cache partitioning problem for time-shared systems based on the model. That work is applicable to any length of time quantum rather than just short time quantum, and shows that the cache performance can be improved by partitioning a cache into dedicated areas for each process and a shared area. However, the partitioning was performed by collecting the miss-rate information of each process off-line. The work of [13] did not investigate how to partition the cache memory at run-time.

Thiébaut, Stone and Wolf applied their theoretical partitioning study [12] to improve disk cache hit-ratios [14]. The model for tightly interleaved streams is extended to be applicable for more than two processes. They also describe the problems in applying the model in practice, such as approximating the miss-rate derivative, non-monotonic miss-rate derivatives, and updating the partition. Trace-driven simulations for 32-MB disk caches show that the partitioning improves the relative hit-ratios in the range of 1% to 2% over the LRU policy.

Our partition work differs from previous efforts. It works for set-associative caches with multiple threads and a coarse-grained partition, whereas Thiébaut et al. [14] only focused on disk caches that are fully-associative with cache block granularity. Finally, this work discusses an on-line method to partition the cache, whereas our previous only covered partitioning based on off-line profiling [13].

3 Partitioning algorithm

This section presents our cache partitioning algorithm. We lead up to a general partitioning method in several steps. First, given a fully-associative cache that can be partitioned on a cache-block basis and knowing the miss-rate for each task as a function of partition size, we show how an optimal partition is obtained by iteratively increasing the partition size for the thread that will benefit the most. Next, we show that it is possible to compute the miss rate functions on-line using many hardware counters for a fully associative cache, and that it is possible to approximate the miss-rate function using fewer counters in the case of a set-associative cache. These results are then combined and applied to the more practical case of coarse grained partitioning. Finally, the algorithm to actually allocate cache blocks to each thread is developed.

3.1 Optimal cache partitioning

Given N executing threads sharing a cache of C blocks with partitioning on a cache block granularity, the problem is to partition the cache into N disjoint subsets of cache blocks so as to minimize the overall miss-rate. For each thread, the miss-rate as a function of partition size (the number of cache blocks), is known. Let c_i represent the number of cache blocks allocated to the ith thread. A cache partition is specified by the number of cache blocks allocated to each thread, i.e., $\{c_1, c_2, ..., c_N\}$. Since it is unreasonable to repartition the cache every memory reference, the partition remains fixed over a time period, π, that is long enough to amortize the repartitioning cost.

The number of cache misses for the ith thread over π is given by a function of partition size ($m_i(x)$). The optimal partition for the period is the set of integer values $\{c_1, c_2, ..., c_N\}$, that minimizes the following expression:

$$\text{total misses over time period } \pi = \sum_{i=1}^{N} m_i(c_i) \tag{1}$$

under the constraint that $\sum_{i=1}^{N} c_i = C$. C is the total number of blocks in the cache.

For the case where the number of misses for each thread is a strict convex function of cache space, Stone, Turek and Wolf [12] noted that finding the optimal partition, $\{c_1, c_2, ..., c_N\}$, falls into the category of separable convex resource allocation problems. The following, well-known, simple greedy algorithm yields an optimal partition [12, 7]:

1. Let the marginal gain, $g_j(x)$, be the number of additional hits for the jth thread, when the allocated cache blocks increases from x to $x+1$.
2. Initialize $c_1 = c_2 = ... = c_N = 0$.
3. *Increase by one the number of cache blocks assigned to the thread that has the maximum marginal gain given the current allocation.* Increase c_j by one, where j is the index for which $g_j(c_j)$ is largest.
4. Repeat step 3 until all cache blocks are assigned (i.e C times).

3.2 Computing the marginal gain

The computation of the marginal gain, $g_i(x)$, depends on the the miss rate for task i as a function of the cache partition size, $m_i(x)$, over a time period, π. For a fully associative LRU cache, it is possible to compute $m_i(x)$ on-line using C counters. When a task references a data item in the cache that is the kth most recently referenced item, then counter k for task i is increased. At the end of the time period, these counters form the miss rate function for each task, as described below. The description below applies to the general case of a set-associative cache.

To perform dynamic cache partitioning, the marginal gains of having one more cache block can be estimated on-line. As discussed in the previous section, $g_i(x)$ is the number of additional hits that the ith thread can obtain by having $x+1$ cache blocks compared to the case when it has x blocks. Assuming the LRU replacement policy is used, $g_i(0)$ represents the number of hits on the most recently used cache block of the ith thread, $g_i(1)$ represents the number of hits on the second most recently used cache block of the ith thread, and so on.

For each thread, a set of counters, one for each associativity (way) of the cache, is maintained. On every cache hit, the corresponding counter is increased. That is, if the hit is on the most recently used cache block of the thread, the first counter is increased by one, and so on. The kth counter value represents the number of additional hits for the thread by having the kth way. If we ignore the degradation due to low associativity, the kth counter value can also be thought of as the number of additional hits for a cache with $k \cdot S$ blocks compared to a cache with $(k-1) \cdot S$ blocks, where S is the number of cache sets. Therefore, $g_i(x)$ satisfies the following equation.

$$\sum_{x=(k-1)\cdot S}^{k\cdot S-1} g_i(x) = count_i(k) \tag{2}$$

where $count_i(k)$ represents the kth counter value of the ith thread.

To estimate marginal gains from Equation 2, assume that $g_i(x)$ is a straight line for x between $k \cdot S$ and $(k+1) \cdot S - 1$. This approximation is very simple to calculate and yet shows reasonable performance in partitioning. This is especially true in the case of large L2 (level 2) caches, which only see memory references that are filtered by L1 (level 1) caches, and often show the miss-rate that is proportional to cache size. To be more accurate, $g_i(x)$ can be assumed to be a form of an power function, e.g., $a \cdot x^b$. Empirical studies showed that the power function often accurately estimates the miss-rate [4].

Since characteristics of threads change dynamically, the estimation of $g_i(x)$ should reflect the changes. This is achieved by giving more weight to the counter value measured in more recent time periods. After every T memory references, we multiply each counter by δ, which is between 0 and 1. As a result, the effect of hits in previous time periods exponentially decays.

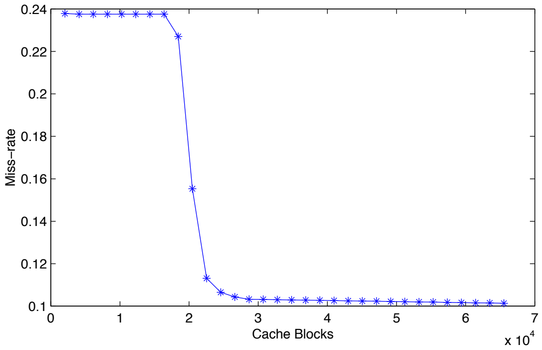


Fig. 1. The miss-rate of `art` as a function of cache blocks.

The number of misses for a real application is often not strictly convex as illustrated in Figure 1. The figure shows the miss-rate curve of `art` from the SPEC CPU2000 benchmark suite [9] for a 32-way 1-MB cache. As long as the miss-rate curve is convex, the marginal gain function decreases, and at the non-convex points, the marginal gain function will increase. In theory, every possible partition should be compared to obtain the optimal partition for non-convex miss-rate curves. However, non-convex curves can be approximated by a combination of a few convex curves. For example, the miss-rate of `art` can be approximated by two convex curves, one before the steep slope and one after that. Once a curve only has a few non-convex points, the convex resource allocation algorithm can be used to guarantee the optimal solution for non-convex cases.

1. For each thread, i, compute the ρ_i non-convex points of its miss-rate curve: $\{p_{i,1}, p_{i,2}, ..., p_{i,\rho_i}\}$, $g_i(p_{i,j}) < g_i(p_{i,j}+1)$.
2. Execute the convex algorithm with c_i initialized to 0 or $p_{i,j}$, $\forall j$.
3. Repeat step 2 for all possible initializations, and choose the partition that results in the maximum $\sum_{i=1}^{N} m_i(c_i)$.

3.3 Coarse granularity partitioning

Since it is rather expensive to control each cache block, practical partitioning mechanisms perform allocation of chunks of cache blocks, referred to as a *partition block*. We will use D to refer to the number of cache blocks in a partition block. We allow the allocation of one partition block to multiple threads and let the replacement policy decide the cache block level partition.

First, consider the no sharing case where each partition block is allocated to only one thread. The algorithm for cache block granularity partitioning can be directly applied. Define the partition marginal gain as $g_i(x) = m_i(x \cdot D) - m_i((x+1) \cdot D)$ and use the greedy algorithm to assign one partition block at a time, resulting in an optimal partition without sharing. However, sharing a partition block is essential to achieve high performance with coarse granularity partitioning. For example, when there are many more threads than partition blocks. It is obvious that threads must share partition blocks in order to use the cache.

Knowing the number of misses for each thread as a function of cache space, the effect of sharing partition blocks can be evaluated once the allocation of the shared blocks by the LRU replacement policy is known. Consider the case when N_{share} threads share B_{share} partition blocks. Since each partition block consists of D cache blocks, the case can be thought of as N_{share} threads sharing $B_{share} \cdot D$ cache blocks. Since SMT systems tightly interleave memory references of the threads, the replacement policy can be thought of as random.

Define $B_{dedicate,i}$ as the number of partition blocks that are allocated to the ith thread exclusively, and x_i as the number of cache blocks that belongs to the ith thread. Since the replacement can be considered as random, the number of replacements for a certain cache region is proportional to the size of the region.

The number of misses that replaces the cache block in the shared space $m_{share,i}(x)$ can be estimated as follows.

$$m_{share,i}(x) = \frac{B_{share}}{B_{dedicate,i} + B_{share}} \cdot m_i(x). \tag{3}$$

Under the random replacement, the number of cache blocks belonging to each process for the shared area is proportional to the number of cache blocks that each process brings into the shared area. Therefore, x_i can be written by

$$x_i = B_{dedicate,i} \cdot S + \frac{m_{share,i}(x_i)}{\sum_{j=1}^{N} m_{share,j}(x_j)} \cdot (B_{share} \cdot S). \tag{4}$$

Since x_i is on both the left and right sides of Equation 4, an iterative method can be used to estimate x_i starting with a initial value that is between $B_{dedicate,i} \cdot S$ and $(B_{dedicate,i} + B_{share}) \cdot S$.

3.4 Partitioning mechanisms

For set-associative caches, various partitioning mechanisms can be used to actually allocate cache space to each thread. One way to partition the cache is to modify the LRU replacement policy which has the advantage of controlling the partition at cache block granularity, but LRU implementations can be expensive for high-associativity caches.

On the other hand, there are mechanisms that operate at coarse granularity. Page coloring [1] can restrict virtual address to physical address mapping, and as a result restrict cache sets that each thread uses. Column Caching [3] can partition the cache space by restricting cache columns (ways) that each thread can replace. However, it is relatively expensive to change the partition in these mechanisms, and the mechanisms support a limited number of partition blocks. In this section, we describe the modified LRU mechanism and column caching to be used in our experiments.

Modified LRU replacement In addition to LRU information, the replacement decision depends on the number of cache blocks that belongs to each thread (b_i). On a miss, the LRU cache block of the thread (i) that caused the miss is chosen to be replaced if its actually allocation (b_i) is larger than the desired one ($x_i \leq b_i$). Otherwise, the LRU cache block of another over-allocated thread is chosen. For set-associative caches, there may be no cache block of the desired thread in the set, so the LRU cache block of a randomly chosen thread is replaced.

Column caching Column caching is a mechanism that allows partitioning of a cache at column or "way" granularity. A standard cache considers all cache blocks in a set as candidates for replacement. As a result, a process' data can occupy any cache block. Column caching, on the other hand, restricts the replacement to a sub-set of cache blocks, which essentially partitions the cache.

Column caching specifies replacement candidacy using a bit vector in which a bit indicates if the corresponding column is a candidate for replacement. A LRU replacement unit is modified so that it replaces the LRU cache block from the candidates specified by a bit vector. Each partitionable unit has a bit vector. Since lookup is precisely the same as for a standard cache, column caching incurs no performance penalty during lookup.

4 Experimental results

This section presents the results of a trace-driven simulation system in order to understand the quantitative effects of our cache allocation scheme. The simulations concentrate on an 8-way set-associative L2 cache with 32-Byte blocks and vary the size of the cache over a range of 256 KB to 4 MB. Due to large space and long latency, our scheme is more likely to be useful for an L2 cache, and so that is the focus of our simulations. We note in passing, that we believe our approach will work on L1 caches as well.

Name	Thread	Description
Mix-1	`art`	Image Recognition/Neural Network
	`mcf`	Combinatorial Optimization
Mix-2	`vpr`	FPGA Circuit Placement and Routing
	`bzip2`	Compression
	`iu`	Image Understanding
Mix-3	`art1`	Image Recognition/Neural Network
	`art2`	
	`mcf1`	Combinatorial Optimization
	`mcf2`	

Table 1. The benchmark sets simulated. All but the Image Understanding benchmark are from SPEC CPU-2000.

Three different sets of benchmarks are simulated, see Table 1. The first set (Mix-1) has two threads, `art` and `mcf` both from SPEC CPU2000.

The second set (Mix-2) has three threads, `vpr`, `bzip2` and `iu`. Finally, the third set (Mix-3) has four threads, two copies of `art` and two copies of `mcf`, each with a different phase of the benchmark.

4.1 Hit-rate comparison

The simulations compare the overall hit-rate of a standard LRU replacement policy and the overall hit-rate of a cache managed by our partitioning algorithm. The partition is updated every two hundred thousand memory references ($T = 200000$), and the weighting factor is set as $\delta = 0.5$. These values have been arbitarily selected; more carefully selected values of T and δ are likely to give better results. The hit-rates are averaged over fifty million memory references and shown for various cache sizes (see Table 2).

Size (MB)	L1 %Hits	L2 %Hits	Part. L2 %Hits	Abs. %Imprv.	Rel. %Imprv.
`art` + `mcf`					
0.2	71.9	15.6	15.3	-0.2	-1.5
0.5		17.2	16.4	-0.8	-4.6
1		26.2	36.9	10.6	40.4
2		50.0	51.1	1.1	2.2
4		76.7	75.0	-1.6	-2.2
`vpr` + `bzip2` + `iu`					
0.2	95.4	22.9	22.1	-0.8	-3.6
0.5		27.5	28.2	0.6	2.5
1		33.5	35.8	2.3	7.0
2		59.6	66.3	6.6	11.2
4		81.3	81.5	0.2	0.2
`art1` + `mcf1` + `art2` + `mcf2`					
0.2	71.5	12.0	12.6	0.6	5.3
0.5		14.2	14.3	0.1	0.7
1		16.9	19.0	2.1	12.5
2		26.6	34.9	8.2	31.0
4		50.5	51.3	0.7	1.5

Table 2. Hit-rate Comparison between the standard LRU and the partitioned LRU.

The simulation results show that the partitioning can improve the L2 cache hit-rate significantly: for cache sizes between 1 MB to 2 MB, partitioning improved the hit-rate up to 40% relative to the hit-rate from the standard LRU replacement policy. For small caches, such as 256-KB and 512-KB caches, partitioning does not seem to help. We conjecture that the size of the total workloads is too large compared to the cache size. At the other extreme, partitioning cannot improve the cache performance if the cache is large enough to hold all the workloads. The range of cache sizes for which partitioning can improve performance depends on both the number of simultaneous threads and the characteristics of the threads. Considering that SMT systems usually support eight simultaneous threads, cache partitioning can improve the performance of caches in the range of up to tens of MB.

The results also demonstrate that the benchmark sets have large footprints. For all benchmark sets, the hit-rate improves by 10% to 20% as the cache size doubles. This implies that these benchmarks need a large cache, and therefore executing benchmarks simultaneously can degrade the memory system performance significantly.

4.2 Effect of partitioning on IPC

Although improving the hit-rate of the cache also improves the performance of the system, modern superscalar processors can hide memory latency by executing other instructions that are not dependent on missed memory references. Therefore, the effect of cache partitioning on the system performance, and in particular on IPC (Instructions Per Cycle), is evaluated based on entire system simulations.

The simulation results in this section are produced by SimpleScalar tool set [2]. SimpleScalar is a cycle-accurate processor simulator that supports out-of-order issue and execution. Our processor model for the simulations can fetch and commit 4 instructions at a time, and has 4 ALUs and 1 multiplier for integers and floating points respectively. To be consistent with the trace-driven simulations, 32-KB 8-way L1 caches with various sizes of 8-way L2 caches are simulated. L2 access latency is 6 cycles and main memory latency is 16 cycles.

Figure 2 (a) shows the IPC of two benchmarks (`art` and `mcf`) as a function of L2 cache size. Each benchmark is simulated separately and is allocated all system resources including all of the L2 cache. L1 caches are assumed to be 32-KB 8-way for all cases. For various L2 cache sizes, IPC is estimated as a function of the L2 hit-rate (Figure 2 (b)).

The figures illustrate two things. First, the IPC of `art` is very sensitive to the cache size. The IPC almost doubles if the L2 cache size is increased from 1 MB to 4 MB. Second, the IPCs of these two benchmarks are relatively low considering there are 10 functional units (5 for

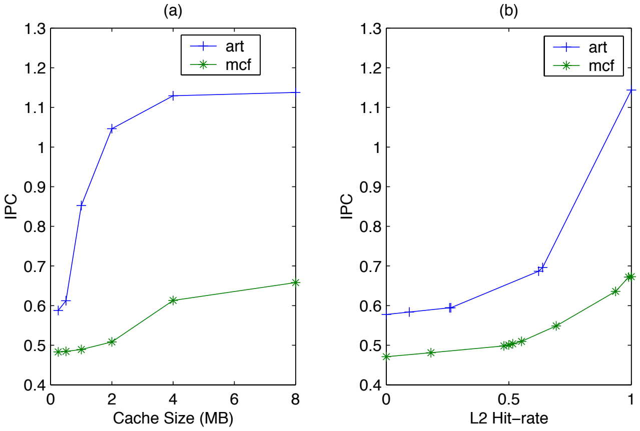


Fig. 2. IPC of `art` and `mcf` under 32-KB 8way L1 caches and various size 8-way L2 caches. (a) IPC as a function of cache size. (b) IPC as a function of L2 hit-rate.

integer, 5 for floating point instructions). Since the utilizations of the functional units are so low, executing these two benchmarks simultaneously will not cause many conflicts in functional resources.

When executing the threads simultaneously the IPC values are approximated from Figure 2 (b) and the hit-rates are estimated from the trace-driven simulations (of the previous subsection). For example, the hit-rates of `art` and `mcf` are 25.79% and 26.63%, respectively, if two threads execute simultaneously with a 32-KB 8-way L1 cache and a 1-MB 8-way L2 cache, from trace-driven simulation. From Figure 2 (b) the IPC of each thread for the given hit-rates can be estimated as 0.594 and 0.486. Assuming no resource conflicts, the IPC with SMT can be approximated as the sum, 1.08. This approximation bounds the maximum IPC that can be achieved by SMT.

Table 3 summarizes the approximated IPC for SMT with a L2 cache managed by the standard LRU replacement policy and one with a L2 cache managed by our partitioning algorithm. The absolute improvement in the table is the IPC of the partitioned case subtracted by the IPC of the standard LRU case. The relative improvement is the improvement relative to the IPC of the standard LRU, and is calculated by dividing the absolute improvement by the IPC of the standard LRU. The table shows that the partitioning algorithm improves IPC for all cache sizes up to 17%.

Cache Size (MB)	LRU			Partition			Abs. Improv. (%)	Rel. Improv. (%)
	Hit-rate(%)		IPC	Hit-rate(%)		IPC		
	`art`	`mcf`		`art`	`mcf`			
`art` + `mcf`								
0.25	8.8	20.4	1.064	8.0	20.5	1.065	0.001	0.09
0.5	10.3	22.2	1.067	14.5	17.8	1.070	0.003	0.28
1	25.7	26.6	1.080	61.6	19.5	1.167	0.087	8.06
2	63.7	40.3	1.189	76.8	33.1	1.347	0.158	13.29
`art1` + `mcf1` + `art2` + `mcf2`								
0.25	6.4/6.7	16.4/15.2	2.123	6.5/3.5	29.8/11.3	2.126	0.003	0.14
0.5	7.3/7.6	19.5/18.2	2.128	7.7/4.6	30.7/15.2	2.131	0.003	0.14
1	9.3/10.1	22.1/21.4	2.134	9.1/32.4	31.1/13.5	2.161	0.027	1.27
2	25.1/25.5	28.1/25.1	2.160	57.2/73.2	32.0/16.0	2.456	0.307	14.21
4	63.9/63.6	41.7/41.2	2.382	73.9/86.7	49.5/26.6	2.786	0.404	16.96

Table 3. IPC Comparison between the standard LRU and the partitioned LRU strategy for the case of executing `art` and `mcf` simultaneously.

The experiment results also show that SMT should manage caches carefully. In the case of four threads with a 2-MB cache, SMT can achieve the overall IPC of 2.160 from Table 3. However, if you only consider one thread (`art1`), its IPC is only 0.594 whereas it can achieve an IPC of 1.04 alone (Figure 2). The performance of a single thread is significantly degraded by sharing caches. Moreover, the performance degradation by cache interference will become even more severe as the latency to the main memory increases. This problem can be solved by smart partitioning of cache memory for some cases. If the cache is too small, we believe that the thread scheduling should be changed.

5 Conclusion

Low IPC can be attributed to two factors, data dependency and memory latency. SMT mitigates the first factor but not the second. We have discovered that SMT only exacerbates the problem when the executing threads require large caches. That is, when multiple executing threads interfere in the cache, even SMT cannot utilize all the functional units because not all required data is present in the memory.

We have studied one method to reduce cache interference among simultaneously executing threads. Our on-line cache partitioning algorithm estimates the miss-rate characteristics of each thread at run-time, and dynamically partitions the cache among the threads that are executing simultaneously. The algorithm estimates the marginal gains as a function of cache size and uses a search algorithm to find the partition that minimizes the total number of misses.

The hardware overhead for the modifications proposed in this paper are minimal. A small number of additional counters is required. The counters are updated on cache hits, however, they are not on the critical path and so a small buffer can absorb any burstiness. To actually partition the cache, we can modify the LRU replacement hardware in a simple way to take the values of the counters into account. Or, we can use column caching which requires a small number of additional bits in the TLB entries, and a small amount of off-critical-path circuitry that is invoked only during a cache miss.

The partitioning algorithm has been implemented in a trace-driven cache simulator. The simulation results show that partitioning can improve the cache performance noticeably over the standard LRU replacement policy for a certain range of cache size for given threads. Using a full-system simulator, the effect of partitioning on the instructions per cycle (IPC) has also been studied. The preliminary results show that we can also expect IPC improvement using the partitioning algorithm. While we have not used a full SMT simulator to generate IPC numbers, the large improvements obtained in hit rates lead us to believe that significant IPC improvements will be obtained using a full SMT simulator, or on real hardware.

The simulation results have shown that our partitioning algorithm can solve the problem of thread interference in caches for a range of cache sizes. However, partitioning alone cannot improve the performance if caches are too small for the workloads. Therefore, threads that execute simultaneously should be selected carefully considering their memory reference behavior. Cache-aware job scheduling is a subject of our ongoing work.

Even without SMT, one can view an application as multiple threads executing simultaneously where each thread has memory references to a particular data structure. Therefore, the result of this investigation can also be exploited by compilers for a processor with multiple functional units and some cache partitioning control.

6 Acknowledgements

Funding for this work is provided in part by the Defense Advanced Research Projects Agency under the Air Force Research Lab contract F30602-99-2-0511, titled "Malleable Caches for Data-Intensive Computing". Thanks also to Enoch Peserico, Derek Chiou, David Chen and Vinson Lee for their comments.

References

1. Brian K. Bershad, Bradley J. Chen, Denis Lee, and Theodore H. Romer. Avoiding conflict misses dynamically in large direct-mapped caches. In *ASPLOS VI*, 1994.
2. Doug Burger and Todd M. Austin. The simplescalar tool set, version 2.0. Technical report, University of Wisconsin-Madison Computer Science Department, 1997.
3. Derek T. Chiou. *Extending the Reach of Microprocessors: Column and Curious Caching.* PhD thesis, Massachusetts Institute of Technology, 1999.
4. C. K. Chow. Determining the optimum capacity of a cache memory. IBM Tech. Disclosure Bull., 1975.
5. Compaq. Compaq alphastation family.
6. Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, and Dean M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17(5), 1997.
7. B. Fox. Discrete optimization via marginal analysis. *Management Science*, 13, 1966.
8. Catherine Freeburn. The hewlett packard PA-RISC 8500 processor. Technical report, Hewlett Packard Laboratories, October 1998.
9. John L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, July 2000.
10. Jack L. Lo, Joel S. Emer, Henry M. Levy, Rebecca L. Stamm, Dean M. Tullsen, and S. J. Eggers. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, 15, 1997.
11. MIPS Technologies, Inc. *MIPS R10000 Microprocessor User's Manual*, 1996.
12. Harold S. Stone, John Turek, and Joel L. Wolf. Optimal partitioning of cache memory. *IEEE Transactions on Computers*, 41(9), September 1992.
13. G. Edward Suh, S. Devadas, and L. Rudolph. Analytical Cache Models with Application to Cache Partitioning. In *the* 15th *international conference on Supercomputing*, 2001.
14. Dominique Thiébaut, Harold S. Stone, and Joel L. Wolf. Improving disk cache hit-ratios through cache partitioning. *IEEE Transactions on Computers*, 41(6), June 1992.
15. Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture*, 1995.

7th International Workshop on Job
Scheduling Strategies for Parallel
Processing, in Lecture Notes in
Computer Science, Vol. 2221,
Springer-Verlag, pp. 116–132, 2001.

Effects of Memory Performance on Parallel Job Scheduling

G. Edward Suh, Larry Rudolph and Srinivas Devadas

MIT Laboratory for Computer Science
{`suh, rudolph, devadas`}`@mit.edu`

Abstract. We develop a new metric for job scheduling that includes the effects of memory contention amongst simultaneously-executing jobs that share a given level of memory. Rather than assuming each job or process has a fixed, static memory requirement, we consider a general scenario wherein a process' performance monotonically increases as a function of allocated memory, as defined by a miss-rate versus memory size curve. Given a schedule of jobs in a shared-memory multiprocessor (SMP), and an isolated miss-rate versus memory size curve for each job, we use an analytical memory model to estimate the overall memory miss-rate for the schedule. This, in turn, can be used to estimate overall performance. We develop a heuristic algorithm to find a good schedule of jobs on a SMP that minimizes memory contention, thereby improving memory and overall performance.

1 Introduction

High performance computing is more than just raw FLOPS; it is also about managing the memory among parallel threads so as to keep the operands flowing into the arithmetic units. Hence, some high performance job schedulers are beginning to consider the memory requirements of a job in addition to the traditional CPU requirements. But memory is spread across a hierarchy, it is difficult to know the real requirements of each job, and underallocation of space to one job can adversely affect the performance of other jobs. Allocating a fixed amount of space to a job regardless of the needs of the other concurrently executing jobs can result in suboptimal performance. We argue that a scheduler must compare the marginal utility or marginal gain accrued by a job to the gains accrued by other jobs, when giving more memory to a job.

Shared-memory multiprocessors (SMPs) [2, 8, 9], have become a basic building block for modern high performance computer systems, and in the near future, other layers of the memory hierarchy will be shared as well, with multiple processors (MPC) on a chip [3] and simultaneous multithreading (SMT) systems [13, 10, 4]. So, in nearly all high performance systems, there will be either threads, processes, or jobs that execute simultaneously and share parts of the memory system. But how many jobs should execute simultaneously? There is no magic number, rather it depends on the individual memory requirements of the jobs. Sometimes, it is even beneficial to let some processors remain idle so as to improve the overall performance.

Although most research on job scheduling for high performance parallel processing is concerned only with the allocation of processors in order to maximize processor utilization [5, 6], scheduling with memory considerations is not new. Parsons [11] studied bounds on the achievable system throughput considering memory demand of parallel jobs. Batat [1] improved gang scheduling by imposing admission control based on the memory requirement of a new job and the available memory of a system. The modified gang scheduler estimates the memory requirement for each job, and assigns a job into a time slice only if the memory is large enough for all jobs in the time slice. Although these works have pointed out the importance of considering memory in job scheduling problems, they did not provide a way of scheduling jobs to optimize the memory performance.

Rather than assuming each job or process has a fixed, static memory requirement, this paper considers a general scenario wherein a process' performance monotonically increases as a function of allocated memory. The characteristics of each process' memory usage are given by the miss-rate as a function of memory size when the process is executed in isolation (which can be easily obtained either in an on-line or off-line manner). With this information, an analytical memory model for time-shared systems [12] can be used to estimate the memory miss-rate for each job and the processor idle time for a given schedule. Therefore, our approach provides a good memory performance metric for job scheduling problems.

The new approach based on the miss-rate curves and the analytical model can be used to evaluate a schedule including the effects of memory performance. If multiple processors share the same memory, our method can effectively schedule a given set of processes to minimize memory contention. Finally, the length of time slices can be determined for time-shared systems so as to minimize pollution effects.

The paper is organized as follows. In Section 2, we present a case study of scheduling SPEC CPU2000 benchmarks, which demonstrates the importance and challenges of job scheduling with memory considerations. Section 3 motivates isolated miss-rate curves, and describes how an analytical memory model evaluates the effect of a given schedule on the memory performance. Section 4 discusses new challenges that memory considerations impose on parallel job scheduling, and suggests possible solutions using the miss-rate curves and the model. Finally, Section 5 concludes the paper.

2 Case Study: SPEC CPU2000

This section discusses the results of trace-driven simulations that estimate the miss-rate of main memory when six jobs execute on a shared-memory multiprocessor system with three processors. The results demonstrate the importance of memory-aware scheduling and the problem of naive approaches based on footprint sizes.

Name	Description	Footprint (MB)
`bzip2`	Compression	6.2
`gcc`	C Programming Language Compiler	22.3
`gzip`	Compression	76.2
`mcf`	Image Combinatorial Optimization	9.9
`vortex`	Object-oriented Database	83.0
`vpr`	FPGA Circuit Placement and Routing	1.6

Table 1. The descriptions and Footprints of benchmarks used for the simulations. All benchmarks are from SPEC CPU2000 [7] benchmark suite.

Six jobs, which have various footprint sizes, are selected from SPEC CPU2000 benchmark suite [7] (See Table 1). Here, footprint size represents the memory size that a benchmark needs to achieve the minimum possible miss-rate. Benchmarks in the SPEC CPU2000 suite are not parallel jobs, however, the insights obtained from the experiments are also valid for parallel processing of multi-threaded jobs since all threads (or processes) from a job can be considered as one large process from the main memory standpoint.

Concurrent execution of six jobs by three processors requires time-sharing. We assume that there are two time slices long enough to render context switching overhead negligible. In the first time slice, three out of the six jobs execute sharing the main memory and in the second time slice the three remaining jobs execute. Processors are assumed to have 4-way 16-KB L1 instruction and data caches and a 8-way 256-KB L2 cache, and 4-KB pages are assumed for the main memory.

Memory Size (MB)		Average of All Cases	Worst Case	Best Case
8	Miss-Rate(%)	1.379	2.506	1.019
	Schedule		(ADE,BCF)	(ACD,BEF)
16	Miss-Rate(%)	0.471	0.701	0.333
	Schedule		(ADE,BCF)	(ADF,BCE)
32	Miss-Rate(%)	0.187	0.245	0.148
	Schedule		(ADE,BCF)	(ACD,BEF)
64	Miss-Rate(%)	0.072	0.085	0.063
	Schedule		(ABF,CDE)	(ACD,BEF)
128	Miss-Rate(%)	0.037	0.052	0.029
	Schedule		(ABF,CDE)	(ACD,BEF)
256	Miss-Rate(%)	0.030	0.032	0.029
	Schedule		(ABF,CDE)	(ACD,BEF)

Table 2. The miss-rates for various job schedules. A schedule is represented by two sets of letters. Each set represents a time slice, and each letter represents a job: A-`bzip2`, B-`gcc`, C-`gzip`, D-`mcf`, E-`vortex`, F-`vpr`.

All possible schedules are simulated for various memory sizes. We compare the average miss-rate of all possible schedules with the miss-rates of the worst and the best schedule. The miss-rate only considers accesses to main memory, not accesses that hit on either L1 or L2 caches. The simulation results are summarized in Table 2 and Figure 1. In the table, a corresponding schedule for each case is also shown. In the 128-MB and 256-MB cases, many schedules result in the same miss-rate. A schedule is represented by two sets of letters. Each set represents a time slice, and each letter represents a job: A-`bzip2`, B-`gcc`, C-`gzip`, D-`mcf`, E-`vortex`, F-`vpr`. In the figure, the miss-rates are normalized to the average miss-rate.

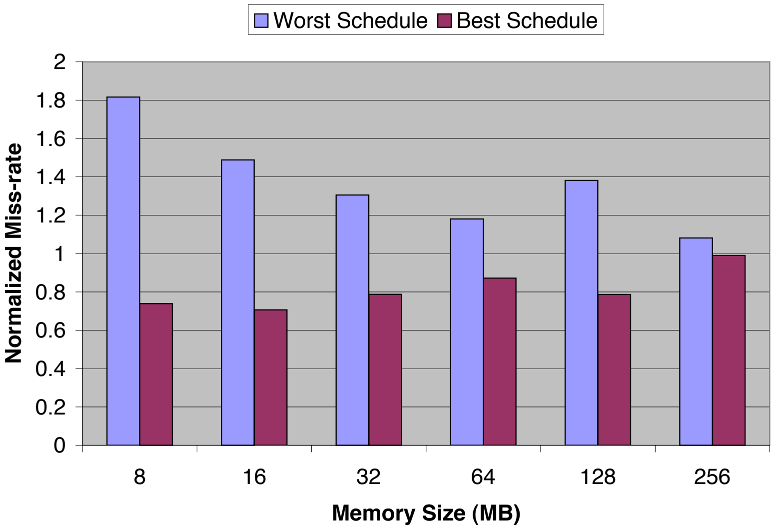


Fig. 1. The comparison of miss-rates for various schedules: the worst case, the best case, and the average of all possible schedules. The miss-rates are normalized to the average miss-rate of all possible schedules for each memory size. Notice that even when the memory is large enough to hold all the footprints of the executing jobs, the set of jobs that execute together has an effect on the miss-rate.

The results demonstrate that job scheduling can have significant effects on the memory performance, and thus the overall system performance. For 16-MB memory, the best case miss-rate is about 30% better than the average case, and about 53% better than the worst case. Given a very long page fault penalty, performance can be significantly improved due to this large reduction in miss-rate. As the memory size increases, scheduling becomes less important since the entire workload fits into the memory. However, the smart schedule can still improve the memory performance significantly even for the 128-MB case (over 20% better than the average case, and 40% better than the worst case).

Memory traces used in this experiment have footprints smaller than 100 MB. As a result, scheduling of simultaneously executing processes is relevant to the main memory performance only for the memory up to 256 MB. However, many parallel applications have very large footprints often larger than main memory. For these applications, the memory size where scheduling matters should scale up.

An intuitive way of scheduling with memory considerations is to use footprint sizes. Since the footprint size of each job indicates its memory space needs, one can try to balance the total footprint size for each time slice. It also seems to be reasonable to be conservative and keep the total footprint size smaller than available physical memory. The experimental results show that these naive approaches do not work.

Balancing the total footprint size for each time slice may not work for memory smaller than the entire footprint. The footprint size of each benchmark only provides the memory size that the benchmark needs to achieve the best performance, however, it does not say anything about having less memory space. For example, in our experiments, executing `gcc`, `gzip` and `vpr` together and the others in the next time slice seems to be reasonable since it balances the total footprint size for each time slice. However, this schedule is actually the worst schedule for memory smaller than 128-MB, and results in a miss-rate that is over 50% worse than the optimal schedule.

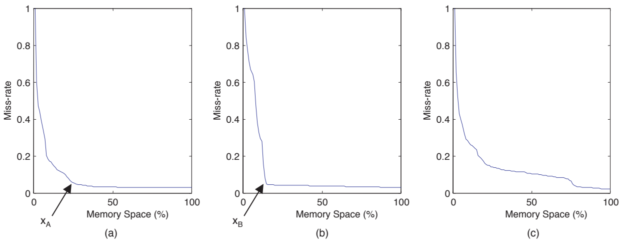
If the replacement policy is not ideal, even being conservative and having larger physical memory than the total footprint may not be enough to guarantee the best memory performance. Smart scheduling can still improve the miss-rate by about 10% over the worst case even for 256-MB memory that is larger than the total footprint size of any three jobs from Table 1. This happens because the LRU replacement policy does not allocate the memory properly. (For a certain job, the LRU policy may allocate memory larger than the footprint of the job).

3 New approach based on miss-rate curves

The previous section pointed out that the conventional scheduling approaches based on static footprints are very limited. This section proposes a new approach based on the *isolated miss-rate curve*, $m_i(x)$. After defining the isolated miss-rate curve, an analytical model is developed that incorporates the effect of time-sharing and memory contention based on the miss-rate curves. Using these curves and the model, we show how to evaluate a given schedule.

3.1 Miss-rate curves

The *isolated miss-rate curve* for process i, namely $m_i(x)$, is defined as the miss-rate when process i is isolated without other competing processes using the memory of size x. Effectively, this miss-rate curve represents the miss-rate when a process occupies only a part of the entire memory.



(a) (b) (c)

Fig. 2. (a) Miss-rate curve for process P_A (`gcc`). (b) Miss-rate curve for process P_B (`swim`). (c) Miss-rate curve for process P_C (`bzip2`). Clearly, process P_A's miss-rate does not reduce very much after the point marked x_A. Similarly, for process P_B after the point marked x_B. If $x_A + x_B$ is less than the total memory size available, then it is likely that processes P_A and P_B can both be run together, achieving good performance, especially if they are restricted to occupy an appropriate portion of the cache. On the other hand, process P_C has a different type of miss-rate curve, and will likely not run well with either P_A or P_B.

The advantage of having a miss-rate curve rather than static footprints is clear for the problem of scheduling processes for shared-memory systems. Consider the case of scheduling three processes, whose miss-rate curves are shown in Figure 2, on a shared-memory system with two processors. *Which two processes should run together?* This question cannot be answered based on the static footprints since the memory is smaller than the individual footprints. However, from the miss-rate curves, it is clear that running both P_A and P_B simultaneously and P_C separately will result in a lower miss-rate than running P_A or P_B with P_C.

3.2 Estimating the miss-rate curves

The miss-rate curves can be obtained either on-line or off-line. Here, an on-line method to estimate a miss-rate curve $m_i(x)$ is described. We use the LRU information of each page and count the number of hits in the kth most recently used page for each process ($counter_i[k]$). For example, $counter_i[1]$ is the number of hits in the most recently used page of process i, and $counter_i[2]$ is the number of hits in the second most recently used page. If we count hits for one time slice, $m_j(x)$ and $counter_j[k]$ have the following relation.

$$counter_i[k] = (m_i(k-1) - m_i(k)) \cdot r_i. \tag{1}$$

where r_i is the number of memory accesses for process i over one time slice. Since $m_j(0) = 1$, we can calculate the miss-rate curve recursively.

3.3 Modeling memory contention

Although isolated miss-rate curves provide much more information than static footprints, the miss-rate curves alone are still not enough to predict the effects of memory contention under a non-ideal replacement policy or under the effects of time-sharing. This subsection explains how a previously developed analytical model can be extended to accurately estimate the overall miss-rate incorporating both space-sharing effects and time-sharing effects. First, the original uniprocessor model of [12] is briefly summarized. Then, we discuss how this original model can be applied to parallel jobs on shared-memory multiprocessor systems.

Uniprocessor model The cache model from [12] estimates the overall miss-rate for a fully-associative cache when multiple processes time-share the same cache (memory) on a uniprocessor system. There are three inputs to the model: (1) the memory size (C) in terms of the number of memory blocks (pages), (2) job sequences with the length of each process' time slice (T_i) in terms of the number of memory references, and (3) the miss-rate curve for each process ($m_i(x)$). The model assumes that the least recently used (LRU) replacement policy is used, and that there are no shared data structures among processes.

Let us consider a case when N processes execute with a given schedule (sequences of processes) and fixed time quanta for each process (T_i). First, the number of misses for each process' time quantum is estimated. Then, the overall miss-rate is obtained by combining the number of misses for each process.

Define the footprint of process i, $x_i(t)$, as the amount of process i's data in the memory at time t where time t is 0 at the beginning of the process' time quantum. Then, $x_i(t)$ is approximated by the following recursive equation, once $x_i(0)$ is known [1];

$$x_i(t+1) = MIN[x_i(t) + m_i(x_i(t)), C], \tag{2}$$

where C is the size of memory in terms of the number of blocks.

The miss-rate curve, $m_i(x)$, can be considered as the probability to miss when x valid blocks are in the memory. Therefore, the number of misses that process i experiences over one time quantum is estimated from the footprint of the process $x_i(t)$ as follows;

$$\text{miss}_i = \int_0^{T_i} m_i(x_i(t))dt. \tag{3}$$

Once the number of misses for each process is estimated, the overall miss-rate is straightforwardly calculated from those numbers.

$$\text{miss-rate}_{\text{overall}} = \frac{\sum_{i=1}^{N} \text{miss}_i}{\sum_{i=1}^{N} T_i} \tag{4}$$

Extension to multiprocessor cases The original model assumes only one process executes at a time. Here, we describe how the original model can be applied to multiprocessor systems where multiple processes can execute simultaneously sharing the memory. Although the model can be applied to more general cases, we consider the situation where all processors context switch at the same time; more complicated cases can be modeled in a similar manner.

No matter how many processes are executing simultaneously sharing the memory, all processes in a time slice can be seen as one big process from the standpoint of memory. Therefore, we take a two-step approach to model shared-memory multiprocessor cases. First, define a conceptual process for each time slice that includes memory accesses from all processes in the time slice, which we call a *combined process.* Then, the miss-rate for the combined process of each time slice is estimated using the original model. Finally, the uniprocessor model is used again to incorporate the effects of time-sharing assuming only the combined process executes for each time slice.

---

[1] The estimation of $x_i(0)$ and more accurate $x_i(t)$ can be found in our previous work [12].

What should be the miss-rate curve for the combined process of a time slice? Since the original model for time-sharing needs *isolated* miss-rate curves, the miss-rate curve of each time-slice s is defined as the overall miss-rate of all processes in time slice s when they execute together without context switching on the memory of size x. We call this miss-rate curve for a time slice as a combined miss-rate curve $m_{combined,s}(x)$. Next we explain how to obtain the combined miss-rate curves.

The simultaneously executing processes within a time slice can be modeled as time-shared processes with very short time quanta. Therefore, the original model is used to obtain the combined miss-rate curves by assuming the time quantum is $ref_{s,p}/\sum_{i=1}^{P} ref_{s,i}$ for processor p in time-slice s. $ref_{s,p}$ is the number of memory accesses that processor p makes over time slice s. The following paragraphs summarize this derivation of the combined miss-rate curves. Here, we use $m_{s,p}$ to represent the isolated miss-rate curve for the process that executes on processor p in time slice s.

Let $x_{s,p}(k_{s,p})$ be the number of memory blocks that processor p brings into memory after $k_{s,p}$ memory references in time slice s. The following equation estimates the value of $x_{s,p}(k_{s,p})$:

$$k_{s,p} = \int_{0}^{x_{s,p}(k_{s,p})} \frac{1}{m_{s,p}(x')} dx'. \tag{5}$$

Considering all P processors, the system reaches the steady-state after K_s memory references where K_s satisfies the following equation.

$$\sum_{p=1}^{P} x_{s,p}(\alpha(s,p) \cdot K_s) = x. \tag{6}$$

In the above equation, x is the number of memory blocks, and $\alpha(s,p)$ is the length of a time slice for processor p, which is equal to $ref_{s,p}/\sum_{i=1}^{P} ref_{s,i}$. In steady-state, the combined miss-rate curve is given by

$$m_{combined,s}(x) = \sum_{p=1}^{P} \alpha(s,p) \cdot m_{s,p}(x_p(\alpha(s,p) \cdot K_s)). \tag{7}$$

Now we have the combined miss-rate curve for each time-slice. The overall miss-rate is estimated by using the original model assuming that only one process executes for a time slice whose miss-rate curve is $m_{combined,s}(x)$.

Dealing with shared memory space The model described so far assumes that there is no shared memory space among processes. However, processes from the same parallel job often communicate through shared memory space. The analytical model can be modified to be used in the case of parallel jobs synchronizing through shared memory space, as described below.

The accesses to shared memory space can be excluded from the miss-rate curve of each process, and considered as a separate process from the viewpoint of memory. For example, if P processes are simultaneously executing and share some memory space, the multiprocessor model in the previous subsection can be used considering $P + 1$ conceptual processes. The first P miss-rate curves are from the accesses of the original P processes excluding the accesses to the shared memory space, and the $(P + 1)^{th}$ miss-rate curve is from the accesses to the shared memory space. Since the $P + 1$ conceptual processes do not have shared memory space, the original model can be applied.

3.4 Evaluating a schedule

A poor schedule has lots of idle processors, and a schedule can be better evaluated in terms of a processor idle time rather than a miss-rate. A processor is idle for a time slice if no job is assigned to it for that time slice or it is idle if it is waiting for the data to be brought into the memory due to a "miss" or page fault. Although modern superscalar processors can tolerate some cache misses, it is reasonable to assume that a processor stalls and therefore idles on every page fault.

Let the total processor idle time for a schedule be as follows:

$$\begin{aligned}\text{Idle}(\%) &= \{\sum_{s=1}^{S}\sum_{p=1}^{N(s)} miss(p,s)\cdot l \\ &\quad + \sum_{s=1}^{S}(P - N(s)\cdot T(s)\}/\{\sum_{s=1}^{S} T(s)\} \\ &= \{(\text{total misses})\cdot l \\ &\quad + \sum_{s=1}^{S}(P - N(s))\cdot T(s)\}/\{\sum_{s=1}^{S} T(s)\}\end{aligned} \tag{8}$$

where $miss(p, s)$ is the number of misses on processor p for time slice s, l is the memory latency, $T(s)$ is the length of time slice s, and $N(s)$ is the number of processes scheduled in time slice s.

In Equation 8, the first term represents the processor idle time due to page faults and the second term represents the idle time due to processors with no job scheduled on. Since the number of idle processors is given with a schedule, we can evaluate a given schedule once we know the total number of misses, which can be estimated from the model in the previous subsection.

4 The effects of memory performance on scheduling

This section discusses new considerations that memory performance imposes on parallel job scheduling and their solutions based on the miss-rate curves and the analytical model. First, we discuss scheduling problems to optimize memory performance for the space-shared systems. Then, scheduling considerations for time-sharing the memory are studied.

4.1 Processes to space-share memory

In shared-memory multiprocessor systems, processes in the same time slice space-share the memory since they access the memory simultaneously. In this case, the amount of memory space allocated to each process is determined by the other processes that are scheduled in the same time slice. Therefore, the performance (execution time) of each process can be significantly affected by which processes are scheduled to space-share the memory (see Section 2). The main consideration of memory-aware schedulers in space-shared systems is to group jobs in a time slice properly so as to minimize the performance degradation caused by the memory contention.

A schedule can be evaluated using the isolated miss-rate curves and the analytical model. Effectively, the model provides a new cost function of memory performance, and any scheduler can be modified to incorporate memory considerations by adding this new cost function from the model. As an example, here we show how a simple gang scheduler can be modified to consider the memory performance. The modification of more complicated schedulers is left for future studies.

Consider the problem of scheduling J jobs on a P_{tot} processor system, which consists of SMPs with P_{node} processors. Gang scheduling is assumed, i.e., all processes from one job are scheduled in the same time slice, and context switch at the end of the time slice. All P_{tot} processors are assumed to context switch at the same time. A processor does not context switch even on a page fault, but only when the time slice expires. The problem is to determine the number of time slices S to schedule all jobs, and assign each job to a time slice so that the processor idle time is minimized. Also, each process should be mapped to a SMP node considering memory contention.

The most obvious way of scheduling with memory consideration is to use the analytical model detailed in Section 3. If the isolated miss-rate curves are obtained either on-line or off-line, the model can easily compare different schedules. The problem is to search for the optimal schedule with the given evaluation method. For a small number of jobs, an exhaustive search can be performed to find the best schedule. As the number of jobs increases, however, the number of possible schedules increases exponentially, which makes exhaustive search impractical. Unfortunately, there appears to be no polynomial-time algorithm that guarantees an optimal solution.

A number of search algorithms can be developed to find a sub-optimal schedule in polynomial time using the analytical model directly. Alternately, we can just utilize the miss-rate curves and incorporate better memory considerations into existing schedulers. Although the analytical model is essential to accurately compare different schedules and to find the best schedule, we found that a heuristic algorithm based only on the miss-rate curves is often good enough for optimizing memory performance for space-sharing cases. The following subsection presents the heuristic search algorithm.

A heuristic algorithm For most applications, the miss rate curve as a function of memory size has one prominent knee (See Figure 2). That is, the miss rate quickly drops and then levels off. As a rough approximation, this knee is considered as a *relative footprint* of the process. Then, processes are scheduled to balance the total size of relative footprints for each node. Although this algorithm cannot consider the complicated effects of memory contention, it is much cheaper than computing the model and often results in a reasonable schedule.

The algorithm works in three steps; First, the *relative footprints* are determined considering the number of processes and the size of memory. At the same time, we decide the number of time slices S. Then, jobs are assigned to a time slice to balance the total relative footprints for each time slice. Finally, processes are assigned to a node to balance the relative footprints for each node.

In the explanation of the algorithm, we make use of the following notations:

- P_{tot}: the total number of processors in the entire system.
- P_{node}: the number of processors in a node.
- J: the total number of jobs to be scheduled.
- $Q(j)$: the number of processors that job j requires.
- m_j: the miss-rate curve for job j.
- r_j: the number of memory references of job j for one time slice.
- S: the number of time slices to schedule all jobs.

The relative footprint for job j, $fp(j)$ is defined as the number of memory blocks allocated to the job when the memory with $C \cdot S \cdot P/P_{node}$ blocks is partitioned among all jobs so that the marginal gain for all jobs is the same. Effectively, the relative footprint of a job represents the optimal amount of memory space for that job when all jobs execute simultaneously sharing the entire memory resource over S time slices.

To compute the relative footprints, the number of time slices S should also be decided. First, make an initial, optimistic guess; $S = \lceil \sum_{j=1}^{J} Q(j)/P \rceil$.

Then, compute the relative footprints for that S and approximate the processor idle time using Equation 8 assuming that each job experiences $m_j(fp(j)) \cdot r_j$ misses over a time slice. Finally, increase the number of time slices and try again until the resultant idle time increases. For a given S, the following greedy algorithm determines the relative footprints.

1. Compute the marginal gain $g_j(x) = (m_j(x-1) - m_j(x)) \cdot r_j$. This function represents the number of additional hits for the job j, when the allocated memory blocks increases from $x-1$ to x.
2. Initialize $fp(1) = fp(2) = ... = fp(J) = 0$.
3. Assign a memory block to the job that has the maximum marginal gain. For each job, compare the marginal gain $g_j(fp(j)+1)$ and find the job that has the maximum marginal gain j_{max}. Increase the allocation for the job $fp_{j_{max}}$ by one.
4. Repeat step 3 until all memory blocks are assigned.

Once the relative footprints are computed, assigning jobs to time slices is straightforward. In a greedy manner, the unscheduled job with the largest relative footprint is assigned to a time slice with the smallest total footprint at the time. After assigning jobs to time slices, we assume that each process from job j has the relative footprint of $fp(j)/Q(j)$. Then, assign processes to nodes in the same manner.

Notice that the analytic model is not used by this algorithm. However, the model is needed to validate the heuristic. For jobs that have significantly different miss-rate curves, new heuristics are needed and the model will be required to validate those as well.

Fig. 3. The performance of the model-based scheduling algorithm and the heuristic scheduling algorithm. The miss-rates are normalized to the average miss-rate of all possible schedules for each memory size.

Experimental validation The model-based algorithm and the heuristic algorithm are applied to solve a scheduling problem in Section 2. The problem is to schedule six SPEC CPU2000 benchmarks using three processors and two time slices. Figure 3 compares the miss-rates of the model-based algorithm and the heuristic algorithm with miss-rates of the best schedule and the worst schedule, which are already shown in Section 2. The best schedule and the worst schedule are found by simulating all possible schedules and comparing their miss-rates. For the model-based algorithm, the average isolated miss-rate curves over the entire execution are obtained by trace-driven simulations. Then, the schedule is found by an exhaustive search based on the analytical model. The heuristic algorithm uses the same average isolated miss-rate curves, but decides the schedule using the algorithm in the previous subsection. Once the schedules are decided by either the model-based algorithm or the heuristic algorithm, the actual miss-rates for those schedules are obtained by trace-driven simulations.

The results demonstrate that our scheduling algorithms can effectively find a good schedule. In fact, the model-based algorithm found the best schedule except for the 16-MB and 64-MB cases. Even for these cases, the model-based schedule found by the algorithm shows a miss-rate very close to the best case.

The heuristic algorithm also results in good schedules in most cases with significantly less computation than the model-based algorithm. However, the heuristic algorithm shows worse performance than the model-based algorithm because it cannot accurately estimate the effects of the LRU replacement policy.

4.2 The length of time slices

When available processors are not enough to execute all jobs in parallel, processors should be time-shared amongst jobs. In conventional batch processing, each job runs to completion before giving up the processor(s). However, this approach may block short jobs from executing and significantly degrade the response time. Batch processing may also cause significant processor fragmentation. Therefore, many modern job scheduling methods such as gang scheduling use time slices shorter than the entire execution time to share processors.

Unfortunately, shorter time slices often degrade the memory performance since each job should reload the evicted data every time it restarts the execution. To amortize this context switching cost and achieve reasonable performance in time-shared systems, schedulers should ensure that time slices are long enough to reload data and reuse them. Time slices should be long to reduce the context switch overhead, but short to improve response time and processor fragmentation.

The proper length of time slices still remains as a question. Conventionally, the length of time slices are determined empirically. However, the proper length of time slices depends on the characteristics of concurrent jobs and changes as jobs and/or memory configuration vary. For example, a certain length of time slice may be long enough for jobs with a small working set, but not long enough for larger jobs. Since the proposed analytical model can predict the miss-rate for a given length of time slices, it can be used to determine the proper length once another cost function such as response time or fragmentation is given.

Figure 4 shows the overall miss-rate as a function of the length of time slices when three SPEC CPU2000 benchmarks, `gzip`, `vortex`, and `vpr`, are concurrently executing with a round-robin schedule. The solid line represents the simulation results, and the dashed line represents the miss-rates estimated by the model. The figure shows a very interesting fact that a certain range of time slices can be very problematic

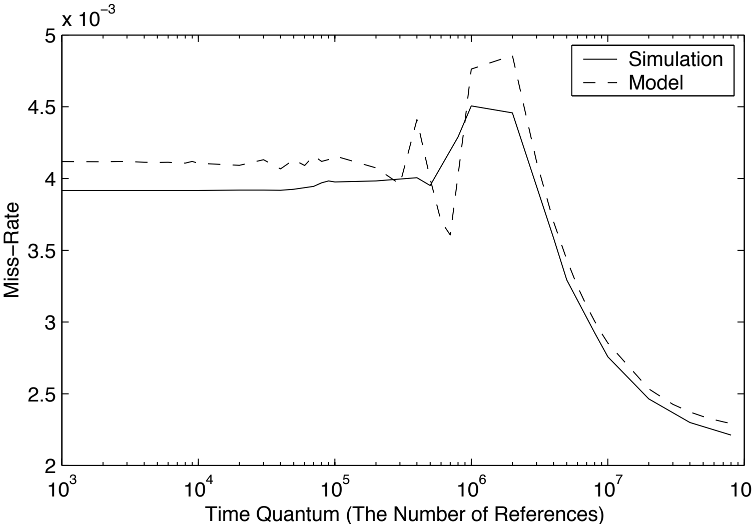


Fig. 4. The overall miss-rate when three processes (`gzip`, `vortex`, `vpr`) are sharing the memory (64 MB). The solid line represents the simulation results, and the dashed line represents the miss-rates estimated by the analytical model. The length of a time quantum is assumed to be the same for all three processes.

for memory performance. Conventional wisdom assumes that the miss-rate will monotonically decrease as the length of time slices increase. However, the miss-rate may increase for some cases since more data of processes that will run next are evicted as the length of time slices increase. The problem occurs when a time slice is long enough to pollute the memory but not long enough to compensate for the misses caused by context switches.

It is clear that time slices should always be long enough to avoid the problematic bump. Fortunately, the analytical model can estimate the miss-rate very close to the simulation results. Therefore, we can easily evaluate time slices and choose ones that are long enough.

5 Conclusion

Modern multiprocessor systems commonly share the same physical memory at some levels of memory hierarchy. Sharing memory provides fast synchronization and communication amongst processors. Sharing memory also enables flexible management of the memory. However, it is clear that sharing memory can exacerbate the memory latency problem due to conflicts amongst processors. Currently, users of high performance computing systems prefer to “throw out the baby with the bathwater” and fore-go virtual memory and sharing of memory resources. We believe such extreme measures are not needed. Memory-aware scheduling can solve the problem.

This paper has studied the effects of the memory contention amongst processors that share the same memory on job scheduling. The case study of SPEC CPU2000 benchmarks has shown that sharing the memory can significantly degrade the performance unless the memory is large enough to hold the entire working set of all processes. Further, memory performance is heavily dependent on job scheduling. We have shown that the best schedule that minimizes memory contention cannot be found based on conventional footprints.

Miss-rate curves and an analytical model has been proposed as a new method to incorporate the effects of memory contention in job scheduling. The analytical model accurately estimates the overall miss-rate including both space-sharing effects and time-sharing effects from the miss-rate curves. Therefore, they provide a new cost function of memory performance, and any scheduler can be modified to incorporate memory considerations by adding this new cost function.

As an example, a simple gang scheduler is modified to optimize the memory performance. Applying theory to practice is not straightforward: First, some mechanism is needed to estimate the miss-rate characteristics at run-time since it is unreasonable to expect the user to provide an accurate function. Second, a heuristic algorithm is required to find a solution in polynomial time. Simulation results have validated our approach that can effectively find a good schedule that results in low miss-rates. Both a model-based algorithm and a heuristic algorithm were simulated and evaluated. Although the exhaustive search algorithm based on the model showed slightly better performance than the heuristic algorithm, the difference is minimal. Therefore, we believe that anything more than an inexpensive heuristic is overkill.

The paper is mainly focused on optimizing the performance for simultaneously executing processes. However, the approach based on the miss-rate curves and the analytical model is also applicable to scheduling problems related to time-sharing. In time-shared systems, there is a tradeoff in the length of time slices. Our model provides the metric of memory performance for this tradeoff. Especially, it is shown that a certain range of time slices can be very harmful for memory performance and this range can be avoided using the model.

The development of more realistic memory-aware schedulers is left for future studies. Practical schedulers have many considerations other than memory performance, thus it is more complicated to incorporate memory considerations into these schedulers as compared to a simple gang scheduler. However, we believe that the miss-rate curves and the analytical model provide a good metric for memory performance and existing schedulers can be modified to optimize the memory performance utilizing the given degrees of freedom.

6 Acknowledgements

Funding for this work is provided in part by the Defense Advanced Research Projects Agency under the Air Force Research Lab contract F30602-99-2-0511, titled “Malleable Caches for Data-Intensive Computing”. Funding was also provided in part by a grant from the NTT Corporation. Thanks also to David Chen, Derek Chiou, Prahbat Jain, Josh Jacobs, Vinson Lee, Peter Portante, and Enoch Peserico.

References

1. Anat Batat and Dror G. Feitelson. Gang scheduling with memory considerations. In *14th International Parallel and Distributed Processing Symposium*, 2000.
2. Compaq. Compaq AlphaServer series. http://www.compaq.com/alphaserver/platforms.html.
3. W. J. Dally, S.W. Keckler, N. Carter, A Chang, M. Filo, and W. S. Lee. M-Machine architecture v1.0. Technical Report Concurrent VLSI Architecture Memo 58, Massachusetts Institute of Technology, 1994.
4. Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, and Dean M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17(5), 1997.
5. D. G. Feitelson and A. M. Weil. Utilization and predictability in scheduling the ibm sp2 with backfilling. In *12th International Parallel Processing Symposium*, 1998.
6. Dror G. Feitelson and Larry Rudolph. Evaluation of design choices for gang scheduling using distributed hierarchical control. *Journal of Parallel and Distributed Computing*, 1996.
7. John L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, July 2000.
8. HP. HP 9000 superdome specifications. http://www.hp.com/products1/unixservers/highend/superdome/specifications.html.
9. IBM. RS/6000 enterprise server model S80. http://www-1.ibm.com/servers/eserver/pseries/hardware/enterprise/s80.html.

10. Jack L. Lo, Joel S. Emer, Henry M. Levy, Rebecca L. Stamm, Dean M. Tullsen, and S. J. Eggers. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, 15, 1997.
11. Eric W. Parsons and Kenneth C. Sevcik. Coordinated allocation of memory and processors in multiprocessors. In *the ACM SIGMETRICS conference on Measurement & modeling of computer systems*, 1996.
12. Gookwon Edward Suh, Srinivas Devadas, and Larry Rudolph. Analytical cache models with applications to cache partitioning. In *15th ACM International Conference on Supercomputing*, 2001.
13. Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture*, 1995.

Perceptual Interfaces

*Proc. of the European Conference on
Speech Communication and Technology,
Aalborg, Denmark, pp. 1335–1338,
September 2001.*

SpeechBuilder: Facilitating Spoken Dialogue System Development

James Glass and Eugene Weinstein *

MIT Laboratory for Computer Science
{glass, encoder}@mit.edu

Abstract. In this paper we report our attempts to facilitate the creation of mixed-initiative spoken dialogue systems for both novice and experienced developers of human language technology. Our efforts have resulted in the creation of a utility called SpeechBuilder, which allows developers to specify linguistic information about their domains, and rapidly create spoken dialogue interfaces to them. SpeechBuilder has been used to create domains providing access to structured information contained in a relational database, as well as to provide human language interfaces to control or transaction-based applications.

1 Introduction

As anyone who has tried to create a mixed-initiative spoken dialogue system knows, building a system which interacts competently with users, while allowing them freedom in what they can say and when they can say it during a conversation, is a significant challenge. For this reason, many systems avoid this tactic, and instead take a more strategic approach which focuses on a directed dialogue. In fact, many researchers argue that conversational, mixed-initiative dialogue systems may not be worth pursuing, both for practical and philosophical reasons. Certainly, there are many technical difficulties to overcome, which

---

* This research was supported by DARPA under contract N66001-99-1-8904 monitored through Naval Command, Control and Ocean Surveillance Center and under an industrial consortium supporting the MIT Oxygen Alliance.

include recognizing and understanding conversational speech, generating reasonable and natural responses, and managing a flexible dialogue strategy.

Over the past decade, the Spoken Language Systems Group at the MIT Laboratory for Computer Science has been actively developing the human language technologies necessary for creating such conversational human-machine interfaces. In recent years we have created several systems which have been publicly deployed on toll-free telephone numbers in North America, including systems providing access to information about weather forecasts, flight status, and flight schedules and prices [12, 9].

Although these applications have been successful, there are limited resources at MIT to develop a large number of new domains. To address this issue, we have recently set out to make it easier to rapidly prototype new mixed-initiative conversational systems. Unlike other portability efforts that we are aware of, which tend to employ directed-dialogue strategies (e.g., [10]), our goal is to enable the kinds of natural, mixed-initiative systems which are now created manually by a relatively small group of expert developers.

In this paper we describe our initial efforts in developing a utility called SpeechBuilder. The next section describes the architecture we have adopted. This is followed by a description of the human language technologies which are deployed in SpeechBuilder, and the knowledge representation it uses. We then describe the current state of development, followed by ongoing and future activities in this project.

2 System architecture

The approach that we have adopted for developing the SpeechBuilder utility has been to leverage the basic technology which is deployed in our more sophisticated conversational systems. There are many reasons for doing this. First, in addition to developing a programmable client-server architecture for conversational systems [8], we have devoted considerable effort over the last decade to improving human language technology (HLT) in speech recognition, language understanding, language generation, discourse and dialogue, and most recently, speech synthesis. By employing these HLT components we minimize duplication of effort, and maximize our ability to adopt any technical advances which are made in any of these areas. Second, by using our most advanced HLT components, we widen the pool of potential developers to include both novices and experts, since the latter can use SpeechBuilder to rapidly prototype a new domain (a very useful feature) and subsequently modify it manually. Third, since we are not limiting any of the HLT capabilities in any way, we allow for the potential for SpeechBuilder-created systems to eventually scale up to the same level of sophistication as our most capable systems. Lastly, by focusing attention on portability as a major issue, we can potentially identify weaknesses in some of our existing HLT components. This can lead to better solutions, which will ultimately benefit all of our conversational systems.

We have made a conscious decision to have as simple an interface as possible for the user, while providing mechanisms to incorporate any needed complexities. For example, developers do not specify natural language grammars for their domain. Instead, they specify the basic semantic concepts (called *keys*), and provide examples of user utterances which trigger different system behaviors (called *actions*). The system then uses these inputs to configure the language understanding grammar automatically. The developer can optionally create additional hierarchy in the grammar by using bracketing to label portions of the example sentences as being subject to a particular structure. Table 1 contains example sentences and their corresponding key/action representations encoded as CGI parameters (which are used for one kind of SpeechBuilder configuration).

2.1 SpeechBuilder Configurations

There are currently two ways in which a SpeechBuilder application can be configured. The first configuration can be used by a developer to create a speech-based interface to structured data. There is no programming required. As shown in Figure 1, this model makes use of the GALAXY architecture and all of the associated HLT components to access information stored in a relational database which is populated by the developer. This database table is used along with the semantic concepts and example utterances to automatically configure the speech recognition, language understanding, language generation (including both SQL and response generation), and discourse components (described in Section 3). A generic dialogue manager handles user interactions with the database. Armed with a table of structured data, an experienced developer can use SpeechBuilder to create a prototype system in a matter of minutes.

The second possible SpeechBuilder configuration provides the developer with total control over the application functionality, as well as the discourse, dialogue, and response generation capabilities [6]. In this model, the developer creates a program implementing domain-specific

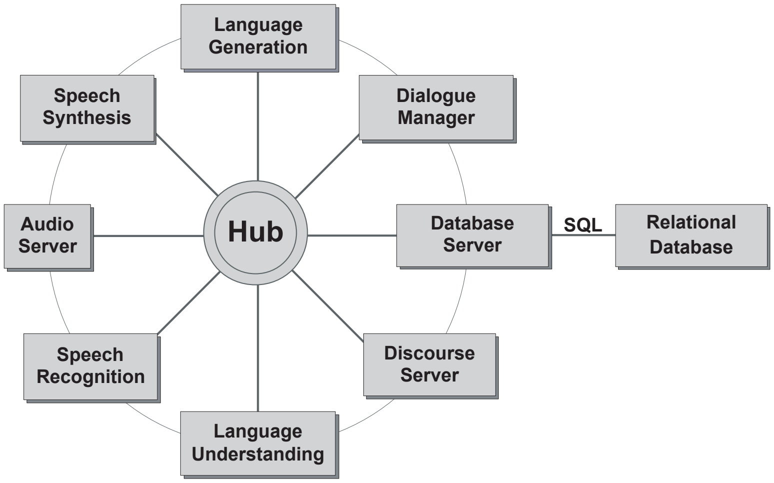


Fig. 1. SpeechBuilder configuration for database access.

turn on the lights in the kitchen **action=set&frame=(object=lights,room=kitchen,value=on)**
will it be raining in Boston on Friday **action=verify&frame=(city=Boston,day=Friday,property=rain)**
show me the Chinese restaurants on Main Street **action=identify&frame=(object=(type=restaurant, cuisine=Chinese,on=(street=Main,ext=Street)))**
I want to fly from Boston to San Francisco arriving before ten a m **action=list&frame=(src=BOS,dest=SFO, arrival_time=(relative=before,time=(hour=10,xm=AM)))**

Table 1. Example sentences and their CGI representations.

functionality and deploys it on a CGI-enabled web server. As shown in Figure 2, this configuration uses a subset of the GALAXY components. The semantic frame is converted to the CGI parameter representation shown in Table 1 by means of the language generation component and is sent to the developer CGI application using HTTP.

Since the developer CGI application is stateless, the SpeechBuilder server maintains a dialogue state variable which is exchanged with the CGI application on every turn. It is the responsibility of the application to decide what information, if any, to inherit from this state variable, and what information to retain for the next dialogue turn. Table 2 shows how the state information can be used to keep track of local discourse context.

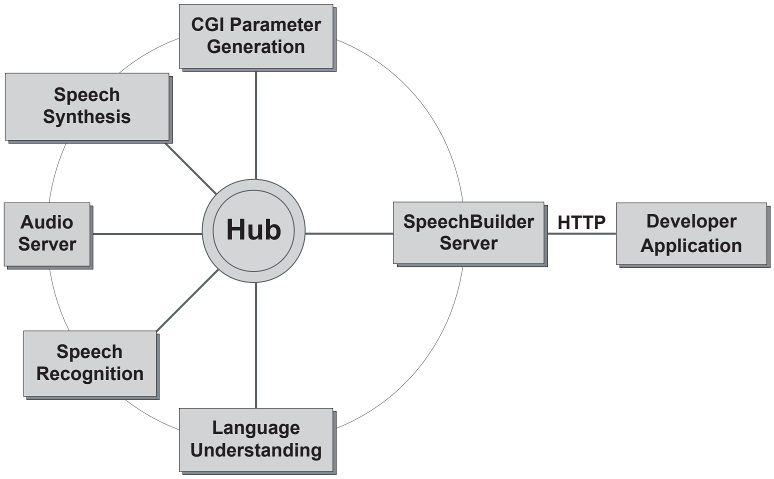


Fig. 2. Alternative SpeechBuilder configuration.

what is the phone number of John Smith **action=identify&frame=(property=phone,name=John+Smith)**
what about his email address **action=identify&frame=(property=email)** **&history=(property=phone,name=John+Smith)**
what about Jane Doe **action=identify&frame=(name=Jane+Doe)** **&history=(property=email,name=John+Smith)**

Table 2. Example interaction using the dialogue state variable.

2.2 Web-based interface

SpeechBuilder employs a web-based interface, and is implemented via a number of Perl CGI scripts. The web interface allows the developer to manipulate all of the domain specifics, such as keys and actions, query responses, and vocabulary word pronunciations. These domain specifics are stored in an XML document, and the developer has the ability to edit this file manually without using the online interface. The web interface contains a facility for downloading and uploading the XML representation of a domain, and for uploading CSV (comma separated value) representations of data tables. The developer also uses the web interface to compile the domain (that is, to configure the galaxy servers according to domain specifics), and to start and stop the runtime modules of the domain. Once a domain has been compiled, the developer can also examine the parse tree and semantic frame produced for each example sentence. The interface makes it very easy for a developer to continually modify, recompile, and redeploy a domain during the development cycle.

3 Human language technologies

As shown in Figure 1, SpeechBuilder makes use of all the major galaxy components [8]. The programmable hub executes a program which was created specifically for the SpeechBuilder application, although it contains all of the functionality of the programs used by our main systems. A specific version of the hub program is configured for each developer.

The speech recognizer is configured to use generic telephone-based acoustic models, and is connected to the language understanding component via an N-best interface [4]. Since users may speak words which are not specified in the vocabulary, we have incorporated an out-of-vocabulary model [2]. Baseform pronunciations which do not occur in our large on-line dictionaries are generated by rule [3]. SpeechBuilder provides an editing facility for developers to modify pronunciations. The recognizer deploys a hierarchical n-gram grammar derived from the language understanding grammar rules and the example sentences provided by the developer.

For language understanding, SpeechBuilder configures a grammar file as well as a file for converting full or partial parses into a meaning representation [7]. Language understanding is configured to back off to robust parsing (i.e., concept spotting) when no full parse is available. The discourse server is based on a new component which performs inheritance and masking after an internal electronic ‘E-form’ has been generated from the semantic frame [9].

The dialogue management server was modeled after the functionality of our main systems [9]. Since this component still tends to be extensively customized for every domain, we created a simple generic server which is intended to handle the range of situations which can arise in database query domains. We fully expect this component to increase in complexity as we consider a wider range of domains.

The language generation component is actually used for generating three different outputs, as it is in our main systems [1]. The first use of generation is for creating the ‘E-form’ representation used by the discourse and dialogue components. The second use is to generate an SQL query for use by the database server. The third use is to generate a response to the user which is vocalized using the speech synthesizer.

4 Knowledge representation

Linguistic constraints are specified by a developer in terms of a set of concept keys and sentence-level actions via the web interface. Each is described in more detail in the following.

4.1 Concept keys

Concept keys usually define classes of semantically equivalent words or word sequences. All the entries of a key class should play the same role in an utterance. Concept keys can be extracted from the database table or can be manually specified by the developer through the web-based interface. In order to appear in the semantic frame a concept *must* be a member of a concept key class. A regularization mechanism allows the developer to specify variations on the spoken form (e.g., “Philadelphia,” vs. “Philly”) that map to a standardized form (e.g., “PHL”). Table 3 contains examples of concept keys.

Key	Examples
color	red, green blue
day	Monday, Tuesday, Wednesday
room	living room, dining room, kitchen

Table 3. Examples of concept keys.

4.2 Sentence-level actions

Actions define classes of functionally equivalent sentences, so that all the entries of an action class perform the same operation in the application. Actions are example queries that one might use in talking to the domain. Action labels determine the clause name of the semantic frame produced by the language understanding component. SpeechBuilder generalizes all example sentences containing particular concept key entries to accept all the entries of the same key class, and thus builds the natural language template. SpeechBuilder also tries to generalize the non-key words in the example sentences so that it can understand a wider variety of user queries than was provided by the developer. Table 4 contains example actions.

Action	Examples
identify	what is the forecast for Boston what will the temperature be on Tuesday I would like to know today's weather in Denver
set	turn the radio on in the kitchen please can you please turn off the dining room lights turn on the TV in the living room

Table 4. Examples of actions.

The system created by SpeechBuilder can potentially understand a larger set of queries than are defined by the set of sentence examples, since the examples are converted into a hierarchical n-gram for the recognizer, and the understanding component backs off to concept spotting when a complete parse is not found. However, the system performs better if given a richer set of examples.

4.3 Hierarchical concept keys

SpeechBuilder allows the developer to build a structured grammar when this is desired. This is done by "bracketing" example sentences within the actions – using parentheses to enforce a structure on the way that each particular sentence is parsed. The semantic frame created by the natural language processor reflects the hierarchy specified by the bracketing.

To bracket a sentence, the developer encloses the substructure which they wish to separate in parentheses, preceded by a name for the substructure followed by either == or =, depending on whether the developer desires to use *strict* or *flattened* hierarchy. Strict hierarchy maintains the key/value structure of all concept keys present in the bracketed text. In contrast, flattened hierarchy collapses all internal key/values into a single concept value. Table 5 shows examples of bracketed example sentences which make use of the hierarchical concept keys.

Put object==(the blue box) location==(on the table) **object=(color=blue,item=box),location=(item=table)**
Put object=(the blue box) location=(on the table) **object=(blue_box),location=(table)**
Put the box location==(on the table location==(in the kitchen)) **item=box,location=(relative=on,item=table,** **location=(relative=in,room=kitchen))**

Table 5. Examples of strict (==) and flattened (=) hierarchy.

4.4 Responses

In addition to configuring ways of asking about information, the developer must also specify how the system will present information to the user. When a database is provided by the developer, SpeechBuilder configures a generic set of replies for each appropriate database query which could be requested by a user. In addition, responses are generated to handle situations common to all database applications (e.g., no data matching the input constraints, multiple matches to a user query, too many matches, etc.). Each of these default responses can be modified by the developer, as desired, to customize the domain.

5 Current status

SpeechBuilder has been accessible from within MIT and limited other locations for beta-testing since November 2000, and it has been used to create several different domains. The *LCSinfo* domain, for example, provides access to contact information for the approximately 500 faculty, staff, and students working at the MIT Laboratory for Computer Science (LCS) and Artificial Intelligence Laboratory (including phone numbers, email addresses, room locations, positions, and group affiliations) and is able to connect the caller to any of the people in the database using call bridging. Applications similar to *LCSinfo* domain are under development by novice developers elsewhere at MIT, as well as at the Space and Naval Warfare Systems Center in San Diego, CA. SpeechBuilder has also been used to create a simple appliance application which controls various physical items in an office (e.g. lights, curtains, projector, television, computer, etc.). This domain is now being used within LCS. SpeechBuilder has also been used by members of our group, as well as our industrial partners to create small mixed-initiative database access applications (e.g., schedule information, music queries, stock quotes).

In order to allow developers to test out their systems, we have set up a centralized telephone access line, which developers can use to access their deployed domains. In addition, we have provided a local-audio setup so that those developers with access to the GALAXY code distribution can run their domains on their own machines independently of any MIT hardware.

6 Ongoing and future activities

There are many ongoing and future activities which will improve the SpeechBuilder infrastructure. For example, in order to be able to handle more complicated schema for information access, we plan to introduce the ability to access multiple intersecting database tables from one domain. Currently, all database cells are treated as strings, but in the future we plan to implement functionality for handling Boolean and numerical values as well.

Although we have created an initial database query dialogue manager, we plan to continue to add features which will provide the developer more control for their particular application. A related area of future research will be to improve the current communication protocol for non-database applications.

In areas of speech recognition and understanding, we plan to incorporate our recent work in confidence scoring [5]. In the longer term we would like to introduce unsupervised training of acoustic and linguistic models. Currently, developers can only upload transcriptions, which are used as training data for these components.

The current configuration of the system defaults to using a commercial speech synthesizer. We would like to provide a facility to developers whereby they can configure a simple concatenative speech synthesizer [11].

The work we have done thus far has been restricted to American English. However, as we have ongoing research efforts in multilingual conversational systems (e.g., Japanese, Mandarin Chinese, and Spanish), we have begun to modify the SpeechBuilder interface so that multilingual systems can be configured as well.

The systems developed thus far have been prototypes that have not been rigorously evaluated. In order to demonstrate the usefulness of the currently deployed architecture, we plan to configure a weather information domain within SpeechBuilder, and evaluate at least the speech recognition and understanding components on our standard test sets [12]. By demonstrating at least compatible performance with our manually created systems, we hope to show the potential for creating robust, yet rapidly configurable, spoken dialogue systems.

7 Summary

This paper has provided an overview of our efforts to facilitate the creation of mixed-initiative spoken dialogue systems. If successful, we believe this research will benefit others by allowing people interested in spoken dialogue systems to rapidly configure applications for their particular interests. We have successfully deployed two distinct configurations of the SpeechBuilder utility which connect to applications located on a remote web-based server, or to a local database. Several prototype systems have been created with this utility, which are currently in active use. In the near future, we hope to extend the current framework so as to allow for the creation of more complex and powerful spoken dialogue systems.

8 Acknowledgements

Many people in the SLS group contributed to this work, including Issam Bazzi, Scott Cyphers, Ed Filisko, TJ Hazen, Lee Hetherington, Jef Pearlman, Joe Polifroni, Stephanie Seneff, Chao Wang and Jon Yi.

References

1. L. Baptist and S. Seneff. GENESIS-II: A versatile system for language generation in conversational system applications. In *Proc. ICSLP*, Beijing, 2000.
2. I. Bazzi and J. Glass. Modeling out-of-vocabulary words for robust speech recognition. In *Proc. ICSLP*, Beijing, 2000.
3. A. Black, K. Lenzo, and V. Pagel. Issues in building general letter to sound rules. In *ESCA Speech Synthesis Workshop*, Jenolan Caves, 1998.

4. J. Glass, T. Hazen, and L. Hetherington. Real-time telephone-based speech recognition in the JUPITER domain. In *Proc. ICASSP*, Seattle, 1999.
5. T. Hazen, T. Burianek, J. Polifroni, and S. Seneff. Recognition confidence scoring for use in speech understanding systems. In *Proc. ISCA ASR Workshop*, Paris, 2000.
6. J. Pearlman. SLS-LITE: Enabling spoken language systems design for non-experts. M.Eng Thesis, 2000.
7. S. Seneff. Tina: A natural language system for spoken language applications. *Computational Linguistics*, 18(1), 1992.
8. S. Seneff, R. Lau, and J. Polifroni. Organization, communication, and control in the GALAXY-II conversational system. In *Proc. Eurospeech*, Budapest, 199.
9. S. Seneff and J. Polifroni. Dialogue management in the MERCURY flight reservation system. In *Proc. ANLP-NAACL 2000, Satellite Workshop*, Seattle, 2000.
10. S. Sutton and et al. Universal speech tools: The cslu toolkit. In *Proc. ICSLP*, Sydney, 1998.
11. J. Yi, J. Glass, and L. Hetherington. A flexible, scalable finite-state transducer architecture for corpus-based concatenative speech synthesis. In *Proc. ICSLP*, Beijing, 2000.
12. V. Zue and et al. JUPITER: A telephone-based conversational interface for weather information. *IEEE Trans. Speech and Audio Proc.*, 8(1), 2000.

Proceedings of ICSLP'00, Vol. II,
pp. 142–145, Beijing, China, October,
2000.

ORION: From On-line Interaction to Off-line Delegation

Stephanie Seneff, Chian Chuu and D. Scott Cyphers*

MIT Laboratory for Computer Science
`{seneff, cchuu, cyphers}@lcs.mit.edu`

Abstract. This paper introduces ORION, a conversational system that performs off-line tasks and initiates later contact with a user at a pre-negotiated time. Orion has two major episodes of activity: the enrollment of new tasks and the execution of pending tasks. The task manager periodically checks the pending tasks and updates their status, sending off requests to other servers for information and possibly launching a phone call when a particular task has reached its trigger time. A separate user interface engages in a dialogue with a user to enroll new tasks and/or update existing tasks. ORION is still in an early stage of its development cycle, but it has introduced several interesting new research issues, such as continuous state maintenance and contact verification.

1 Introduction

For more than a decade, the Spoken Language Systems (SLS) Group at the MIT Laboratory for Computer Science has been conducting research leading to the development of conversational interfaces: interfaces that will enable naive users to access and manage information using spoken dialogue. To realize such systems, several language-based input and output technologies, including speech recognition/synthesis and language understanding/generation have been developed and integrated. Typically, these systems engage the user in a dialogue to retrieve information from databases.

---

* This work was supported by DARPA under contract N66001-99-1-8904 monitored through Naval Command, Control and Ocean Surveillance Center.

Until now, all of our systems have assumed that each task is completed as soon as the user hangs up the phone. However, it seems reasonable to suppose in principle that a conversational system could perform certain tasks *off-line*, i.e., that the user could *delegate* information-dependent activities to the system, which would later inform them of the outcome, either through e-mail or through system-initiated telephone contact.

This paper describes our newest conversational domain, called ORION, which for the first time begins to address the idea of delegation. The user can request a task to be executed at some later time, and ORION must then initiate a follow-up contact with the user, once the assigned task has been completed. Some of the tasks involve monitoring a dynamically changing database over time. Others involve a single look-up at the designated call-back time.

The Orion system is configured using the Galaxy Communicator architecture, in which all servers communicate via a common hub. Orion consults other domain servers to retrieve critical information. The Orion server plays two distinct roles, each implemented as a separate stream. One is devoted to the enrollment of new tasks and the other is concerned with the execution of existing tasks. Users first enroll by providing critical information about their name, appropriate phone numbers, and e-mail address. To edit existing tasks or add new tasks, the user interacts with ORION at a Web site, in conjunction with conversation through speech over the telephone, or through typing at the GUI interface. Pending tasks are displayed in the graphical interface, and ORION engages the user in a mixed-initiative conversation until a new task is fully specified, or a pre-existing task is appropriately modified. If a new task needs to be executed today, it is sent to the agent stream for an immediate update. Orion can currently handle a number of types of requests, as suggested by the examples given in Figure 1.

Call me at 4 p.m. tomorrow to remind me to pick up my son at soccer practice.
Call me every weekday morning at 6:30 a.m. and tell me the weather in Boston.
Call me an hour before American flight 93 lands in Dallas.
Call me at work between 5 and 6 p.m. if the traffic on route 93 is at a standstill.

Fig. 1. Examples of the types of tasks that Orion can currently handle.

A unique aspect of ORION is that it behaves like a user in requesting information from other domains. ORION finds it convenient to make these requests using standard English as a natural and intuitive scripting language.

The notion of off-line delegation is a radical departure from our prior research activities, and it opens up a number of interesting research issues, such as continuous maintenance of state information, dealing with the termination of or corrections to previously assigned reiterant tasks, and verification that the call-back was successful. A domain of the degree of complexity of ORION would not have been feasible for us to develop prior to the introduction of the GALAXY Communicator architecture [2].

In the remainder of this paper, we will first describe the task negotiation phase, followed by the task execution phase. We will then discuss a number of issues that have emerged as a consequence of our explorations with ORION. We conclude with a discussion of some of our future visions.

2 Task negotiation phase

The first step in task negotiation is to enroll a new user. This is done at a graphical interface where ORION asks a series of questions and the user types the responses into a GUI type-in window integrated into a Web browser. ORION seeks three phone numbers (home, work, and cell) as well as the user's name and e-mail address. Once the user has enrolled, they can type, "call me now at work," to initiate an immediate call-back for spoken interaction, or they can continue to interact through typed queries.

ORION then launches a mixed initiative dialogue to invite the user to specify a task, or to edit any existing tasks, should this be a follow-up interaction. In the latter case, all pre-existing tasks are displayed to the user in the graphical interface. The user could, for example, click on a given task and say, "Edit this task."

If the user intends a new task, they can simply specify it through speech. They might say, for example, "Call me to remind me to call my sister." ORION will prompt the user for a date, time, and phone number, where "work," "home," and "cell" are all possible, in addition to a unique newly introduced task phone number. Figure 2 shows an example dialogue between a user and ORION, where the user is requesting a wake-up call the following morning.

If the user cannot remember the particulars of a task, he/she can ask to talk to one of the other domains. The user might, for example,

Orion: Welcome to the Orion planning system. Please type your user name, or, if you have not enrolled, type 'new user'.
User: Stephanie Seneff

Orion: Here is your personal information in our database, Stephanie Seneff. What is the task that you have in mind? (wakeup call,reminder, flight status, or traffic update) <**shows user profile and list of prior tasks**>
User: Call me at six thirty tomorrow morning and tell me the weather in Boston.

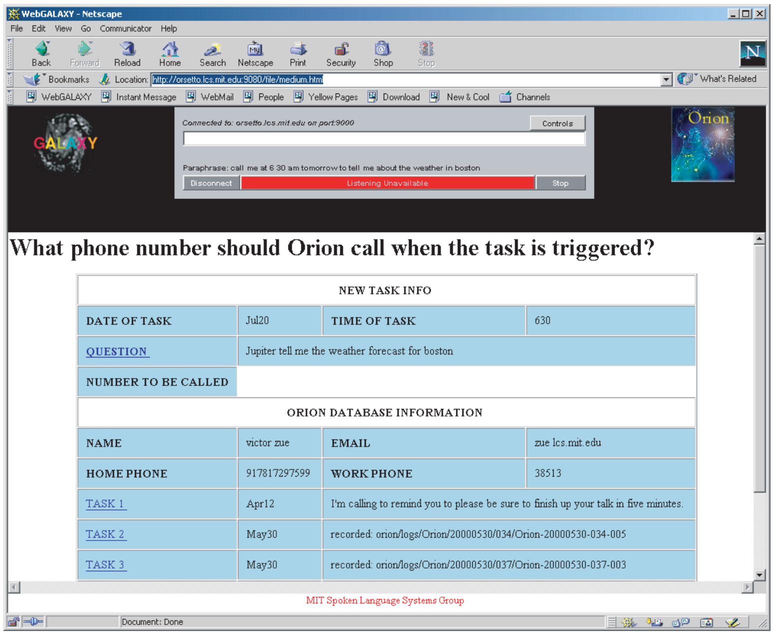
Orion: What phone number should I call you at?
User: Call me at home.

Orion: You have entered all the necessary data, stephanie seneff. Is the following information correct? <**shows task**>
User: Yes.

Orion: You will receive a call at the appropriate time. Thank you for using Orion.

Fig. 2. Example dialogue between a user and Orion negotiating a wake-up call, in the context of a graphical interface.

find out from the PEGASUS flight status domain or the MERCURY flight reservations domain [3] the flight number of a flight they are interested in monitoring. In the same phone conversation, the user can return to ORION, and ORION will be able to obtain directly from MERCURY's dialogue state all the information concerning the flight in focus, which ORION then assumes is of relevance to the conversation.



Connected to: orsetto.lcs.mit.edu on port:9000

Paraphrase: call me at 6 30 am tomorrow to tell me about the weather in boston

What phone number should Orion call when the task is triggered?

NEW TASK INFO			
DATE OF TASK	Jul20	TIME OF TASK	630
QUESTION	Jupiter tell me the weather forecast for boston		
NUMBER TO BE CALLED			
ORION DATABASE INFORMATION			
NAME	victor zue	EMAIL	zue lcs.mit.edu
HOME PHONE	917817297599	WORK PHONE	38513
TASK 1	Apr12	I'm calling to remind you to please be sure to finish up your talk in five minutes.	
TASK 2	May30	recorded: orion/logs/Orion/20000530/034/Orion-20000530-034-005	
TASK 3	May30	recorded: orion/logs/Orion/20000530/037/Orion-20000530-037-003	

MIT Spoken Language Systems Group

Fig. 3. The graphical interface for Orion, showing the task under negotiation, the user's profile, and a list of three previously specified tasks.

Figure 3 shows a GUI interaction with ORION. Once the new task is fully specified and confirmed, ORION updates the user's personalized file of pending tasks. If the task is scheduled for today, ORION wakes up the agent stream via a special hub-mediated operation devoted to newly introduced tasks, as will be more fully described in the next section.

If the user asks to be called at some future time, but does not mention a particular task, ORION by default invites them to pre-record a message to be played back at the designated time. This is a very general device with wide utility, that allows the user to say anything at all in the recording, subject to a length constraint.

QUERY: ”Call me at six a m and tell me the weather in Boston”

LINGUISTIC FRAME:

```
{c call_me
   :domain "Orion"
   :reason
     {c inform
        :topic {q weather
                  :quantifier "def"
                  :pred {p in
                           :topic
                             {q city
                                :name "boston"
                             }
                         }
                }
      }
   :pred {p at :topic {q time :hour 6
                              :xm "am" } } }
```

E-FORM:

{c eform :action “call_me” :time “6:00 am” :clause “inform” :task_label “weather” :in “boston” }

Fig. 4. Example linguistic frame and *e*-form for a user query in the ORION domain.

In order to plan its interactions with the user and the other servers, ORION makes use of a dialogue control table, as described more fully in [3]. The GENESIS-II [1] generation server is invoked to paraphrase users’ requests into an appropriate electronic form (henceforth *e*-form) format, which is used to initialize the dialogue state at each new turn. Figure 4 shows the query, the semantic frame, and the derived *e*-form obtained via GENESIS-II for the sentence, “Call me at six a m and tell me the weather in Boston.” The dialogue state is consulted to determine which functions to call to carry out the task-specification phase, using dialogue control procedures that are common to all of our domain servers.

GENESIS-II is also responsible for altering the pronominal references from first to second person, so that “call me to remind me to call my sister” becomes “This is ORION calling to remind *you* to call *your* sister.”

3 Agent activities

The agent stream currently maintains information about all of its users in data structure files that are reloaded every day at midnight. It first determines which tasks are possible on the current day, including reiterant tasks (e.g., “every weekday”). It then creates an internal data structure for each pending task that includes slots for various temporal events such as the next update time or the estimated (or exact) *trigger* time (time to contact the user).

To execute its various plans, the system consults a second dialogue control table [3], distinct from the one used to plan new tasks. It iterates through the dialogue table repeatedly until all tasks are completed as fully as possible. At midnight, this would typically include consulting the MERCURY flight information server to determine the scheduled departure and arrival times of any flights being monitored.

Once all tasks have been initiated, the ORION agent stream goes into a sleep cycle until the minimum next update time of its set of pending tasks. However, it could be awakened at any time by a new request from a user who has just finished specifying all the particulars of this request, or who may have altered a previous request that now needs immediate attention. These interruptions are implemented as a separate operation, invoked by a rule in a hub script, according to the GALAXY Communicator design [2].

When the next update time has been reached, ORION consults a dialogue control table to determine which functions to call. Functions may involve module-to-module subdialogues with other domains to look up information about weather [5], flights [3], or traffic. These subdialogues are controlled by a separate hub program appropriately named *orion*, and they involve calls to the natural language server to parse ORION’s request, and calls to the designated turn manager to fully interpret the parsed frame and produce a reply frame. This is accomplished via a database request initiated in a nested module-to-module subdialogue (through normal channels used for all user requests). Selected rules from the *orion* program are shown in Figure 5.

```
PROGRAM: orion

RULE:     :call_request → dispatch_to_main
IN:       (:input_string :call_request) :domain :recording

RULE:     !:request_frame & :input_string → create_frame
IN:       :input_string :domain
OUT:      :request_frame :domain

RULE:     :request_frame & :domain Jupiter → jupiter.turn_management
IN:       :request_frame
OUT:      :reply_frame

RULE:     !:reply_string & :reply_frame → paraphrase_reply
IN:       :reply_frame :domain (:out_lang english)
OUT:      :reply_string :reply_frame
```

Fig. 5. Example rules from the *orion* program used by the GALAXY hub. Note: “!” stands for “NOT”, and “&” is a logical “AND.”

For example, if ORION has been requested to call the user at some designated time relative to the arrival of a flight, it monitors the flight at appropriate intervals throughout the period of time that the flight is in the air. Its algorithm is to cut the time by half between the current time and the currently estimated arrival time to determine when to next check the flight status. Checking the status involves a contact with the PEGASUS flight status domain. As soon as the estimated callback time is reached, ORION delegates to PEGASUS the task of actually calling the user.

When a trigger time is reached, ORION executes the function named “call_user_now,” but only after it has prepared a frame containing an appropriate welcome message. If the task involves a particular domain of expertise, ORION currently delegates the phone call to the appropriate domain server[1]. From ORION’s standpoint, this is easy to do: it just prepares a query of the form “<domain_name> call me at <task phone number>.” The *orion* hub program dispatches the query to the main program[2], along with the welcome message, which might be prerecorded. It follows normal channels, modelled after the pre-existing

---

[1] Although the user can ask to speak to ORION should they have further tasks in mind.

[2] see the dispatch_to_main operation in the example rules of Figure 5.

feature that a user can type a call-back request into a GUI interface. For reminders that don't invoke any of the other domains, e.g., "call me to remind me to pick up my son," ORION itself initiates the call, and is then prepared to negotiate further tasks if the user so desires.

4 Unresolved issues

ORION is an early prototype of a conversational intelligent agent (IA) that interacts with the real world. We have barely tapped into the large body of research in IAs at this time [4]. Special considerations are required when IAs are exposed to the real world, several of which are discussed here.

Security ORION must be protected from hackers who could potentially use it to place unwanted calls, change other users' tasks, etc. We must devise secure mechanisms for user authentication, and extend existing access control mechanisms to conversational systems.

Reliability ORION must be dependable. Tasks must survive system crashes. Infrastructure must be developed to reduce the complexity of our existing system development and support the new capabilities required by ORION.

Efficiency ORION currently uses a polling mechanism to track actions. Polling is complex and innefficient, and will not scale as the number of users and tasks increases.

In the future, we expect GALAXY to make use of daemon agents to update system data such as weather and flight information as we receive it from our data sources. For example, a user agent can tell the flight information update daemon to notify it whenever flight information about United flight 805 changes, and then sleep until thirty minutes before arrival. If the flight information changes, the daemon agent notifies the user agent which wakes up, reschedules its future wakeup, and goes back to sleep. This approach reduces complexity because it allows the same user agent code to be used for weather and flight notifications, and it increases efficiency because the agent only runs when it has something to do.

Resource Management To satisfy ORION's future needs, GALAXY will need to support an additional layer of distributed operations that provide common services like speech recognition, virtual device management (such as a pool of phone lines), and persistence to agents. Systems such as JUPITER and MERCURY will be collections of related agents that ORION interacts with.

Common Sense If a user asks for a wakeup call every day at 6:00 a.m., ORION should be aware that this means local time for the user.

If ORION knows a particular user will be travelling during a certain time interval, it should be able to pro-actively enquire whether the call should still happen, and, if so, perhaps suggest calling on the cell phone. Also, how should ORION negotiate day by day monitoring of such reiterant tasks? If, on Friday, the user says, "Don't call me Monday," should ORION then enquire about Tuesday, or assume that Tuesday is the next time to call?

Customization We expect that in the future ORION will become increasingly capable of managing an experienced user's tasks. ORION should be able to accumulate information on patterns based on past experience, such that it will be increasingly able to take the initiative in proposing or even executing specific plans. Thus, for example, in the future, an expert user might be able to ask ORION to make travel arrangements to Chicago, and ORION would be able to make decisions about the flights, hotel, and rental car, calling back only to specify a fully executed solution.

Verification If the user fails to answer the phone when ORION calls, then ORION would need to recognize that the call is incomplete, and perhaps try again ten minutes later, and/or send e-mail informing the user of the failed contact. A more serious problem is a pick up by an answering machine. How should ORION be made aware of this situation, and what should it do? Another problem is the recognition of the voice of the user, or a verbal verification that the person answering the phone is indeed the intended user.

Social Users will be reluctant to depend on a system like ORION for important tasks, particularly if the agents will be searching for information, making decisions, or spending the user's money. We will need to provide ways for users to check the reasoning process and to verify that ORION is really planning to do what they think it will do.

5 Summary and future plans

This paper introduces ORION, a system devoted to off-line delegation as a new paradigm in conversational systems. ORION was made feasible as a consequence of the powerful capabilities of the GALAXY Communicator architecture. We found that it was straightforward to leverage off of pre-existing mechanims with only minor modifications to carry out both the server initiated queries to other domains and the user call-back procedures. Thus far ORION has been used only by researchers in our group and their families, as well as in live demonstrations. We have not yet begun the task of data collection from naive users, as the system is not sufficiently mature.

We expect that ORION represents the early stages of an ambitious long-term project. In the future, we envision that people will be able to take advantage of available on-line information systems such that routine tasks can be delegated to the computer as much as possible, thus freeing humans to attend to tasks that truly need their attention. We will explore how we can build systems that can easily customize and adapt to the users' needs and desires. In the process, we will also examine how a system such as ORION could be made more intelligent by incorporating into its decision-making process all available information at its disposal, both about the domain and about the individual user. We believe that ORION has tremendous potential, although we are still working out the details of some of the more difficult aspects of its design, including user identification, customization, resource allocation, and task completion verification.

References

1. L. Baptist and S. Seneff. GENISIS-II: A versatile system for language generation in conversational system applications. In *Proceedings of ICSLP'00*, Beijing, China, 2000.
2. S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. GALAXY-II: A reference architecture for conversational system development. In *ICSLP '98*, pages 931–934, Sydney, Australia, December 1998.
3. S. Seneff and J. Polifroni. Dialogue management in the MERCURY flight reservation system. In *ANLP-NAACL 2000*, Seattle, WA, May 2000.
4. Yoav Shoham. An overview of agent-oriented programming. In Jeffrey M. Bradshaw, editor, *Software Agents*. AAAI Press/The MIT Press, Cambridge, Massachusetts, 1997.
5. V. Zue, S. Seneff, J. Glass, J. Polifroni, T.J. Hazen C. Pao, and L. Hetherington. JUPITER: A telephone-based conversational interface for weather information. *IEEE Trans. on Speech and Audio Processing*, 8(1):85–96, January 2000.

Proceedings of the International
Conference on Computer Vision
(ICCV'01), Vancouver, Canada,
2001.

Plan-View Trajectory Estimation with Dense Stereo Background Models

Trevor Darrell, David Demirdjian, Neal Checka
and Pedro Felzenszwalb

MIT Artificial Intelligence Laboratory
{`trevor, demirdji, checka, pff`}`@ai.mit.edu`

Abstract. In a known environment, objects may be tracked in multiple views using a set of background models. Stereo-based models can be illumination-invariant, but often have undefined values which inevitably lead to foreground classification errors. We derive dense stereo models for object tracking using long-term, extended dynamic-range imagery, and by detecting and interpolating uniform but unoccluded planar regions. Foreground points are detected quickly in new images using pruned disparity search. We adopt a "late-segmentation" strategy, using an integrated plan-view density representation. Foreground points are segmented into object regions only when a trajectory is finally estimated, using a dynamic programming-based method. Object entry and exit are optimally determined and are not restricted to special spatial zones.

1 Introduction

Tracking people in known environments has recently become an active area of research in computer vision. Several person tracking systems have been developed to detect the number of people present as well as their 3-D position over time. These systems generally use a combination of foreground/background classification, clustering of novel points, and trajectory estimation in one or more camera views [18, 16, 10, 13, 7, 17, 5]

Many color-based approaches to background modeling have considerable difficulty with fast illumination variation due to changing lighting and/or video projection. To overcome this, several authors have advocated the use of background shape models based on stereo range data [7, 5, 11]. Unfortunately, the background models built by these systems are often sparse, due to the many regions of uniform brightness where stereo estimation fails in a typical background training sequence. Additionally, most of these systems are based on exhaustive stereo disparity search.

In contrast, we show here how dense, fast range-based tracking can be performed with modest computational complexity. We recover dense depth data using multiple-gain imaging and long-term observation approaches. We match uniform but unoccluded planar regions in the scene and interpolate their interior range values. We apply ordered disparity search techniques to prune most of the disparity search computation during foreground detection and disparity estimation, yielding a fast, illumination-insensitive 3-D tracking system.

When objects are moving on a ground plane and are observed from multiple widely-separated viewpoints, rendering an orthographic vertical projection of foreground activity is useful [13, 3, 2, 17]. A "plan-view" image facilitates correspondence in time since only 2D search is required. Typically, previous systems would segment foreground data into regions *prior to* projecting into a plan-view, followed by region-level tracking and integration, potentially leading to sub-optimal segmentation and/or object fragmentation. (But see [12] for a way to smooth fragmented trajectories.)

Instead, we develop an approach which altogether avoids any early segmentation of the foreground data. We merge the plan-view images from each viewpoint and estimate over time a set of trajectories that best accounts for the integrated foreground density. Trajectory estimation is performed using a dynamic programming-based algorithm, which can optimally estimate the position over time as well as the entry and exit locations of an object. This contrasts previous approaches, which generally used instantaneous measures, and/or specific object creation zones to decide on the number of objects per frame [3, 13].

In the next section, we detail our new algorithm for computing dense range-based foreground estimates and for fast estimation of foreground disparities. Following that, we introduce a plan-view tracking representation and our algorithm for optimally estimating trajectories with limited temporal extent. We show how this method can accurately detect the entry and exit of objects without constraints on the spatial location of such events. We finish with a discussion of the overall system's performance and implications, as well as possible avenues for future work.

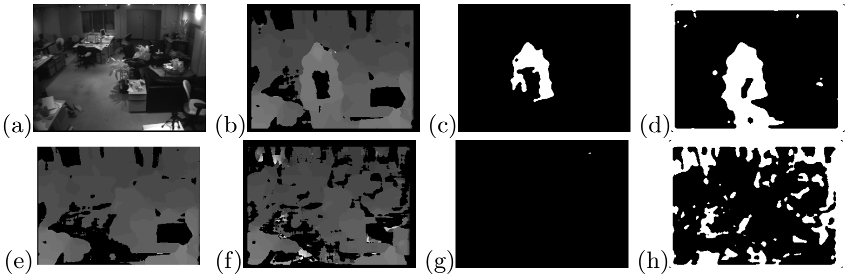


Fig. 1. The problem with sparse range backgrounds. Given a sparse background model (e) of a scene (a), a new range image with a foreground person (b), and a new range image with no foreground object but a changed illumination condition (f), we see that a conservative segmentation (c,g) misses many foreground points on the object. However the alternative approach (d,h), has many false positives when the illumination changes, and erroneously includes background points in the foreground. To achieve illumination invariance one must adopt a conservative approach and obtain very dense range background models.

2 Range-based foreground detection

Segmentation of foreground regions using range measurements is inherently robust to the illumination variation that disrupts most color-based approaches. However, when range data is used directly to build background models, experience shows that the models are often sparse and are well-defined at fewer points than in a color model [7, 5]. These background models will have pixels which are set to an "unknown" depth value.

With a sparse depth map, one has to decide whether to detect foreground pixels when the background is invalid and a new range value is observed. The conservative option–not declaring the pixel foreground–will forever prevent detection of any valid foreground points in the empty regions of the background. This will lead to Type I errors (Figure 1(c).) The alternative approach, declaring a pixel to be foreground even when the background is undefined, leads to Type II errors. If imaging condition changes such that a previously uniform background region suddenly has contrast and a defined range value, then a background point will incorrectly be declared a foreground point. (Figure 1(h)). This can commonly happen when the illumination level changes and pixels de-saturate, or when shadows or other projections are cast on a uniform surface.

To overcome this problem we construct dense background models with long-term and extended dynamic-range data. We can resolve depth values within unoccluded, uniform, and planar regions using a constraint on the appearance of these regions in two views. We also use predictive disparity search to prune unnecessary computation and quickly estimate foreground regions of new images.

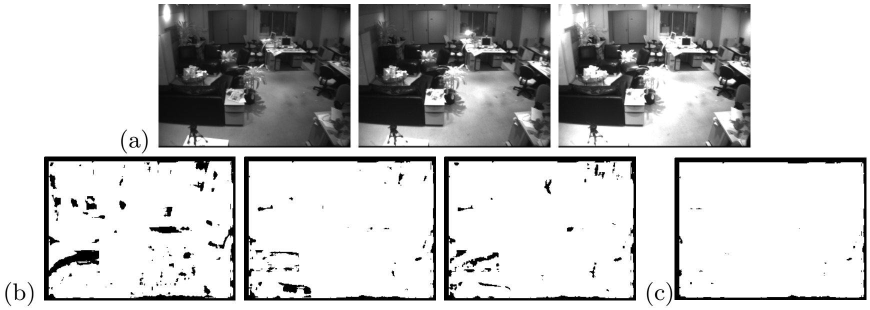


Fig. 2. Stages in building a dense background model. (a) Examples of variable gain and illumination conditions for a scene. (b) Map of valid disparity values for each condition. (c) Map of valid disparity values for integrated disparity map.

2.1 Extended dynamic range stereo

Variable gain (or equivalently, variable aperture or extended dynamic range) imaging has been developed for intensity model acquisition with wide dynamic range [6, 15], but has not to our knowledge been applied to stereo range estimation. This particularly benefits disparity estimation, since different regions of a scene likely will have maximal contrast at different camera gain settings. With any single gain setting, a large portion of the image may be (de-)saturated, yielding unknown depth values.

Correspondence matching with extended range intensity images is straightforward–one can simply apply traditional matching algorithms to floating point pixel intensity data. However, acquiring extended range intensity images from a conventional camera requires a relatively precise photometric calibration of each camera. If photometric calibration is not available, we can approximate the result by separately computing disparity at each gain setting, and integrating the results over multiple observations.

Our basic stereo engine is based on classic window-based robust correspondence. If $I^L_{g,t}(\boldsymbol{x})$ and $I^R_{g,t}(\boldsymbol{x})$ are the rectified left and right intensity images acquired at time t and gain level g, indexed by pixel location $\boldsymbol{x} = [x\ y]^T$, we denote the quality of a disparity match using a match function

$$D(I^L_{g,t}, I^R_{g,t}, \boldsymbol{x}, d) =$$
$$-\sum_{\boldsymbol{n}\in\mathcal{N}} ||T_{\boldsymbol{x},\boldsymbol{n}}(I^L_{g,t}(\boldsymbol{x}+\boldsymbol{n}) - T_{\boldsymbol{x},\boldsymbol{n}}(I^R_{g,t}(\boldsymbol{x}+\boldsymbol{n}+[d\ 0]^T))||_r$$

for some suitable neighborhood $\mathcal{N}$ and robust transform T (e.g., [19]) and/or robust norm $||\cdot||_r$ (e.g., [4]).

At each pixel, we can apply the standard technique of evaluating D across all disparities, testing whether these values can be approximated well by a single mode. If so we set the match value $\hat{d}(\boldsymbol{x}, g, t)$ to be the disparity with the highest match, and $\hat{D}(\boldsymbol{x}, g, t)$ to be the match value. When the distribution is not approximated by a single mode of sufficient height, we declare the pixel invalid $\hat{d}(\boldsymbol{x}, g, t) = null$. We compute a global estimate of σ_D, the standard deviation of $\hat{D}$, based on $\hat{D}$ values when the same $\hat{d}$ is chosen in two consecutive frames.

2.2 Long term observation

To acquire background models online without an training sequence of foreground-free observations, we use a variant of the algorithms presented in [7] and [9]. During background training, we incrementally compute a histogram of disparity values across a range of time and gain conditions. Given a new range observation $\hat{d}(\boldsymbol{x}, g, t)$, we increment a histogram $H_B(\boldsymbol{x}, \hat{d}(\boldsymbol{x}, g, t))$. After a period of background training, we determine the background disparity values by inspecting H_B to find modes which are large and far. We note that choosing the farthest modes helps to reduce problems of disparity ambiguity in periodic textures.

We choose the background values to be those which are less than the median valid disparity over time, $d_B(\boldsymbol{x})$.[1] We then set a background weight map W_B to have the same values as H_B, except that any values greater than d_B are set to zero. We require that the ratio of valid to invalid range observations at a pixel be sufficiently large to keep a background value. If this ratio is not greater than a threshold (typically $\beta = 0.1$), then we set the background to be undefined, $d_B(\boldsymbol{x}) = null$, and W_B, H_B to be uniformly zero.

---

[1] For scenes where the background is covered more than half the time, we could use a rank filter with lower threshold.

Gathering background observations over long-term sequences has the advantage that lighting variation can be included in the background training set. Extreme lighting variation is useful, since it can cause previously uniform regions to have apparent contrast. Either the overall (ambient) or relative (shadow or projected texture) illumination level can cause contrast to appear where previously the image region was uniform. As with variable gain, each illumination condition will likely yield a different pattern of valid/invalid range.

We generally choose to observe the natural variation of illumination, e.g., from natural outdoor light entering through windows, and from users' regular actuation of indoor lighting appliances. However, it is also possible to specifically train the background using active illumination if desired. During a background training period we also set the camera parameters to cycle through the range of possible gain settings. Figure 2(a) shows a set of images taken under different gain and illumination conditions, and Figure 2(b) shows the map of valid range pixels computed from stereo pairs in each of these conditions. Figure 2(c) shows the map of valid disparity values for an integrated background recovered from the set of variable gain and illumination training data.

2.3 Detection of unoccluded uniform planar regions

In indoor environments there are many planar uniform regions on which disparity is difficult to obtain, even with long-term, variable-gain observations. In general, determining whether to interpolate range values within such surfaces without inferring global structure in the scene is difficult.

However, there is one special case that is locally computable and proves to be particularly useful for our purposes. When a planar uniform region is unoccluded in two views, the extent of the homogeneous patch in each view will be equivalent, according to the homography determined by the stereo viewing geometry and the orientation of the patch in the world. This condition often happens in practice–for example in saturated pixels in dark or light clothing worn by a person in the scene (e.g., Figure 3).

If the correspondences at the border of the region are well approximated by a homography and are consistent with the epipolar stereo constraint, we can test to see whether the shape of the homogeneous patch in each view is related by the appropriate warping function. We can compute a boolean image indicating the extent of the uniform region in each image, warp one according to the given homography, and compare the overlap of the resulting images. If they agree, then we mark this to be a uniform solid plane hypothesis.

There are two degenerate cases: a region from a window looking out to a completely featureless background, and a solid but non-planar region with no apparent texture. By maintaining the hypothesis throughout an extended time period, we can alleviate the impact of the former and only keep the hypothesis if no object is ever seen further in depth at any point within the region. The latter degeneracy seems unavoidable, but has not been a problem in practice.

Our method is related to the classic idea of region stereo, however we do not attempt to perform a dense color segmentation of the image as a pre-processing step, and only opportunistically apply the technique when there are large uniform connected components in the scene. We also make explicit use of planar homographies and the epipolar constraint to constrain possible matches.

2.4 Fast foreground disparity estimation

Fast foreground detection is necessary to detect rapidly moving objects and dynamic activity patterns. Traditional approaches to tracking with stereo-based backgrounds usually have relied on general-purpose stereo computation, which exhaustively searches for the best disparity matches at each frame. However most of this computation is unnecessary for scenes with predictable backgrounds, as pointed out by [11]. They demonstrated how disparity testing could find foreground silhouettes, given a previously computed static background model. We have extended their method to the case of dynamic backgrounds with multiple range modes, and to predict the entire range image, including disparities in foreground regions.

We use our background disparity weights, W_B together with similar weights corresponding to short-term foreground predictions, W_F, to guide the disparity search in a new frame. For each new range match we increment W_F as we increment W_B above, but only after we reduce the previous values of W_F by a constant factor (typically $\gamma = 0.9$):

$$W_F(\boldsymbol{x}, d) = \gamma W_F(\boldsymbol{x}, d) + \delta(d - \hat{d}(\boldsymbol{x}, g, t))$$

where $\delta(x)$ is the impulse function at $x = 0$.

We maintain a separate W_B and W_F for each pixel in the image. Given a new frame at gain g and time t we find those (d_i^*, w_i^*) with sufficiently large value:

$$||D(I_{g,t}^L, I_{g,t}^R, \boldsymbol{x}, d_i^*) - \hat{D}(\boldsymbol{x}, g, t)||^2 \ <? \ \rho\sigma_D^2$$

Fig. 3. Disparity estimation with uniform planar unoccluded regions. We match a candidate homography between connected components of uniform regions in two views. Unoccluded planes will yield connected component shapes that exactly match according to the given homography. Occluded planes, such as the computer monitor under the user's arm in the far background, or non-planar objects, such as the plant in the foreground, will not have equivalent shapes and will not be matched. In this example uniform regions were determined by saturated pixels, and the plane was restricted to be fronto-parallel so that image plane translation would determine equivalent matches. (a) shows a stereo pair, (b) saturated pixels in each view, (c) connected components that exactly match in both views for some translation, (d) computed disparity values for each region, shown in grayscale.

and whose background or foreground weight is also above a threshold, typically $\alpha = 0.25$. (Usually $\rho = 2$.) If no such candidate is confirmed, we compute all disparity values and estimate $\hat{d}$ using the conventional approach described above. If the selected disparity is less than $d_B(\boldsymbol{x})$, we label it foreground. Points for which $d_B = null$ or $\hat{d} = null$ are by definition not foreground. During a foreground detection phase we have the option of using the automatic gain control setting, or searching through the range of gain levels. If run-time speed is the primary concern, we choose the former approach.

This search pruning optimization can dramatically reduce run-time costs when the foreground regions of a scene are relatively small or are moving slowly. The final result of range-based foreground detection is a map of foreground pixels $\boldsymbol{p}_j = (c_j, x_j, y_j, t_j, d_j)$, each from a particular location x, y in camera c at time t with disparity d.

3 Plan-view trajectory estimation

We combine information from multiple stereo views to estimate the trajectory of objects over time. The true extent of an individual object in a given image is generally difficult to identify. An optimal trajectory segmentation ought to consider the assignment of an individual pixel to all possible trajectories estimated over time. Systems which perform an early segmentation and grouping of foreground data before trajectory estimation preclude this possibility.

We adopt a late-segmentation strategy, finding the best trajectory in an integrated spatio-temporal representation that combines foreground pixels from each view. By assuming that objects move on a ground plane and do not overlap in the vertical dimension in our environment, a “plan-view assumption” allows us to completely model instantaneous foreground information as a 2-D orthographic density projection. Over time, we compute a 3-D spatio-temporal plan-view volume.

We project (x_j, y_j, d_j) from each foreground point $\boldsymbol{p}_j$ into world coordinates (U_j, V_j, W_j) using the calibration given by camera c_j. (See Figure 4.) U, V are chosen to be orthogonal axes on the ground plane, and W normal to the ground plane. We then compute the spatio-temporal plan view volume, with $P(u, v, t) = \sum_{\{\boldsymbol{p}_j | U_j = u, V_j = v, t_j = t\}} 1$.

3.1 Distance transform-based dynamic programming

We characterize the quality of a trajectory by its smoothness over time, and the support of the object track in each time frame. Given $P(u, v, t)$

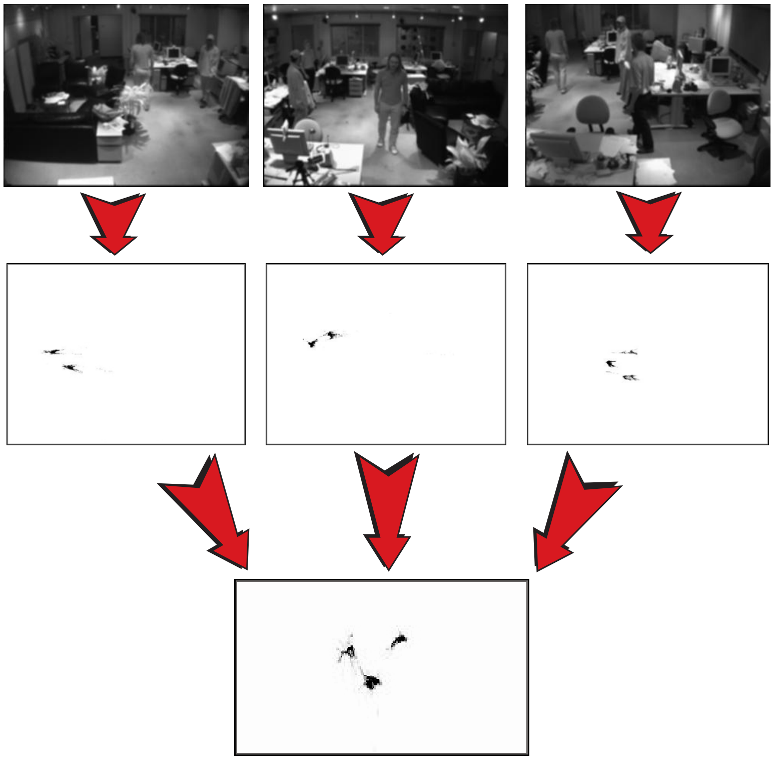


Fig. 4. Foreground points are projected from each view individually to a plan view representation, then are integrated into a single spatio-temporal sequence before segmentation into individual trajectories is performed.

and a set of possible poses and positions of the object at each time step, we characterize a single optimal trajectory over all time as,

$$L^* = \arg\max_L \sum_{0<t\leq T} M(\boldsymbol{l}_t) - \sum_{0<t<T} d(\boldsymbol{l}_t, \boldsymbol{l}_{t+1}) \tag{1}$$

where $\boldsymbol{l}_t$ is one of the possible discrete 2-D location, size and pose parameters of the object in frame t, $L = \{\boldsymbol{l}_t \mid 0 < t < T\}$ is the object trajectory, $M(\boldsymbol{l}_t)$ is the support of the object track at location $\boldsymbol{l}_t$, and $d(\boldsymbol{l}_t, \boldsymbol{l}_{t+1})$ is cost of matching $\boldsymbol{l}_t$ with $\boldsymbol{l}_{t+1}$. We compute support to be the integral of the plan-view density within the shape given by $\mathcal{S}(\boldsymbol{l}_t)$:

$$M(\boldsymbol{l}_t) = \sum_{(x,y)\in\mathcal{S}(\boldsymbol{l}_t)} P(x, y, t)$$

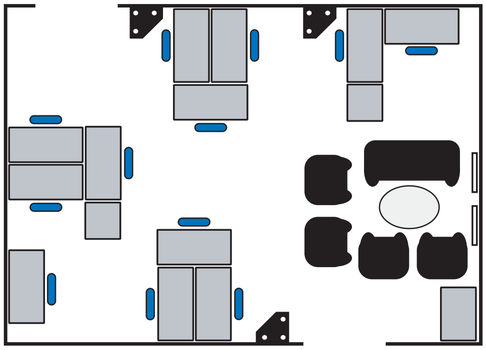


Fig. 5. Plan view diagram of office environment used for example in previous figure. Workstations are shown as rectangles, and a sitting area is shown as chairs and couch around an oval table. Three trinocular stereo cameras, shown as black triangles with three circles, are drawn at the approximate location where they are mounted in the ceiling.

Classically, it is possible to solve equation (1) with complexity $O(m^2T)$ using dynamic programming techniques [1]. Unfortunately, m is the number of discrete configurations, so m^2 can grow prohibitively large. Using the distance transform formulation introduced in [8], we can reduce this complexity to $O(mT)$, by restricting the form of d to be the norm or norm squared of transformed location values. We simply set $d(\boldsymbol{l}_t, \boldsymbol{l}_{t+1}) = (\boldsymbol{l}_t - \boldsymbol{l}_{t+1})^T \mathbf{V}^{-1} (\boldsymbol{l}_t - \boldsymbol{l}_{t+1})$, where $\mathbf{V}$ is a diagonal matrix

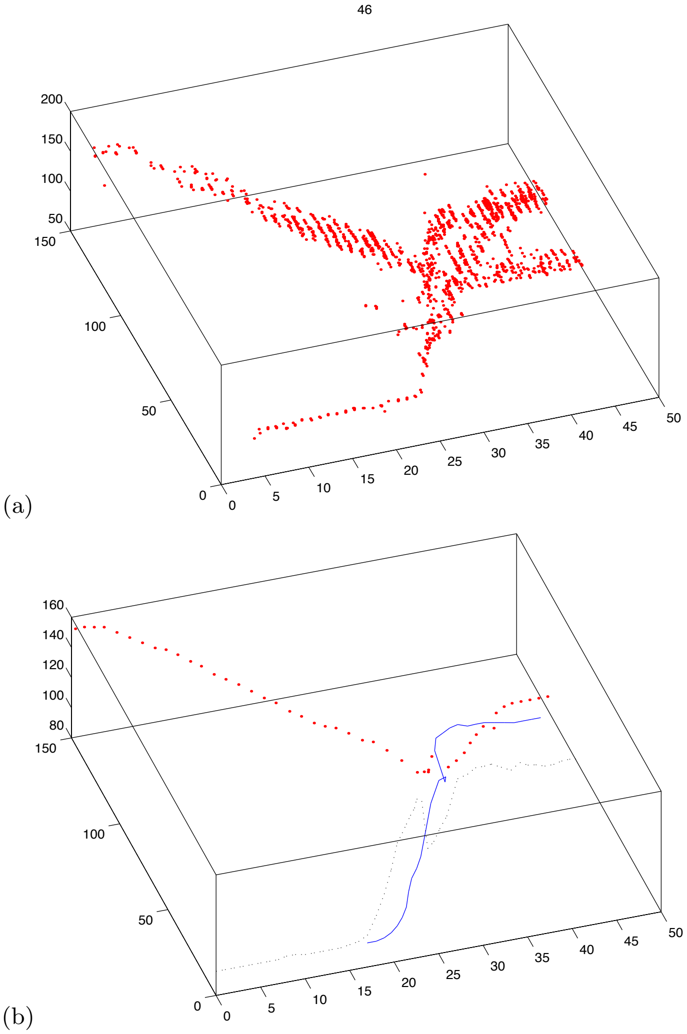


Fig. 6. (a) (U,V,T) plot of plan-view foreground density over time for a sequence with three moving people. The third person enters around frame 20. (b) Segmentation using dynamic programming method described in text.

of variances for each pose parameter. This simply says that each pose parameter should change slowly over time.

To solve equation (1), we first compute the best value of the final $\boldsymbol{l}_T$, as a function of the location at the previous time using dynamic programming:

$$B_T(\boldsymbol{l}_{T-1}) = \max_{\boldsymbol{l}_T}(M(\boldsymbol{l}_T) - d(\boldsymbol{l}_{T-1}, \boldsymbol{l}_T))$$

Recursively, we then compute the best value of $\boldsymbol{l}_t$ as a function of $\boldsymbol{l}_{t-1}$

$$B_t(\boldsymbol{l}_{t-1}) = \max_{\boldsymbol{l}_t}(M(\boldsymbol{l}_T) - d(\boldsymbol{l}_{t-1}, \boldsymbol{l}_t) + B_{t+1}(\boldsymbol{l}_t))$$

and finally $B_0 = \max_{\boldsymbol{l}_0}(M(\boldsymbol{l}_0) + B_1(\boldsymbol{l}_0))$. The optimal trajectory is then given by replacing max with arg max in the above equations and reversing the recursion to compute the optimal location at each time, $\boldsymbol{l}_t^*$. This method finds a single optimal trajectory from the initial time 0 to the final time T.

3.2 Trajectory start/end determination

A difficult challenge in person tracking systems has been the estimation of the number of people a given environment under general entry and exit conditions. Previous systems [13] have relied on specific spatial zones to delimit the start and end of person trajectories, while other systems generally use instantaneous criteria to initiate or terminate a new track. We take advantage of our spatio-temporal plan-view representation to optimally estimate trajectory extent as well as shape in a single dynamic programming optimization.

We extend the above method to find trajectories with explicit start and end points. Let

$$(L^*, s^*, e^*) = \arg\max_{L,s,e} \sum_{s \le t \le e} (M(\boldsymbol{l}_t) - \psi) + \sum_{s \le t < e} d(\boldsymbol{l}_t, \boldsymbol{l}_{t+1}) \qquad (2)$$

where ψ is the cost of extending (s^*, e^*) per time unit.

Fortunately, we can solve equation 2 by modifying the recursive algorithm above. First we replace $M(l)$ with $(M(l) - \psi)$ in the equations above. After computing each B_t, we inspect each location to check if $B_t(\boldsymbol{l}_{t-1})$ is negative. If so, we mark that location, indicating that if the object goes through $\boldsymbol{l}_{t-1}$, that should be the end of the trajectory. The start of the trajectory is given by the location maximizing $B_s(\boldsymbol{l}_s)$, over all locations and time frames. The trajectory can be found by tracing back from $\boldsymbol{l}^*_{s^*}$ until reaching a location marked as a trajectory end. The estimation of L^*, s^*, e^* for a single trajectory is optimal in that the dynamic programming method computes a global maxima of equation 2.

3.3 Implementation and examples

Figure 5 shows the configuration of cameras in our test environment. To align multiple views, we expect to use an automatic calibration process where objects moving on the plane are used to determine the orientation of each camera view [14, 13]. However, this section's results were obtained with an approximate manual calibration based on hand selected correspondences in each camera view.

We collect variable gain and illumination images in our environment during a background acquisition phase. When there are multiple objects in the scene, we solve for a set of trajectories by first finding the trajectory with highest quality given by Equation 2, removing the points that contributed to its support in the plan-view sequence, and repeating until no further trajectory can be found with positive quality.

For the examples shown in this section, we used a simplified shape model of rectangular regions with fixed size and pose, given by average human torso dimensions. We expect to extend our implementation shortly to include varying size and pose, which will allowing extended arm position tracking, etc. We discretized ground plane position to a $320x240$ grid, and set ψ to be the median value of $\hat{d}(l)$ evaluated at random locations. (Again, assuming that our scene is on average more than half background.) We truncate the time history at 50 frames. Figure 6 shows the result on a sequence of 3 people moving within our test environment.

4 Discussion and future work

While the results are appealing, a problem remains: when the trajectory of two objects or people overlap, it is not possible from a foreground density representation to disambiguate trajectories if they subsequently separate. Appearance information can resolve this, as shown by [10]. Unfortunately, including this in the dynamic programming optimization would greatly increase the size of the state space of locations at each time frame, making the solution for the optimal trajectory impractical. Resolving this is a topic of ongoing and future work. We plan to use a trajectory-level correspondence process that uses a graph based on the overall trajectory data, computes aggregate appearance information along each edge (e.g., using color histograms), and then matches these features to resolve identity along each edge.

Our system currently uses an iterative approach to estimating multiple trajectories, and for each person solves a batch trajectory estimation problem. This is clearly impractical for real-time, on-line use. To overcome this limitation we are developing an incremental version of the algorithm that maintains an more compact representation of prior trajectory state.

Finally, we have left open the issue of what schedule of gain settings to use when building a background model and when detecting foreground points. A topic of future work is to determine the minimum number of samples needed to obtain a maximally dense integrated range image for a given scene or scene class.

5 Conclusions

This paper presents new algorithms which make tracking objects in widely varying illumination conditions possible. There are two main contributions presented.

First, we formulate stereo range estimation using extended dynamic-range imagery and show how a dense background model can be built with long-term observations. Without this, stereo range data is too sparse to construct a useful background model for tracking. We derive a constraint on the projection of planar uniform surfaces in stereo views and use this to interpolate range within such regions. We implement our scheme with a predictive, ordered disparity search technique, that prunes most of the computation typically required to process new images.

Second, we developed an optimal method for estimating trajectories without performing an initial segmentation of foreground points. Foreground points from multiple views are projected into a plan-view density representation, and are segmented into object regions as part of a globally optimal dynamic programming optimization. Estimation of trajectory extent (object entry/exit) is included in the global optimization, and does not require any spatial constraints. We demonstrated our prototype system estimating the trajectory of several people moving in an environment.

References

1. A. Amini, T. Weymouth, and R. Jain. Using dynamic programming for solving variational problems in vision. *PAMI*, 12(9):855–867, September 1990.
2. D.J Beymer. Person counting using stereo. In *Workshop on Human Motion, Austin, Texas*, 2000.
3. D.J. Beymer and K. Konolige. Real-time tracking of multiple people using stereo. In *Workshop on Frame-Rate Applications, ICCV'99*, 1999.
4. M.J. Black and P. Anandan. A framework for the robust estimation of optical flow. In *ICCV93*, pages 231–236, 1996.
5. T. Darrell, G.G. Gordon, M. Harville, and J. Woodfill. Integrated person tracking using stereo, color, and pattern detection. *IJCV*, 37(2):175–185, June 2000.
6. Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. *Proceedings of SIGGRAPH 97*, pages 369–378, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.
7. C. Eveland, K. Konolige, and R.C. Bolles. Background modeling for segmentation of video-rate stereo sequences. In *CVPR98*, pages 266–272, 1998.
8. P.F. Felzenszwalb and D.P. Huttenlocher. Efficient matching of pictorial structures. In *CVPR00*, pages II:66–73, 2000.
9. G.G. Gordon, T. Darrell, M. Harville, and J. Woodfill. Background estimation and removal based on range and color. In *CVPR99*, pages II:459–464, 1999.
10. I. Haritaoglu, D. Harwood, and L.S. Davis. W4: Real-time surveillance of people and their activities. *PAMI*, 22(8):809–830, August 2000.
11. Y.A. Ivanov, A.F. Bobick, and J. Liu. Fast lighting independent background subtraction. *IJCV*, 37(2):199–207, June 2000.
12. P. Kornprobst and G. Medioni. Tracking segmented objects using tensor voting. In *CVPR00*, pages II:118–125, 2000.
13. J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera multi-person tracking for easyliving. In *3rd IEEE Workshop on Visual Surveillance (Dublin,Ireland)*, 2000.
14. L. Lee, R. Romano, and G. Stein. Monitoring activities from multiple video streams: Establishing a common coordinate frame. *PAMI*, 22(8):758–767, August 2000.
15. S.K. Nayar and T. Mitsunaga. High dynamic range imaging: Spatially varying pixel exposures. In *CVPR00*, pages I:472–479, 2000.
16. C. Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. In *CVPR99*, pages II:246–252, 1999.
17. M. Trivedi, S. Mikic, and S. Bhonsle. Active camera networks and semantic event databases for intelligent environments. In *IEEE Workshop on Human Modeling, Analysis and Synthesis*, 2000.
18. C.R. Wren, A. Azarbayejani, T.J. Darrell, and A.P. Pentland. Pfinder: Real-time tracking of the human body. *PAMI*, 19(7):780–785, July 1997.

19. R. Zahib and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proc. ECCV*, 1994.

Proceedings of the IEEE Workshop on
Stereo and Multi-Baseline Vision
(SMVB 2001), to appear.

Range Segmentation Using Visibility Constraints

Leonid Taycher and Trevor Darrell

MIT Artificial Intelligence Laboratory
{`lodrion, trevor`}`@ai.mit.edu`

Abstract. Visibility constraints can aid the segmentation of foreground objects observed with multiple range images. In our approach, points are defined as foreground if they can be determined to occlude some *empty space* in the scene. We present an efficient algorithm to estimate foreground points in each range view using explicit epipolar search. In cases where the background pattern is stationary, we show how visibility constraints from other views can generate virtual background values at points with no valid depth in the primary view. We demonstrate the performance of both algorithms for detecting people in indoor office environments.

1 Introduction

Object segmentation is an important preliminary step for many high-level vision tasks, including person detection and tracking. State-of-the-art systems [14, 2, 1, 5] use foreground/background classification followed by pixel clustering and analysis. These systems commonly maintain a background model and label all pixels that differ significantly from this model as foreground.

Ideally, these systems should be robust to rapid illumination variation, such as from outdoor weather or indoor video projection systems. Several segmentation methods have been proposed which use background models based on color/intensity [14, 13, 12], stereo range [7, 1] or both [6]. Generally, non-adaptive color-based models suffer from varying illumination. Adaptive color models [12] are more stable under lighting changes, but can erroneously incorporate objects that stop moving into the background model. Range-based background models can be illumination invariant, but are usually sparse. To avoid ambiguity at undefined background values (and the resulting illumination dependence [3]), they have been either used in conjunction with color models [6], or are built using observations from widely varying illumination and imaging conditions [3]. In this paper we show how visibility constraints from other range images can aid segmentation.

Our approach to foreground segmentation is to combine *free space* constraints found from multiple stereo range views. We decide if a given pixel is "foreground" by checking whether there is any *free space* behind it, as seen from other range views. We scan the set of epipolar lines in the other views corresponding to the given pixel, and test whether there are range points on the epipolar lines which indicate empty space behind the given point.

This is a similar computation to algorithms proposed for the rendering of image-based visual hulls [9]. The key difference is that our method takes as input unsegmented noisy range data and evaluates 3-D visibility per ray, while the visual hull method presumes segmented color images as input and simply identifies non-empty pixels along the epipolar lines in other views. Also related are space carving and coloring methods [8, 11], which split the space into *voxels* and use color consistency across multiple cameras to locate opaque voxels and to detect *free space*. These methods are quite general, and work with an arbitrary set of monocular views. They also require the construction of a volumetric representation of the scene for reconstruction or segmentation. We are interested in algorithms that perform segmentation solely in the image domain, without computing a volumetric reconstruction. We believe ours is the first method for range image segmentation using image-based (non-voxel) freespace computation.

In this paper we develop two complimentary segmentation algorithms that use visibility constraints. The first is an instantaneous foreground detection algorithm, which is independent of previous time points and does not presume scene or illumination constancy. The second assumes a stationary scene and a background range model per view, and generates virtual background values at pixels which would otherwise have had insufficient contrast to have valid range.

In the next section we describe our method for using "complimentary" camera(s) to determine whether a single 3-D point, visible by a "primary" camera, occludes any *free space*. We describe how to cluster such points, and to determine whether two clusters provably belong to separate objects. We then propose a method for creating dense virtual backgrounds for stationary scenes. Finally, we demonstrate the results using our algorithm tracking people in an indoor office environment.

2 Foreground segmentation

We wish to segment objects that are not attached to any "background" surface other than the floor, by detecting pixels that occlude some *empty space*.

When a 3D point $\mathbf{P}$ is imaged by a range sensing device (e.g. stereo camera) C^1 (Figure 1), we know that there exists a nontransparent material at that point, and that all points between it and the camera's center of projection are transparent. But C^1 is unable to provide any information about what lies behind $\mathbf{P}$ on the same projection ray. On the other hand, we can use the observation $\mathbf{B}$ of the appropriately located range sensor C^2 to discover that $\overline{\mathbf{P}}$, which is occluded by $\mathbf{P}$ in C^1's view is transparent.

An ideal (rectified) stereo rig may be completely described by the baseline B, focal length f and the image coordinates of the principal point (c_x, c_y). The following equations describe a relation between point (x, y) in the disparity image I_D and the corresponding 3-D location (X, Y, Z).

$$\begin{cases} \overline{x} = x - c_x = f\dfrac{X}{Z} \\ \overline{y} = y - c_y = f\dfrac{Y}{Z} \\ d = I_D(x, y) = f\dfrac{B}{Z} \end{cases} \tag{1}$$

As has been shown in [4], the disparity space is a projective space, and we can write the transformation from disparity to camera-centered Euclidean projective coordinates as

$$\begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = \mathbf{T_C}(f, B, c_x, c_y) \begin{pmatrix} x \\ y \\ d \\ w \end{pmatrix} \tag{2}$$

where

$$\mathbf{T_C}(f, B, c_x, c_y) = \begin{pmatrix} B & 0 & 0 & -c_xB \\ 0 & B & 0 & -c_yB \\ 0 & 0 & 0 & fB \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3}$$

In the rest of the paper we refer to $(x\ y\ d)^T$ and $(X\ Y\ Z)^T$ ($(x\ y\ d\ w)^T$ and $(X\ Y\ Z\ W)^T$) as disparity and Euclidean (projective) spaces, and denote them D and E respectively.

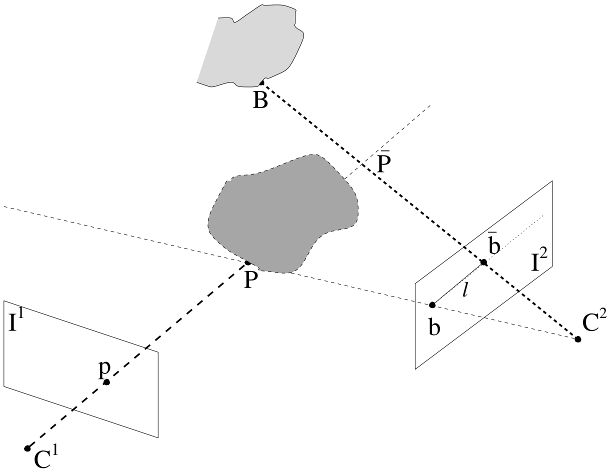


Fig. 1. Visibility-based segmentation. Observation of **B** in C^2 allows us to infer that **P** in C^1 is foreground. Point **B** visible in I^2 (projecting to C^2 disparity point $\overline{\mathbf{b}}$) lies behind point $\overline{\mathbf{P}}$ relative to C^2, and thus provides evidence for existence of *free space* behind **P** (projecting to **b**) by demonstrating that $\overline{\mathbf{P}}$ is transparent. Line l contains the oversilhouette for the part of an object lying along ray $[\mathbf{C^1}, \mathbf{P})$

The general setup of the imaging system assumed in our algorithm is presented in the Figure 1. There are two calibrated stereo rigs, C^1 ("primary") and C^2 ("complimentary"), with disparity-to-Euclidean camera coordinate transforms $\mathbf{T}_{\mathbf{D^1}}^{\mathbf{E^1}} = \mathbf{T_C}(f^1, B^1, c_x^1, c_y^1)$ and $\mathbf{T}_{\mathbf{D^2}}^{\mathbf{E^2}} = \mathbf{T_C}(f^2, B^2, c_x^2, c_y^2)$, image planes I^1 and I^2, and disparity images I_D^1 and I_D^2 respectively. The Euclidean coordinate transform between views is $\mathbf{T}_{\mathbf{E^1}}^{\mathbf{E^2}}$.

2.1 Pixel-level segmentation

The first stage of our foreground segmentation algorithm applied to the disparity image of the camera C^i is to determine which 3-D points visible by C^i occlude some *free space*. That is, for each disparity point $\mathbf{p} = (p_x \; p_y \; I_D^i(p_x, p_y) \; 1)^T$, where $I_D^i(p_x, p_y)$ is valid, we check whether there is some *free space behind* the point **P**,

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathbf{P} \simeq \mathbf{T}_{\mathbf{D^i}}^{\mathbf{E^i}} \mathbf{p} \tag{4}$$

where $\simeq$ denotes equality up to a scale factor.

We say that point $\overline{\mathbf{P}}$ is *behind* point $\mathbf{P}$ relative to rig C^i, if $\overline{\mathbf{P}}$ lies on the optical ray $[\mathbf{C^i}, \mathbf{P})$, and $|\overline{\mathbf{P}} - \mathbf{C^i}| > |\mathbf{P} - \mathbf{C^i}|$.

Let point $\overline{\mathbf{P}}$ be *behind* $\mathbf{P}$ relative to camera C^i. We project $\overline{\mathbf{P}}$ into $D^j, i \neq j$ to obtain point $\overline{\mathbf{b}}$, such that

$$\begin{pmatrix} \overline{b}_x \\ \overline{b}_y \\ \overline{b}_d \\ 1 \end{pmatrix} = \overline{\mathbf{b}} \simeq (\mathbf{T}^{\mathbf{E^j}}_{\mathbf{D^j}})^{-1} \mathbf{T}^{\mathbf{E^j}}_{\mathbf{E^i}} \overline{\mathbf{P}} \tag{5}$$

The points $(\overline{b}_x, \overline{b}_y)$, corresponding to all possible $\overline{\mathbf{P}}$s, form a ray of the line epipolar to (p_x, p_y) in I^j (passing through projection of the $\mathbf{C^i}$ to I^j and $\mathbf{b}$).

If the disparity value $I^j_D(\overline{b}_x, \overline{b}_y)$ corresponding to any of such $\overline{\mathbf{b}}$s is *valid* and is *smaller* than $\overline{b}_d$ (i.e. some point $\mathbf{B}$ *behind* $\overline{\mathbf{P}}$ relative to camera C^j is visible by C^j), then point $\overline{\mathbf{P}}$ is transparent, and may be assumed to belong to *free space.* When we can find cases where

$$I^j_D(\overline{b}_x, \overline{b}_y) < \overline{b}_d \tag{6}$$

we consider that point to be *evidence* for $\mathbf{P}$ (correspondingly $\mathbf{p}$) belonging to foreground.

In the current implementation of the algorithm, we use the number of found *evidence* pixels as a measure of certainty that point $\mathbf{p}$ belongs to foreground. If more than one "complimentary" camera is available, then the results from each of them may be combined to provide more robust output. We compute a map of the number of observed occluded free-space points:

$$\begin{aligned} \theta(\overline{\mathbf{b}}) &= \begin{cases} 1 & I^j_D(\overline{b}_x, \overline{b}_y) \text{ is valid and } \lambda I^j_D(\overline{b}_x, \overline{b}_y) < \overline{b}_d \\ 0 & \text{otherwise} \end{cases} \\ OFS(\mathbf{p}) &= \sum_{\substack{\overline{\mathbf{b}}, \\ \text{for all } \overline{\mathbf{P}}}} \theta(\overline{\mathbf{b}}) \end{aligned} \tag{7}$$

Where the factor $\lambda > 1$ is introduced to deal with noise inherent in disparity computation. Since we expect the stereo-based range to be less robust for locations that are far from the camera, we can classify $\mathbf{p}$ as foreground if $OFS(\mathbf{p}) > T_{OFS}(p_d)$, where $T_{OFS}(d) \sim 1/d$.

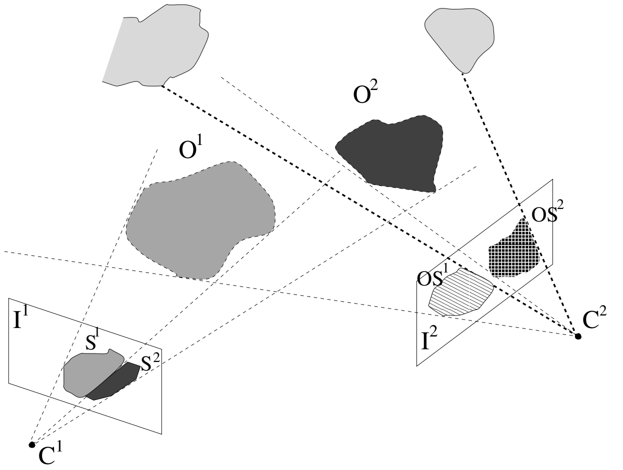


Fig. 2. Segmenting multiple objects. The silhouettes S^1 and S^2 of objects O^1 and O^2 are adjacent in I^1, but the I^2 oversilhouettes, OS^1 and OS^2, which are computed from S^1 and S^2 (and the first freespace points found behind them) do not overlap, so we may conclude that O^1 and O^2 are indeed separate objects.

2.2 Range cluster generation

The method described in the previous section provides us with a measure of how much *free space* is occluded by each pixel in a given view. We use this information to estimate the extent and connectivity of foreground regions in each view, and then link regions across views based on their projected overlap. Individual pixels are first clustered in each view, and we then determine whether two clusters belong to separate objects (Figure 2).

A naive approach would be to cluster the points based on proximity in either disparity or Euclidean space, and assume that each such cluster corresponds to a separate object. Such assumptions are correct in cases such as one in Figure 3(a), but lead to oversegmentation in the example in Figure 3(b), where components S^1 and S^2 actually correspond to parts of the same object. To resolve this ambiguity, we use the visibility information computed for each pixel (Section 2.1).

In Figure 2 we cannot separate O^1 and O^2 using only information from I^1 (S^1 and S^2). If the actual silhouettes of the objects in the second image were available, we could see that in fact they are non-overlapping, and thus objects are unconnected. In practice we do not have a set of complete silhouettes when range data is sparse. Instead, we compute an approximation of the silhouette in I^2 using the *free space* visibility constraints found for I^1. We define an "oversilhouette" to be the projection into I^2 of 3-D line segments formed by observed points P from I^1 and the first confirmed freespace point they occlude. If such "oversilhouettes" of two components do not overlap in I^2, then we conclude these components belong to different objects (Figure 2).

If the components' "oversilhouettes" overlap, we assume that components correspond to parts of the same object (Figure 3(b)). This can lead to undersegmentation (e.g. Figure 3(c)) if there are insufficient views to observe the segmentation between disjoint objects. With additional cameras this could be resolved as shown in Figure 3(d).

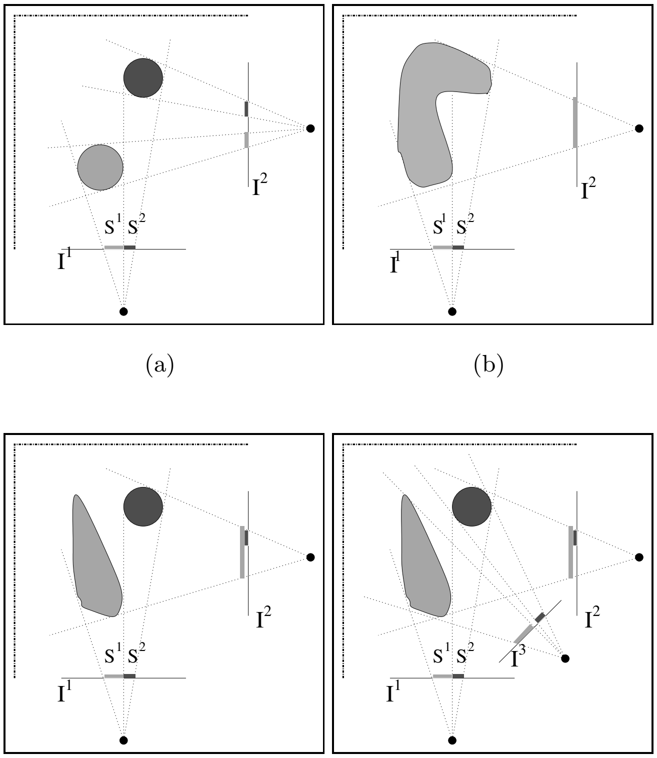
3 Virtual background generation

While the algorithm described in the previous section is capable of semi real-time performance (2fps on full resolution images) on current hardware, our tracking applications require much faster segmentation algorithms. The common range background subtraction algorithms provide high-speed performance, but are unreliable in the absence of the dense range data [3, 1]. While some improvement may be obtained using statistical training, the range images obtained in the indoor environment would generally be sparse (Figures 4(d), 5(d)). In this section we describe a method for generating dense virtual background images.

When the common range background subtraction methods are used, each pixel in the "background" image represents an upper limit on the depth (lower limit on the disparity) of *free space* visible along the corresponding optical ray when no foreground objects are present. Such upper limit may be obtained by, for example, taking the minimum of the observed valid disparity values at the pixel over time [3].

If no range data is available at the point, we can estimate this limit from visibility constraints obtained from "complimentary" cameras. For each point in I^i with invalid range we search the corresponding optical ray to detect all *free space* points along it that are visible by other cameras C^js, and select the one with the greatest depth as the virtual background.

In order to simplify the algorithm we inverse the order of computation. Instead of searching along the optical rays of C^i, we compute all *free space* points visible by C^j, $i \neq j$, using the computation described in the next section.



(a) (b)

(c) (d)

Fig. 3. Connected components configurations. (a) the components S^1, and S^2 belong to separate objects, as the oversilhouettes computed from them in I^2 do not overlap. (b) the components belong to the same object. (c) The algorithm undersegments the scene, assuming that S^1 and S^2 belong to the same object, since their "oversilhouettes" overlap in I^2, but the same scene may be correctly segmented (d) if an extra view (I^3) is available.

For each valid range point $\mathbf{p} = (p_x \; p_y \; I_D^j(p_x, p_y) \; 1)^T$, all points on the optical ray between $\mathbf{P} \simeq \mathbf{T}_{\mathbf{D^j}}^{\mathbf{E^j}}\mathbf{p}$ and $\mathbf{C^j}$ are transparent and may be assumed to belong to *free space*. Thus any point $\overline{\mathbf{P}} \in (\mathbf{P}, \mathbf{C^j})$ is a candidate virtual background for the corresponding point in I^i, $(\overline{b}_x, \overline{b}_y)$, such that

$$\begin{pmatrix} \overline{b}_x \\ \overline{b}_y \\ \overline{b}_d \\ 1 \end{pmatrix} = \overline{\mathbf{b}} \simeq (\mathbf{T}_{\mathbf{D^i}}^{\mathbf{E^i}})^{-1}\mathbf{T}_{\mathbf{E^j}}^{\mathbf{E^i}}\overline{\mathbf{P}} \tag{8}$$

After a set of candidates for a single (discrete) image location $(\{\tilde{\mathbf{b}}_i = (\tilde{b}_x \; \tilde{b}_y \; \tilde{b}_{d_i} \; 1)^T\})$ is computed, we select

$$v = \min_i \tilde{b}_{d_i} \tag{9}$$

as the virtual background value at the location $(\tilde{b}_x, \tilde{b}_y)$.

If both virtual and statistically trained background images are available, we may combine them to increase robustness by using data from statistically trained background when it is available (as it represents the true limit), and using virtual background data otherwise (Figure 7(a-f)).

4 Experimental results

The foreground segmentation algorithm as implemented currently consists of several parts. The first part performs per-pixel computation described in Section 2.1 with $\lambda = 1.05$. We then cluster the pixels using techniques from Section 2.2, and finally pass the connected components through a size filter (accepting components greater than 1% of the image). The algorithm, running on 700MHz Pentium III, achieved the performance of 2 frames per second on the full resolution images.

To test our algorithms we used an installation with two Point Grey Digiclops cameras [10]. One camera used 6mm lenses, and another had 3.8mm lenses (wide-angle) (Figures 4 and 5), with approximately perpendicular viewing directions. The cameras were calibrated offline. We used the Triclops SDK [10] to produce rectified reference and disparity images. High surface and texture validation thresholds were specified to produce much more reliable (although more sparse) disparity output.

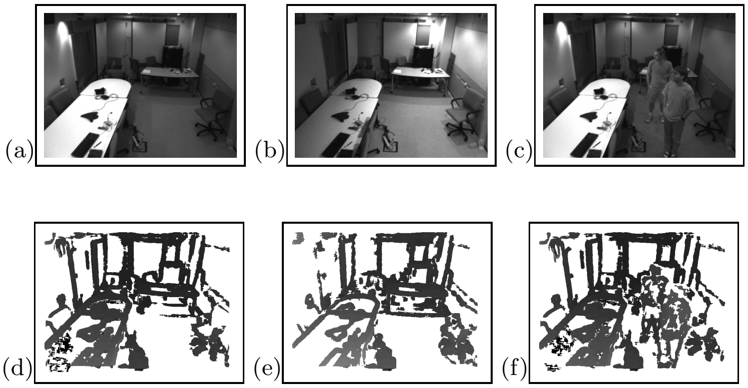


Fig. 4. Intensity and disparity pairs obtained from camera C^1. (a, d) – empty room. (b, e) – empty room under different lighting conditions. (c, f) room with two people. The pixels with invalid disparity are shown in white.

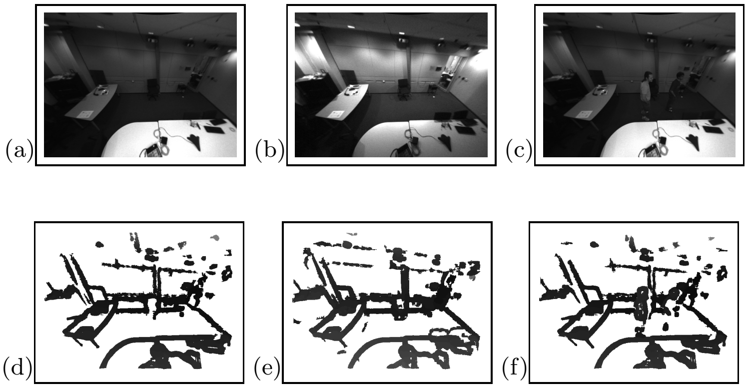


Fig. 5. Intensity and disparity pairs obtained from camera C^2. (a, d) – empty room. (b, e) – empty room under different lighting conditions. (c, f) room with two people. The pixels with invalid disparity are shown in white.

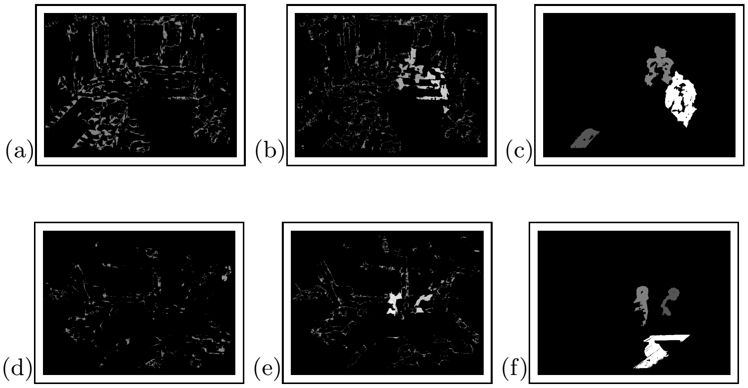


Fig. 6. Foreground detection results. (a, d) Non-conservative disparity background subtraction result for the views of the empty room under different illumination. This approach detects many false positives, when new valid range data becomes available in the background regions as illumination changes. (b, e) Conservative disparity background subtraction result between views of empty room and room with two people. This approach never detects foreground points when they appear in parts of the model with invalid range data. (c, f) Results of applying foreground segmentation algorithm described in this paper. Note that only instantaneous range information is used (Figures 4(f) and 5(f). The connected components are shown in different colors.

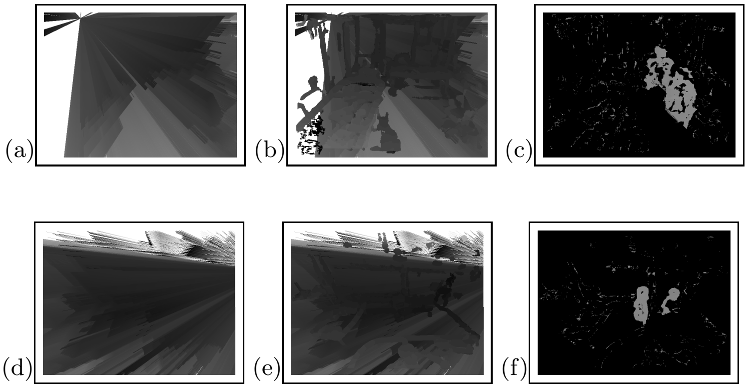


Fig. 7. Results of applying virtual backgrounds algorithm. The background disparity images from Figures 4(d) and 5(d) were used to generate virtual backgrounds (a) and (d). (b) and (e) are the background images obtained by combination of direct observations and virtual backgrounds.The (unfiltered) results of applying conservative background subtraction to images 4(f) and 5(f) are shown in (c) and (f) respectively. Compare the results from our foreground segmentation algorithm (Figure 6(c, f)), and conservative background subtraction results using background model obtained via direct observations (Figure 6(b, e)) and combined direct and virtual observations (e, f).

In Figures 6(a, d) we show the results of nonconservative background subtraction (i.e. labeling new pixel as foreground if a valid range was detected where the background model is invalid) between range views of the same scene under different lighting conditions. Figures 6(b, e) show the results of conservative background subtraction (i.e. labeling new pixel as foreground only if background model contained valid value different from the new one) when trying to segment two people in the room. As can be seen neither method produces acceptable results under the conditions we expect the segmentation algorithm to handle. Nonconservative background subtraction produces large number of false positives when illumination changes, and contrast (and thus valid range data) become available on previously uniform background regions. A conservative approach, on the other hand, never detects foreground objects where no valid range data is available in the background model.

Figures 6(c, f) demonstrate the results of applying our foreground segmentation algorithm to the same data as in figures 6(b, e). Note that the algorithm was able to correctly segment people where no background range data was available (cf. Figures 6(a) and 6(c)). Classifying parts of the table as foreground is, in fact, correct behavior of the algorithm, as there is empty space detectable behind them.

The output of the virtual background generation algorithm applied to the same data is shown in Figure 7. The dense pure virtual background images (a, d) were generated from the disparity images in Figures 4(d), 5(d). The background models (b, d) used in our conservative background subtraction, were obtained by combining direct observations and virtual range images. The resulting (unfiltered) segmentations are shown in (c, f).

5 Conclusions

We have presented two novel range-based segmentation algorithms, that take advantage of availability of multiple, widely spaced stereo views. The semi real-time foreground segmentation algorithm relies on the visibility information obtained from other views to locate points that occlude *free space*. Since the algorithm does not maintain an explicit background model and uses only immediately available reliable range information, it is able to handle variable lighting conditions. We further extended the algorithm to use visibility constraints to improve clustering of the object points. The virtual backgrounds algorithm uses the visibility information to create dense range background images which can then be used with common real-time conservative background subtraction methods.

References

1. D. Beymer and K. Konolige. Real-time tracking of multiple people using continous detection. In *Proc. International Conference on Computer Vision (ICCV'99)*, 1999.
2. B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: Technologies for intelligent environments. In *Proceedings of Second International Symposium on Handheld and Ubiquitous Computing, HUC 2000*, pages 12–29, September 2000.
3. T. Darrell, D. Demirdjian, N. Checka, and P. Felzenszwalb. Plan-view trajectory estimation with dense stereo background models. In *Proc. International Conference on Computer Vision (ICCV'01)*, 2001.
4. D. Demirdjian and T. Darrell. Motion estimation from disparity images. In *Proc. International Conference on Computer Vision (ICCV'01)*, 2001.
5. W.E.L Grimson, C. Stauffer, R. Romano, and L. Lee. Adaptive background mixture models for real-time tracking. In *Proceedings of CVPR'98*, 1998.
6. M. Harville, G. Gordon, and J. Woodfill. Foreground segmentation using adaptive mixture models in color and depth. In *Workshop on Detection and Recognition of Events in Video.*, 2001.
7. J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera multi-person tracking for easyliving. In *IEEE Workshop on Visual Surveillance*, July 2000.
8. K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *Int. Journal of Computer Vision*, 38(3):199–218, July 2000.
9. Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 369–374. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
10. B.C. Point Grey Research, Vancouver. http://www.ptgrey.com/.
11. G. Slabaugh, B. Culbertson, and T. Malzhender. A survey of methods for volumetric schene reconstruction from photographs. In *VG'01*, 2001.
12. C. Stauffer and W.E.L Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of CVPR'99*, 1999.
13. K. Toyama, J. Krumm, B. Brumitt, and B. Meyes. Wallflower: Principles and practice of background maintenance. In *In Proc. International Conference on Computer Vision (ICCV'99)*, 1999.
14. C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. In *Photonics East, SPIE, volume 2615*, 1995.

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), Kauai, HI, December 2001.

Integrated Face and Gait Recognition From Multiple Views

Gregory Shakhnarovich, Lily Lee and Trevor Darrell

MIT Artificial Intelligence Laboratory
{`gregory, llee, trevor`}`@ai.mit.edu`

Abstract. We develop a view-normalization approach to multi-view face and gait recognition. An image-based visual hull (IBVH) is computed from a set of monocular views and used to render virtual views for tracking and recognition. We determine canonical viewpoints by examining the 3-D structure, appearance (texture), and motion of the moving person. For optimal face recognition, we place virtual cameras to capture frontal face appearance; for gait recognition we place virtual cameras to capture a side-view of the person. Multiple cameras can be rendered simultaneously, and camera position is dynamically updated as the person moves through the workspace. Image sequences from each canonical view are passed to an unmodified face or gait recognition algorithm. We show that our approach provides greater recognition accuracy than is obtained using the unnormalized input sequences, and that integrated face and gait recognition provides improved performance over either modality alone. Canonical view estimation, rendering, and recognition have been efficiently implemented and can run at near real-time speeds.

1 Introduction

Person tracking and recognition systems should ideally integrate information from multiple views, and work well even when people are far away. Two key issues that make this challenging are varying appearance due to changing pose, and the relatively low resolution of images taken at a distance. We have designed a system for real-time multi-modal recognition from multiple views that substantially overcomes these two problems.

To address the first issue we adopt a view-normalization approach and use an approximate shape model to render images for recognition at canonical poses. These images are sent to externally provided recognition modules which assume view-dependent input. For distant observations view-normalization must not presume accurate 3-D models are available; our system is designed for environments where relatively coarse-disparity stereo range images or segmented monocular views are provided. We have chosen to use shape models derived from silhouette information since they are practically computable in real time from these types of input data.

To overcome the second issue, we adopt a multi-modal recognition strategy. Low-resolution information makes it less likely that recognition using any single modality will be accurate enough for many desired applications. By combining cues together, we can obtain increased performance. A typical drawback of multi-modal approaches is that they presume different types of imagery as input. Face recognition usually works best with front-parallel images of the face, whereas gait recognition often requires side-view sequences of people walking. It can be difficult in practice to simultaneously acquire those views when the person is moving along a variable path. We propose a method for view-normalization which performs this automatically, generating appropriately placed virtual views for each modality.

We have implemented a system for integrated face and gait recognition using a shape model based on an image-based visual hulls. Our recognition algorithms were separately developed for view-dependent recognition. In our system a small number of static calibrated cameras observe a workspace and generate segmented views of a person; these are used to construct a 3-D visual hull model. Canonical virtual camera positions are estimated, and rendered images from those viewpoints are passed to the recognition methods.

In the following section we will review some of the previous work related to multi-view, pose-invariant face and gait recognition. We will consider different approximate shape models for virtual view rendering, and argue for the use of the image-based visual hull algorithm due to its appealing tradeoff of accuracy and computational efficiency. We will then present new methods for estimating canonical frames given visual hull representations, based on shape, appearance, and motion cues. Finally, we will show recognition results integrating face and gait cues with separately developed view-dependent recognition modules. The particular modules we have used for our current experiments are based on principle components analysis and spatio-temporal templates, for face and gait respectively, but our framework is applicable to any view-dependent face or gait recognition method.

2 Previous work

To achieve pose-invariance, recognition models generally must incorporate information from multiple views of an object's pose. Broadly speaking, there are several classes of techniques for view-independent face recognition, including modular learning, elastic matching, view-interpolation, and geometric warping. Our visual hull approach is an instance of the last category, using multiple views and silhouette inputs.

Several authors have developed methods for recognition using a set of distinct view categories. The well-known eigenfaces paradigm was extended to recognize a set of different poses using an eigenspace for each view [17]. Rather than using replicated classifiers for distinct views, several authors have investigated elastic matching or view interpolation methods [21, 22]. Beymer and Poggio introduced a method for interpolating face views for recognition given dense correspondences, using a Radial Basis Function paradigm [2]. Seitz [19] developed a view morphing technique which used dense correspondences to interpolate rigid views of an object, but did not apply this technique to recognition.

Generalizing the notion of elastic matching, recognition based on principle components analysis of shape and texture distributions has been shown to be able to model and recognize a range of object poses[8]. When a model has been constructed fast optimization of shape and texture coefficients is possible. However, all these methods have generally presumed either knowledge of face pose and/or an accurate, dense depth or correspondence field during model training. This can be difficult to acquire in practice, so we have focused on geometric warping methods.

2.1 Geometric models

If we presume a model of the underlying geometry of the object, we can use that geometry to warp one view onto another view. For tracking faces, previous authors have used planar [3] and ellipsoidal [1] models to bring images into a canonical view. Several authors have used affine, cylindrical and ellipsoidal models for warping views during motion tracking [9, 6, 1].

Simple shape models are often inaccurate for view warping. More complex models may be used, such as warping with a depth map obtained from a laser range scanner. But as model detail increases, it becomes difficult to precisely align a static model with dynamically changing observations. This negates the value of the detailed features. To overcome these problems, we would like to use a dynamic model of actual object shape, computed in real-time from the object being tracked. Dynamic models can be recovered from a variety of sources, but we will restrict ourselves to models recovered from a set of regular cameras.

We know the relative camera positions between the views, so if we accurately knew the depth at each pixel we could simply apply view morphing or traditional rigid motion warping. However, our source views are monocular and widely separated, so it is difficult to determine correspondences using traditional methods for multi-view matching.

With a rich statistical 3-D shape model of the object class, such as developed in [4], we could estimate a 3-D shape directly from the set of 2-D appearance images, and use that to render a high-quality image from the desired view. While this is an appealing idea, we would like our method to be general, and will not in practice assume such a statistical range model is available.

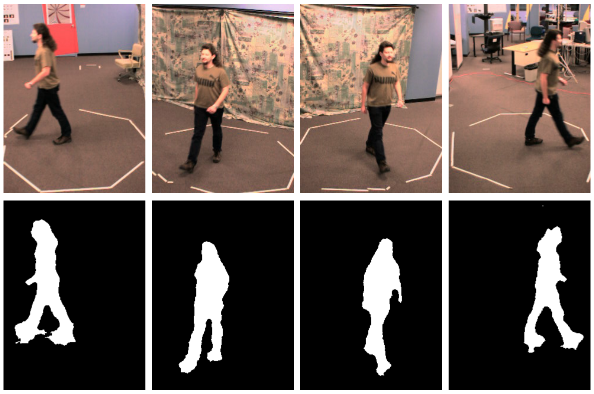
An equally appealing approach would be to apply voxel coloring or carving techniques [19, 11], to recover a discrete 3-D volumetric representation, and then use volume rendering techniques to generate the canonical view. However, these systems are computationally expensive, and require a specified discretization in 3-D which may not be optimal to re-render a given viewpoint.

We are interested in dynamic 3-D shape models that are computable without requiring dense correspondence or volumetric reconstruction. We will use a model which is computable solely from silhouette input, which we can obtain either from monocular analysis or segmentation of coarse-disparity range data.

2.2 Visual hulls

The concept of *visual hull* (VH) was introduced in [12]. A VH of an object is the maximal volume that creates all the possible silhouettes of the object. The VH is known to include the object, and to be included in the object's convex hull. In practice, the VH is usually computed with respect to a finite (often small) number of silhouettes.

An efficient technique consists of computing an *image-based VH* (IBVH) ([15]). For a desired viewpoint, for each pixel in the resulting image the intersection of the corresponding viewing ray and the VH is computed. The computation can be performed in 2D image planes, resulting in an algorithm that renders a desired view of n^2 pixels in



(a) Input

(b) Output

Fig. 1. (a) An example of rendering virtual views with an image-based visual hulls: the images obtained at the 4 cameras (top row) and their segmentation (bottom row). (b) The polyhedral VH model built from the input silhouettes in (a) (top pair), and synthetic views (bottom pair) rendered by a "virtual camera" corresponding to a frontal viewpoint. The view from the back has poor texture but reasonable shape.

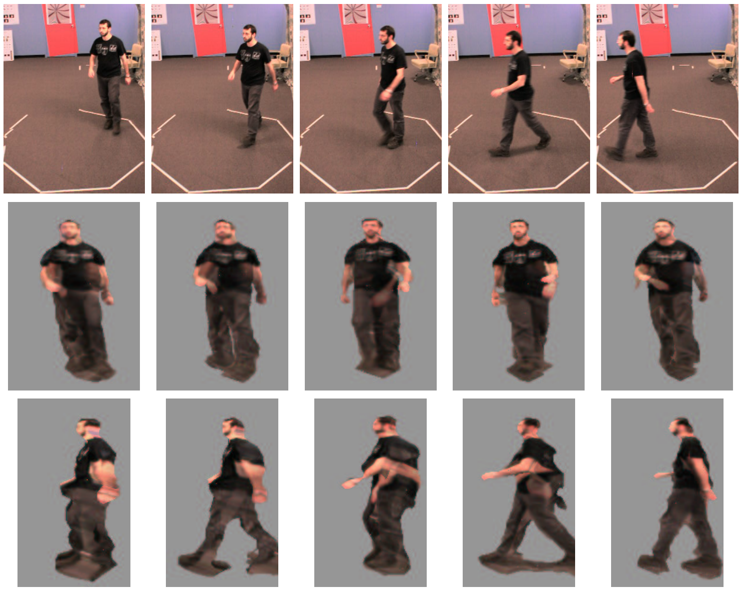


Fig. 2. An example of tracking body position and orientation using a Kalman Filter: input from one of the cameras (top row), synthetic frontal view (middle row) and synthetic side-view (bottom row).

$O(kn^2)$ where k is the number of input images (the number of views). A variant of this algorithm provides a polyhedral 3D approximation of the VH [14]. This $O(k^2n^2)$ algorithm represents contour of each silhouette as a polygon set, and computes in 2D image planes the pairwise intersections between every pait of cones, resulting in $k-1$ polygon sets for each silhouette. Intersection of these polygons set at each cone face defines the 3D polyhedron; this is the approximation of the surface of the VH with a polygonal mesh.

After the VH is constructed, its surface is texture-mapped based on the original images ([14]). Let θ_i be the angle between the viewing ray of the virtual camera for a given pixel p, and the viewing ray of the i-th camera for p. Then each view is assigned a weight $1 - \theta_i / max_i \theta_i$, and the value of p in the synthetic view is a weighted sum of the values in the original images.

Figure 1 shows an example of the original images, and the resulting VH without and with texture. The VH allows us to render a synthetic view of the object from desired viewpoints, at a moderate computational cost, and also provides information about the object's 3D location and shape. We use this information to track the position and pose of a user in the environment, and to reduce the complex task of view-invariant recognition to the simpler one of view-normalized recognition.

3 Tracking and estimating canonical views

To render virtual views for recognition, we need to determine the canonical pose of the camera which will generate the most discriminative view. In general, one could formulate the view selection process as part of the overall recognition framework, as in [5]. Indeed, given freedom to design the recognition method as well as to select the optimal view, a general optimization would be necessary. In our current work, however, we presume the use of external, black-box recognition engines for face and gait recognition. These methods have been constructed with the explicit assumption of a canonical view, so we use them directly. For faces we place the camera in the plane fronto-parallel to the face, and for gait sequences we place the camera so that it observes a side-view of the walking sequence. We have developed algorithms based on motion analysis and pattern detection to estimate these viewpoints. A strong assumption that we make is that the person is walking and generally facing forward; this allows us to use trajectory analysis to help constrain the search for canonical views.

3.1 Trajectory analysis

Without loss of generality we presume that the XZ-plane of our coordinate system is the ground plane, and the Y axis is the normal to the ground. We estimate the location of the centroid of the subject by taking the center of gravity of the VH $\mathbf{c} = \langle c_x, c_y, c_z \rangle$. The method of computing $\mathbf{c}$ depends on the VH algorithm. For the polyhedral VH, it is simply the centroid of the polyhedral model, which can be computed while building the model. This method was used in all the experiments described in this paper. For the sampled VH, one estimates the VH by integrating the volume enclosed within the endpoints of the ray intervals, and computes the zero-th moment of that volume. A third, more ad-hoc approach consisting of computation of the 3D bounding prism (which can be done directly from the silhouettes) and taking its centroid, was found by us to be inferior in practice.

Given the estimated centroids of the VH in two consecutive frames $\mathbf{c}_t$ and $\mathbf{c}_{t+1}$, we estimate the motion of the object between t and $t+1$ by $\Delta\mathbf{c} = \mathbf{c}_{t+1} - \mathbf{c}_t$. Under the assumption that the motion is parallel

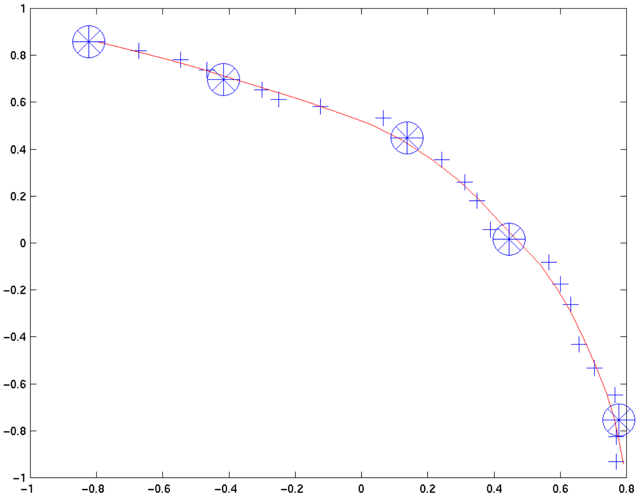


Fig. 3. Trajectory estimated from sequence in Figure 2. Frames shown in top row of Figure 2 are marked with an asterisk. Virtual views are generated along the tangent and normal to this trajectory for face and gait recognition, respectively.

to the XZ plane, we consider the projection of $\Delta\mathbf{c}$ on that plane as the motion vector.

We shall call the set of the synchronized views obtained at time t a *multiframe* f_t. The VH computed from f_t will be denoted by VH_t. Instantaneously, we need to fit a straight line $z = mx + b$ to the (noisy) centroid observations. This is done by solving a linear least-squares optimization problem, for the $\langle x_t, z_t \rangle$ in each multiframe VH_t. This gives us the unit vector $\mathbf{v}_t$ in the estimated direction of the person at time t. Once we have established the direction, we can place a "virtual camera", say, in front of the person, at a desired distance δ:

$$O_t = \mathbf{c}_t + \delta \mathbf{v}_t \tag{1}$$

For a general trajectory, we use a constant-velocity Kalman filter to recover the centroid path.

Figures 2 and 3 demonstrates the results of the method. Input from only one camera out of four is shown for reference. While the orientation estimate is not perfect, we keep track of the orientation after the person turns at about 60 degrees, and can automatically produce synthetic frontal (middle row of Figure 2) and profile (bottom row) views. (Note that there are some texture rendering artifacts present in the profile sequence–these are visually distracting but do not cause problems for our silhouette based gait algorithm.)

The assumption of fronto-parallel motion implicit in our trajectory analysis can be relaxed by combining the motion-based orientation estimate with one based on face-detection, as described in the next section.

3.2 Detection-based view estimation

Fig. 4. View-normalized gait and face recognition features based on trajectory in Figure 3.

A pattern detection approach can be applied to a set of rendered virtual views to find those that are most "canonical" relative to a desired class. For faces, we use a real-time face-detection method [20] to detect the frontal view condition. This implementation, which uses small number of highly-relevant features, can process images of 400x300 pixels in roughly .07 seconds. However, we need to apply it to much smaller images. Given the VH of a person, and assuming roughly upright body pose, we need to consider only the top part of the VH. In our experiments we chose to look at the top 1.5 feet. We place the virtual camera at the distance that would produce the desired resolution of the image (in the described setup, 60x60 pixels).

If no trajectory information is available, we can search a circle of views around the 3-D location of a users head (Figure 5). If trajectory information is available, the head area is then rendered for a small range of spatial angles around the currently estimated face orientation.

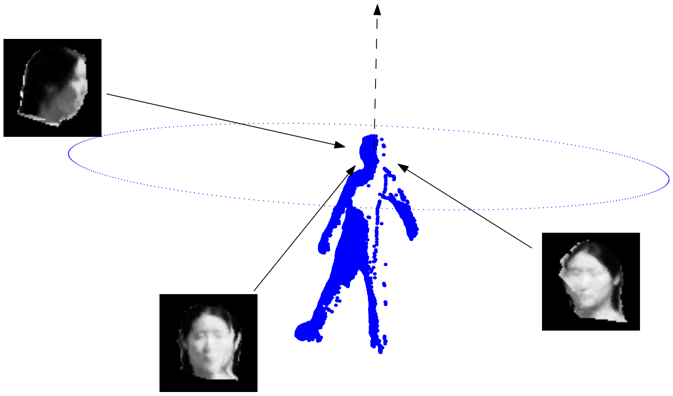


Fig. 5. Virtual views can also be generated based on the position of the users head and a ground plane constraint.

A set of 25 such images has the same total size as one 300x300 image, and takes similar time for a face detector to process.

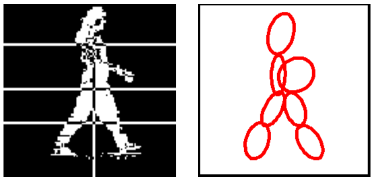
We also reduce the scale space, since the virtual camera is placed at a known distance from the VH, thus leading only a small range of possible sizes of the face.

4 Recognition on virtual sequences

We take the virtual sequences rendered from canonical viewpoints and input them to view-dependent face and gait recognition algorithms. Typically these methods are based on 2-D or 2.5-D (XY+T) analysis.

4.1 Gait recognition

Human gait can serve as a discriminative feature for visual recognition, as suggested by theoretical biometric ([10]) and empirical ([7, 16, 18]) results. Here we applied a simple gait recognition scheme based on silhouette extent analysis, which was developed separately from our work. The basic method is reported in [13] and was successfully demonstrated on sequences where the direction of motion was explicitly parallel to the camera plane.



(a) Partition of a silhouette

(b) Fitting an ellipse to each region

Fig. 6. Computing the feature vector for gait recognition. From [13].

The gait dynamics feature vector consists of smoothed versions of moment features in image regions containing the walking person. For each silhouette of a gait video sequence, we find the centroid of the whole silhouette and divide it into 7 regions using the centroid. For each of the regions, we fit an ellipse to describe the centroid, the aspect ratio and the orientation of the portion of foreground object visible in that region(Figure 6(b)). These silhouette–based features are computed for each frame of a video sequence. These time-varying signals from a video sequence are compressed across time using the mean and standard deviation of the centroid, aspect ratio, and orientation of each region. The time-compressed features from all 7 regions together form a gait feature vector. A diagonal covariance Gaussian model is used for each of these features, and a nearest neighbor classifier is used to decide which person has walking dynamics closest to the query feature vector. This method is surprisingly simple, but works in a range of realistic conditions [13]. More complex models, including those that recover kinematic biometrics and/or periodic features, could also be easily integrated into our framework.

The features used in this gait recognition algorithm are clearly view-dependent, and it is generally impractical to collect data for each person across all possible views. Recognition using a sequence rendered from a virtual viewpoint in canonical position is an appealing alternative. For each sequence of multiframes $\mathbf{x}$, two silhouette sequences are produced - a synthetic view from the left and from the right can be created for each frame, relative to the estimated motion vector. We denote those by $\mathbf{s}_L$ and $\mathbf{s}_R$. Figure 4(top) shows an example view-normalized silhouette input to the recognition method.

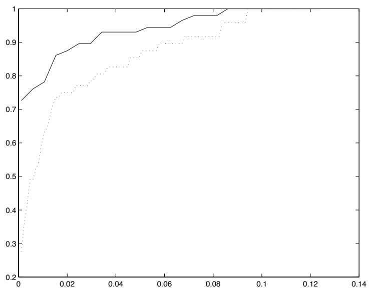


Fig. 7. A rank-threshold plot for gait recognition using view-normalization (solid line) versus using only the raw input silhouettes (dotted line).

We maintain an ID-tagged database of silhouette sequences, obtained from the VH of the previously observed people. To recognize a new sequence, we compute the distances between the feature vector of $\mathbf{s}_L$ and $\mathbf{s}_R$ and those of all the silhouette sequences $\mathbf{s}$ in the database. We exclude $\mathbf{s_L}$ and $\mathbf{s_R}$ themselves, and choose the minimum between the two values as the distance between $\mathbf{x}$ and the other silhouettes. Then, we normalize the vector

$$\mathbf{p}_g(\mathbf{x}) = \left[1/\min_{label(\mathbf{s})=1} dist(\mathbf{s},\mathbf{x}),\ldots,1/\min_{label(\mathbf{s})=K} dist(\mathbf{s},\mathbf{x})\right] \tag{2}$$

The estimated confidence that $\mathbf{x}$ is actually from person k is denoted $\mathbf{p}_{g_k}(\mathbf{x})$. Choosing k which maximizes this confidence gives our classification decision.

4.2 Face recognition

When a scene is viewed by a small number of far-placed cameras, often there is no view close enough to frontal to allow face recognition, and even detection. For example, on all of the original textures in Figure 8(a) face detection fails. However, faces are easily detected in the frontal virtual views, such as that shown in Figure 4(bottom) and Figure 8(b). Figure 8(c) shows a sample of view-normalized model faces.

We consider face recognition algorithms that are trained on a database with certain amount of view-dependence. Typically such a database includes frontal views of faces. So far, we tested our approach with eigenfaces.

For each multiframe x_t, we render synthetic views of the top part of VH for a small range of spatial angles around the estimated motion

(a) Original

(b) VN

(c)

Fig. 8. Face detection typically fails on the input views due to varying pose (a), but succeeds on the visual hull-based view-normalized image (b). Pairs of view-normalized faces from the same individual are shown in (c). Conventional view-dependent face recognition methods can match (b) to the appropriate individual in (c) (top row, right or bottom row, second right).

vector. These images are processed by a face detector, and the ones where a face was detected are included in a set of $Faces_t(\mathbf{x})$. After having seen n frames, the set $Faces(\mathbf{x}) = \bigcup_{i=1}^{n} Faces_i(\mathbf{x})$. If $Faces(\mathbf{x})$ is non-empty, we can use all the face images in it for recognition. Let $m = |Faces(\mathbf{x})|$. Let D be an $m \times K$ matrix of distances between each $I_i \in Faces(\mathbf{x})$ and each one of the K eigenspaces represented in the database:

$$D_{ij} = |I_i - \mathbf{S}_j I_i|, \tag{3}$$

Then we compute for each image I_i a weight vector $w_i = [1/D_{i1}, \ldots, 1/D_{iK}]$, which is further normalized to produce a confidence vector. This vector describes the estimated confidence that I_i belongs to the Kth person. We have m images, so for the whole sequence $\mathbf{x}$ we compute the confidence vector

$$\mathbf{p}_f(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{w}_i. \tag{4}$$

Our classification is then done by selecting

$$\mathbf{x} = \underset{j}{\operatorname{argmax}}\, p_{f\,j}(\mathbf{x}).$$

4.3 Multi-modal recognition

Finally, we combine the face and gait recognition results in order to establish a higher confidence level. Since empirically the success rates of face and gait classifiers were similar (c.f. Table 1(d)), we assigned an equal weight of .5 when combining confidence vectors. Given $\mathbf{p}_f(\mathbf{x})$ and $\mathbf{p}_g(\mathbf{x})$ for the observed sequence of multi-views $\mathbf{x}$, we compute the multi-modal confidence vector

$$\mathbf{p}_c(\mathbf{x}) = \begin{cases} \mathbf{p}_g(\mathbf{x}), & \text{if } Faces(\mathbf{x}) = \emptyset \\ \left(\mathbf{p}_g(\mathbf{x}) + \mathbf{p}_f(\mathbf{x})\right)/2, & \text{otherwise.} \end{cases} \tag{5}$$

(a)

5	0	0	0	0	0	0	0	0	**1**	0	0
0	**4**	0	0	0	0	0	0	**1**	0	0	**1**
0	0	**3**	0	0	0	0	**1**	0	0	0	0
0	0	0	**4**	0	0	0	0	0	0	0	0
0	0	0	0	**4**	0	0	0	0	**1**	0	0
0	0	0	0	0	**3**	0	0	0	0	**1**	0
0	0	0	0	0	0	**6**	0	0	0	0	0
0	0	0	**1**	0	0	0	**4**	0	0	**1**	**1**
0	0	0	0	0	0	0	0	**2**	0	0	0
0	0	0	0	0	0	0	0	0	**6**	0	0
0	0	0	0	0	0	0	0	0	**1**	**2**	0
0	0	0	0	0	0	0	0	0	0	0	**4**

(b)

5	0	**1**	0	0	0	0	0	0	0	0	0
0	**4**	0	0	0	0	0	0	0	0	0	0
0	0	**2**	0	0	0	**2**	0	0	0	0	0
0	0	0	**4**	0	0	0	0	0	0	0	0
0	0	0	0	**5**	0	0	0	0	0	0	0
0	0	0	0	**3**	0	0	0	0	0	0	**1**
0	0	**1**	0	0	0	**5**	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	0	**6**	0	0
0	0	0	0	0	0	0	0	0	0	**3**	0
0	0	0	0	0	0	0	0	0	0	0	**4**

(c)

6	0	0	0	0	0	0	0	0	0	0	0
0	**4**	0	0	0	0	0	0	0	0	0	0
1	0	**3**	0	0	0	0	0	0	0	0	0
0	0	0	**4**	0	0	0	0	0	0	0	0
0	0	0	0	**4**	0	0	0	0	**1**	0	0
0	0	0	0	0	**3**	0	0	0	0	**1**	0
0	0	0	0	0	0	**6**	0	0	0	0	0
0	0	0	0	0	0	0	**4**	0	0	**1**	**1**
0	0	0	0	0	0	0	0	**2**	0	0	0
0	0	0	0	0	0	0	0	0	**6**	0	0
0	0	0	0	0	0	0	0	0	0	**3**	0
0	0	0	0	0	0	0	0	0	0	0	**4**

(d)

Modality	(chance)	No VH-face	No VH-gait	No VH-gait and face	VH-face only	VH-gait only	VH-gait and face
Recognition rate	.08	.31	.52	.44	.8	.87	.91

Table 1. Confusion matrices for (a) gait-only, (b) face-only, and (c) integrated recognition using VH. Note that there was no face data obtained for subject 8, who was wearing a hat during the experiments. (d) Summary of the recognition results.

5 Results

We tested our methods using an installation with four monocular cameras. Each were located at roughly the same height, approximately 45 degrees apart, yielding set of images like that in Figure 1. The intersection of their fields of view defines the working space of our system. The cameras were calibrated off-line and temporally synchronized in hardware.

Silhouettes were computed using a simple color background model. For each pixel, the mean and variance of its values are computed over a large number of frames when the scene is known to contain no object. Segmentation is performed with three steps. First, each pixel in the data image is labelled 'background' if its value is within two standard deviations from the mean, and 'foreground' otherwise. Second, a normalized correlation analysis is then computed for a small window around each foreground pixel, and it is reset to background if the correlation score is sufficiently high. Finally, a morphological close operation is performed. The last two steps reduce the impact of shadows.

For 12 subjects we collected between 2 and 6 VH sequences as they walked in an arbitrary direction through the visual hull workspace, which was approximately 3m in diameter. The accuracy of gait classification was estimated using leave-one-out cross-validation. Figure 7 compares gait recognition performance using normalized vs. unnormalized views. Accuracy vs. rank threshold is plotted for the each approach, indicating the percentage of trials where the correct label was within the top n predicted labels (where n is the rank-threshold value). As can be seen, recognition with the unnormalized sequences was substantially worse than with our view-normalization approach. A confusion matrix for $n = 1$ is shown in Table 1(a)

View-normalized face recognition was also performed on these data, using the method described above. Table 1(b) shows the results of classification using only the face observations. Finally, Table 1(c) shows the confusion matrix for integrated recognition. Table 1(d) summarizes the overall recognition rates for face-only, gait-only, and integrated recognition. Integrated recognition reduced the rank-threshold=1 recognition error rate from 13% to 9%.

Note the significantly inferior performance of the recognition in both modalities with the same data, but when no view-normalization is applied (Table 1 (d)). In this experiment, we used the images from all the four cameras, where segmented silhouettes were fed to the gait classifier, and face detection was used to extract faces from the textured camera inputs (with silhouettes defining the search regions). Face recognition performed especially poorly. In many sequences not a single face was detected, which is not surprising after looking at Figure 8. In addition, some false detections further decrease the performance.

6 Conclusions and future work

We have described a view-normalization approach for integrated tracking and recognition of people. Our system combines face and gait recognition methods, and information from multiple views. An image-based visual hull is used for shape modeling and for trajectory tracking. Results were shown using view-dependent face and gait recognition modules, and were better than the unnormalized or single modality results. Each component of the system runs at real-time speeds.

Currently the implementation uses monocular silhouettes based on color segmentation with static backgrounds, but could be extended to accommodate more sophisticated segmentation algorithms. Our system works within the strict intersection of the field of view of all cameras, but we expect this to be relaxed as a more general visual hull algorithm is developed. Finally, our confidence integration method is clearly primitive in present form, and should be extended to an explicit probabilistic framework.

7 Acknowledgments

We are grateful to M. Jones and P. Viola who provided us with the implementation of the face detector, and to W. Matusik, C. Buehler and L. McMillan who developed the Visual Hull system, on which our experiments were based. This generous financial support by the DARPA HumanID program and MIT Project Oxygen is gratefully acknowledged.

References

1. Sumit Basu, Irfan Essa, and Alex Pentland. Motion regularization for model-based head tracking. In *Proceedings, 13th International Conference on Pattern Recognition*, Vienna, Austria, August 1996. IEEE Computer Society Press.
2. D. Beymer and T. Poggio. Face recognition from one example view. In *Proccedings of the International Conference on Computer Vision*, pages 500–507, 1995.
3. M. J. Black and Y. Yacoob. Recognizing facial expressions in image sequences using local parameterized models of image motion. In *Procceedings of the International Conference on Computer Vision*, 1995.
4. V. Blanz and T. Vetter. A morphable model for the synthesis of 3–d faces. In *Computer Graphics, SIGGRAPH Proceedings*, pages 71–78, Los Angeles, CA, 1999.
5. F. G. Callari and F. P. Ferrie. Active recognition: Using uncertainty to reduce ambiguity. In *In Proceedings of the 13th International Conference on Pattern Recognition*, Vienna, Austria, Aug.25-30 1996.

6. M. La Cascia, J. Isidoro, , and S. Sclaroff. Head tracking via robust registration in texture map images. In *Proceedings Computer Vision and Pattern Recognition (CVPR'98)*, pages 508–514, Santa Barbara, CA, 1998.
7. J.E. Cutting and L.T. Kozlowski. Recognizing friends by their walk: gait perception without familiarity cues. *Bull. Psychonometric Soc.*, (9):353–356, 1977.
8. G. J. Edwards, T. F. Cootes, and Christopher J. Taylor. Face recognition using active appearance models. In *ECCV (2)*, pages 581–595. Springer, 1998.
9. G. D. Hager and P. N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(10):1025–1039, 1998.
10. G Johansson. Visual motion perception. *Scientific American*, (232):76–88, 1975.
11. K. Kutulakos and S. Seitz. A theory of shape by space carving. In *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV-99)*, volume I, pages 307–314, Los Alamitos, CA, September 20–27 1999. IEEE.
12. A. Laurentini. The visual hull concept for silhouette-based image understanding. *PAMI*, 16(2):150–162, February 1994.
13. L Lee. Gait dynamics for recognition and classification. Technical Report AIM-2001-019, MIT AI Lab Memo, Sept. 2001.
14. Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. to appear in *Proceedings of EGWR-2001*, 2001.
15. Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 369–374. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
16. S.A. Niyogi and E.H. Adelson. Analyzing and recognizing walking figures in xyt. In Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp.469-474, 1994.
17. A. Pentland, B. Moghaddam, T. Starner, O.Oliyide, and M. Turk. View-based and modular eigenspaces for face recognition. Technical Report 245, MIT Media Lab Vismod, 1993.
18. R. Polana and R. Nelson. Low level recognition of human motion. In IEEE Workshop on Motion of Non-Rigid and Articulated Objects, pages 77– 82, Austin, 1994.
19. S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proceedings Computer Vision and Pattern Recognition (CVPR'97)*, pages 1067–1073, 1997.
20. Paul A. Viola and Michael J. Jones. Robust real-time object detection. Technical report, COMPAQ Cambridge research Laboratory, Cambridge, MA, February 2001.

21. L. Wiskott, J.M. Fellous, N. Kruger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7):775-779, 1997.
22. A.L. Yuille. Deformable templates for face recognition. Journal of Cognitive Neuroscience, 3(1):59-70., 1991.

Proceedings of the 2001 Workshop on
Perceptual User Interfaces (PUI'01),
Orlando, FL, May 2001.

Audio-Video Array Source Separation for Perceptual User Interfaces

Kevin Wilson, Neal Checka, David Demirdjian and Trevor Darrell

MIT Artificial Intelligence Laboratory
{`kwilson, nealc, demirdj, trevor`}`@ai.mit.edu`

Abstract. Steerable microphone arrays provide a flexible infrastructure for audio source separation. In order for them to be used effectively in perceptual user interfaces, there must be a mechanism in place for steering the focus of the array to the sound source. Audio-only steering techniques often perform poorly in the presence of multiple sound sources or strong reverberation. Video-only techniques can achieve high spatial precision but require that the audio and video subsystems be accurately calibrated to preserve this precision. We present an audio-video localization technique that combines the benefits of the two modalities. We implement our technique in a test environment containing multiple stereo cameras and a room-sized microphone array. Our technique achieves an 8.9 dB improvement over a single far-field microphone and a 6.7 dB improvement over source separation based on video-only localization.

1 Introduction

Many current perceptual user interface applications require high-quality audio signals for acceptable performance. Examples include automated speech recognition (ASR) and smart teleconferencing. When hands-free operation is required, the most common ways to obtain audio signals for these applications are to use close-talking microphones that are attached to the speakers of interest or to use single-element directional microphones pointed at the speakers of interest.

However, both of these techniques leave much to be desired. Close-talking microphones require that each user be equipped with his own microphone, while directional microphones are often bulky and are limited to a fixed beam pattern, thus restricting their ability to track multiple users.

An alternate technique that has become more attractive with the decreasing cost of computation and digital communication is the microphone array. A microphone array consists of several microphones in fixed locations relative to each other. The microphones' audio signals can be filtered and summed to perform spatial filtering of the audio sources in the room. By altering the filters applied to the individual microphones' signals, sounds coming from different regions of the room can be selectively amplified or attenuated.

Microphone arrays address many of the problems inherent in more passive audio capture techniques. Unlike close-talking microphone systems, microphone arrays do not require users to remember to wear special equipment when they anticipate that they will interact with the environment. Instead, microphone arrays have, as a fundamental property, an explicit notion of the spatial relationships among sound sources.

This association between sound and location makes a microphone array a powerful tool in the context of perceptive environments. In combination with additional sensors and contextual information from the environment, a microphone array can effectively amplify and separate sounds of interest from complex background noise.

To focus a microphone array, the location of the speaker(s) of interest must be known in order for the microphone array to modify its filter response to amplify the selected speakers. A number of techniques exist for localizing sound sources using the array data itself [12], but the performance of these localization techniques tends to degrade significantly in the presence of reverberation and/or multiple sound sources. Unfortunately, most common office and meeting room environments are highly reverberant, with reflective wall and table surfaces, and will normally contain multiple speakers.

In our application, we can take advantage of other sensors in the perceptive environment domain to perform multimodal localization of multiple speakers despite reverberation. Because the wavelength of visible light is much smaller than the wavelength of audible sound, cameras can be much more precise in their localization, and multiple users can be more easily segmented in space.

Cameras, however, are not perfect for steering a microphone array. It may be difficult to obtain a precise joint calibration between the cameras and the microphone array. In addition, the features that a camera-based system can most easily track, such as extremities of the body, are not directly relevant to the microphone array; the microphone array requires information about the location of the speaker's mouth, which is difficult to obtain from wide-angle camera views of the environment.

Because of these issues, a microphone array aimed using only information from a set of cameras will likely be incorrectly aimed, resulting in a loss of several decibels of performance and an undesirable spectral coloration of the signal of interest. In spite of these problems, video localization information is accurate enough to restrict the range of possible acoustic source locations to a region small enough to allow for acoustic localization techniques to operate without severe problems with reverberation and multiple speakers.

As far as we are aware, our system is the first visually guided large-aperature microphone array. This paper demonstrates the use of 3-D visual localization in combination with acoustic localization to acquire high-quality audio speech signals from moving users in a perceptually enabled environment.

2 Background

This work brings together techniques from array signal processing with techniques from vision-based person tracking to implement a system that can selectively amplify audio from a selected speaker as he moves through the room. Much work has been done in both of these areas. The relevant background is summarized below.

2.1 Microphone arrays

Microphone arrays are a special case of the more general problem of sensor arrays, which have been studied extensively in the context of applications such as radar and sonar [11]. The Huge Microphone Array project[10] is investigating the use of very large arrays containing hundreds of microphones. Their work concentrates on audio-only solutions to array processing. Another related project is Wang and Brandstein's audio-guided active camera[13], which uses audio localization to steer a camera on a pan/tilt base.

Many problems can be addressed through array processing. The two array processing problems that are relevant to our system are beamforming and source localization.

Beamforming is a type of spatial filtering in which the signals from individual array elements are filtered and added together to produce an output that amplifies signals coming from selected regions of space and attenuates sounds from other regions of space. In the simplest form of beamforming, delay-and-sum beamforming, each channel's filter is a pure delay. The delay for each channel is chosen such that signals from a chosen "target location" are aligned in the array output. Signals from other locations will tend to be combined incoherently. For example, if a three element array consists of elements that are 2, 4, and 7 meters away from a target location, the elements' signals should be delayed by the time that it takes for sound to travel 5, 3, and 0 meters, respectively. This type of beamforming is simple and robust to small uncertainties in microphone and target locations.

Source localization is a complementary problem to beamforming whose goal is to estimate the location of a signal source. One way to do this is to beamform to all candidate locations and to pick the location that yields the strongest response. This method works well, but the amount of computation required to do a full search of a room is prohibitively large. Another method for source localization consists of estimating relative delays among channels and using these delays to calculate the location of the source. Delay-estimation techniques are computationally efficient but tend to perform poorly in the presence of multiple sources and/or reverberation.

A number of projects [2–4] have used vision to steer a microphone array, but because they use a single camera to steer a far-field array, they cannot obtain or make use of full 3-D position information; they can only select sound coming from a certain direction.

For microphone arrays that are small in size compared to the distance to the sources of interest, incoming wavefronts are approximately planar. Because of this, only source direction can be determined; source distance remains ambiguous. When the array is large compared to the source distance, the sphericity of the incoming wavefronts is detectable, and both direction and distance can be determined. These effects of array size apply both to localization and to beamforming, so if sources at different distances in the same direction must be separated, a large array must be used.

As a result, with large arrays the signal-to-noise ratio (for a given source) at different sensors will vary with source location. Because of this, signals with better signal-to-noise ratios should be weighted more heavily in the output of the array. Our formulation of the steering algorithm presented below takes this into account.

2.2 Person tracking

Tracking people in known environments has recently become an active area of research in computer vision. Several person-tracking systems have been developed to detect the number of people present as well as their 3D position over time. These systems use a combination of foreground/background classification, clustering of novel points, and trajectory estimation over time in one or more camera views [6, 9].

Color-based approaches to background modeling have difficulty with illumination variation due to changing lighting and/or video projection. To overcome this problem, several researchers have supported the use of background models based on stereo range data [6, 8]. Unfortunately, most of these systems are based on computationally intense, exhaustive stereo disparity search.

We have developed a system that can perform dense, fast range-based tracking with modest computational complexity. We apply ordered disparity search techniques to prune most of the disparity search computation during foreground detection and disparity estimation, yielding a fast, illumination-insensitive 3D tracking system. Details of our system are presented in [5]; here we review the details of our visual tracking system which are relevant to the integration with audio processing in our microphone array.

When tracking multiple people, we have found that rendering an orthographic vertical projection of detected foreground pixels is a useful representation (see also [1, 9]). A "plan view" image facilitates correspondence in time since only 2D search is required. Previous systems would segment foreground data into regions prior to projecting into a plan-view, followed by region-level tracking and integration, potentially leading to sub-optimal segmentation and/or object fragmentation. Instead, we develop a technique that altogether avoids any early segmentation of foreground data. We merge the plan-view images from each view and estimate over time a set of trajectories that best represents the integrated foreground density. Trajectory estimation is performed by finding connected components in a spatio-temporal filtered volume.

To estimate the trajectory of objects over time, we combine information from multiple stereo views. The true extent of an individual object in a given image is generally difficult to identify. An optimal trajectory segmentation should consider the assignment of an individual pixel to all possible trajectories estimated over time. Systems which perform an early segmentation and grouping of foreground data before trajectory estimation preclude this possibility.

We adopt a late-segmentation strategy that finds the best trajectory in an integrated spatio-temporal representation by combining foreground pixels from each view. By assuming that objects move on a ground plane, a "plan-view assumption" allows us to completely model instantaneous foreground information as a 2-D orthographic density

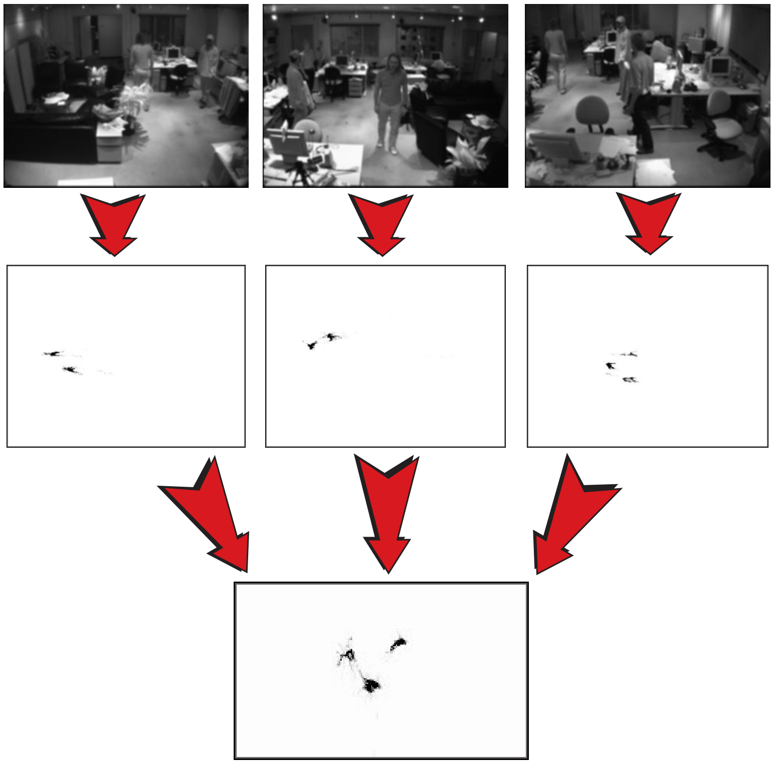


Fig. 1. Detecting locations of users in a room using multiple views and plan-view integration. Three people are standing in a room, though not all are visible to each camera. Foreground points are projected onto a ground plane. Ground plane points from all cameras are then superimposed into a single data set before clustering the points to find person locations.

projection. Over time, we compute a 3-D spatio-temporal plan-view volume.

We project (x_j, y_j, d_j) from each foreground point $\boldsymbol{p}_j$ into world coordinates (U_j, V_j, W_j). (See Figure 4.) U, V are chosen to be orthogonal axes on the ground plane, and W normal to the ground plane. We then compute the spatio-temporal plan view volume (Figure 1), with

$$P(u,v,t) = \sum_{\{\boldsymbol{p}_j | U_j=u, V_j=v, t_j=t\}} 1$$

Each independently moving object in the scene generates a continuous volume in the spatio-temporal plan view volume $P(u, v, t)$. When the trajectories of moving objects do not overlap, the trajectory estimation is easy and consists in running a connected-component analysis in $P(u, v, t)$ (each component is then a trajectory).

When the trajectories of moving objects overlap (*e.g.* crossing of two people), the volume associated with these trajectories in $P(u, v, t)$ also overlap and make the extraction of trajectories more difficult. In order to overcome this, a graph is built from a piece-wise connected-component analysis of $P(u, v, t)$. Nodes correspond here to trajectory crossing and branches to non-ambiguous trajectories between two crossing. A color histogram is then estimated for each branch of the graph (using all images associated with this branch). Trajectories are estimated by finding in the graph the paths consisting of branches having the most similar color histograms. This may be done instantaneously using a greedy search strategy or using the slower but optimal dynamic programming technique described in [5].

3 Large-array volume selection

Our system performs both audio localization and beamforming with a large, ceiling-mounted microphone array. Localization uses information from both audio and video, while beamforming uses only the audio data and the results of the localization processing. A large array gives the ability to select a *volume* of 3-D space, rather than simply form a 2-D beam of enhanced response as anticipated by the standard array localication algorithms. However, the usual assumption that of constant target signal-to-noise ratio (SNR) across the array does not hold when the array geometery is large (array width on same scale as target distance.) As described below, we need to model the SNR term in the array localization algorithm.

3.1 Localization

Our system uses the location estimate from the vision tracker as the initial guess from which to begin a gradient ascent search for a local maximum in beam output power. Beam power is defined as the integral over a half-second window of the square of the output amplitude.

It is difficult to characterize the error in the tracker's estimate because this error depends on the person's position in the room, the person's appearance, and a number of other characteristics of the situation. However, experience leads us to believe that the vision tracker is accurate to within less than one meter. Gradient ascent to the nearest local maximum can therefore be expected to converge to the location of the speaker of interest when no other speakers are very close by.

Gradient ascent is complicated by the fact that there are many high-spatial-frequency ripples superimposed on the large-scale peak whose maximum we wish to find. These small ripples in the response result in many undesirable local maxima that must be avoided. Because speech is a broadband signal, it is possible to start the gradient ascent using a low-passed version of the speech signal. As the peak is approached, the cut-off frequency of the filter can be raised, thus incorporating more of the signal into the location estimate. This technique is similar to one used in [7] as part of an exhaustive search for a power maximum.

3.2 Source separation

For small microphone arrays, the relative SNRs of the individual channels do not vary significantly as a function of source location. This is, however, not true for larger microphone arrays. For our array, which is roughly 4 meters across, we must take into account the fact that some elements will have better signals than others. Specifically, if we assume that we have signals x_1 and x_2 which are versions of the unit-variance desired signal, s, that have been contaminated by unit-variance uncorrelated noise, we can analyze the problem as follows:

$$x_1 = a_1 s + n_1$$

$$x_2 = a_2 s + n_2$$

In this model, the signal to noise ratios of x_1 and x_2 will be a_1^2 and a_2^2, respectively. Their optimal linear combination will be of the form $y = bx_1 + x_2$. Because of the uncorrelated noise assumption, the SNR of this combination will be

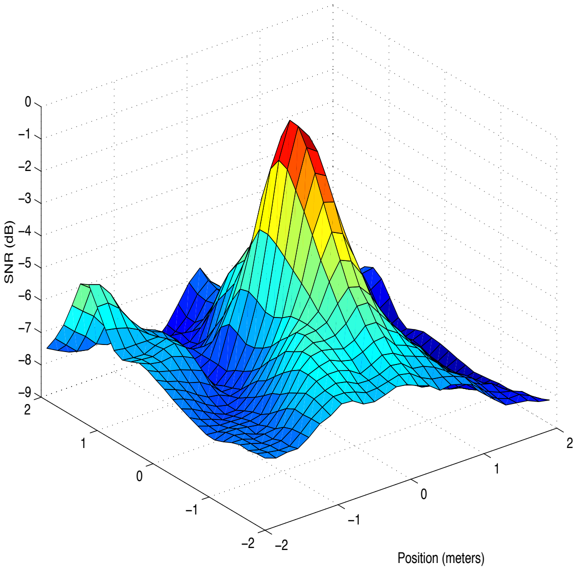


Fig. 2. Array power response as a function of position (single speaker close-up). This plot shows the array output power as the array's focus is scanned through a plane centered on a speaker.

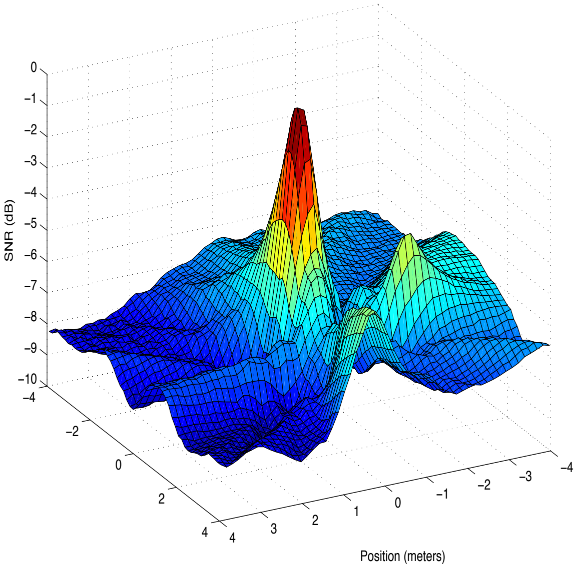


Fig. 3. Array power response as a function of position (two speakers). This plot shows the array output power as the array's focus is scanned through a plane centered on one speaker while another speaker is nearby. The central speaker is easily discernible in the plot, but the peak corresponding to the weaker speaker is difficult to distinguish among the sidelobe peaks.

$$SNR(y) = \frac{(ba_1 + a_2)^2}{b^2 + 1}$$

By taking the derivative of this expression with respect to b and setting the result equal to zero, one finds that the optimal value of b is:

$$b = \frac{a_1}{a_2} = \frac{SNR(x_1)}{SNR(x_2)}$$

Because of the symmetry of the signals, this result implies that, in general, individual elements' signals should be scaled by a constant proportional to the square root of their SNRs.

Ideally, we would like to have complete knowledge of the strengths and statistical relationships among the noise signals at the individual sensors. This information is not easy to obtain, but because of our large array and multiple stereo cameras, it is easy for us to use our location estimate to weight individual channels assuming a $1/r$ attenuation due to the spherical spreading of the source. Assuming $1/r$ attenuation from a source to each microphone, we have $a_n = 1/r_n$ in the above equations, so the optimal weighting factor for channel n is $1/r_n$. This is intuitively appealing since it means that microphones far from the source contribute relatively little to the array output.

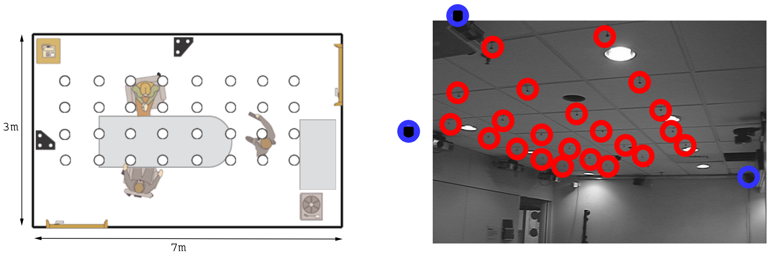


Fig. 4. The test environment. On the left is a schematic view of the environment with stereo cameras represented by black triangles and microphones represented by empty circles. On the right is a photograph of the environment with microphones and camera locations highlighted.

4 Results

Our test environment, depicted in Figure 4, is a conference room equipped with 32 omnidirectional microphones spread across the ceiling and 2 stereo cameras on adjacent walls.

The audio and video subsystems were calibrated independently, and for our experiments, we performed a joint calibration by finding the least-squares best-fit alignment between the two coordinate systems.

Table 1. Audio-video localization performance comparison

Localization Technique	SNR (dB)
Single microphone	-6.6
Video only	-4.4
Audio only	2.0
Audio-Video	2.3

Figure 3 is an example of what happens when multiple speakers are present in the room. Audio-only gradient ascent could easily find one of the undesirable local maxima. Because our vision-based tracker is accurate to within one meter, we can safely assume that we will find the correct local maximum even in the presence of interferers.

To validate our localization and source separation techniques, we ran an experiment in which two speakers spoke simultaneously while one of them moved through the room. We tracked the moving speaker with the stereo tracker and processed the corresponding audio stream using three different localization techniques. For each, we used a reference signal collected with a close-talking microphone to calculate both a time-averaged SNR (Table 1) and a sequence of short-time SNRs (Figure 5).

As a reference for performance comparison, we use the signal from a single microphone near the center of the room. This provides no spatial selectivity, but for our scenario it tends to receive the desired speech more strongly than the interfering speech. The SNR for this case is negative because of a combination of the interfering speaker and diffuse noise from the room's ventilation system.

To evaluate the video-only approach, we steer the microphone array directly to the location returned by the stereo tracker. If the stereo tracker could reliably return the location of the speaker's mouth, this method would work quite well. For our system, this technique improves

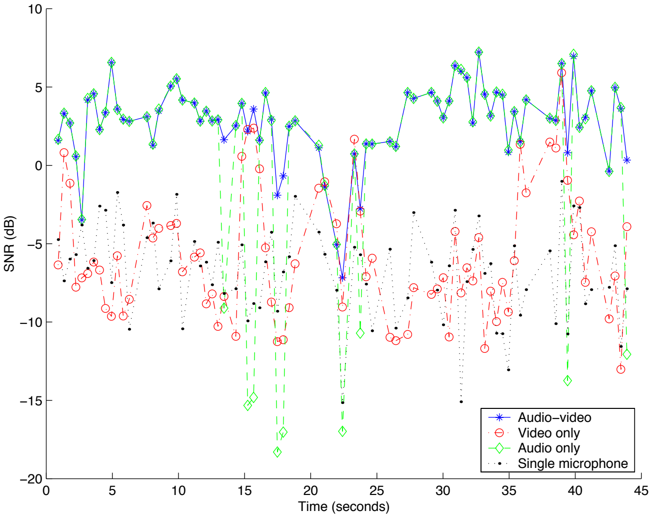


Fig. 5. Short-time signal-to-noise ratio comparison. We calculate a sequence of SNRs for non-overlapping half-second windows of audio. Much of the variation in the SNR of audio-video result is due to variations in speech energy.

the SNR by 2.2 dB, which, while noticeable, is not close to the theoretical performance of a 32 element array (15 dB in uncorrelated noise). Figure 5 shows large fluctuations in SNR for this and other methods. For some t, all three curves are low, corresponding to times when the speaker pauses between words. Other fluctuations for this technique, however, are due to stereo tracking errors and other biases of the stereo system or microphone array.

To evaluate the audio-only approach, we search the room for the location of maximum acoustic power and steer the array to that location. For our test scenario, this worked quite well when tracking the louder speaker. Even so, there are several points in time where the array locks onto the interfering speaker. When attempting to track the quieter speaker, this method fails completely.

The fourth entry in Table 1 and Figure 5 uses the stereo tracker's location estimate as the initial guess from which to perform gradient ascent in the signal output power. This technique's average SNR is well above that of either the single-microphone or video-only methods, and its short-time SNRs are consistently highest or nearly the highest of any of the four techniques.

These experiments demonstrate that audio-video localization is superior to video alone in our environment. We believe our approach improves upon audio-only localization in cases where there are multiple simultaneous speakers and the reverberant energy is nearly equal or greater than the direct path energy. The initial position estimate provided by video localization reduces the amount of computation required compared to an unconstrained audio-only search.

5 Conclusion

We have implemented a computationally efficient hybrid sound source localization scheme. This scheme makes use of the complementary information available in the audio and video streams available in our test environment and is suitable for use as part of perceptive environments requiring high-quality audio signals for higher-level applications such as automated speech recognition.

In the future, we plan to incorporate more sophisticated beamforming techniques into our system to further improve the SNR of the output. In addition, we hope to be able to feed the final results of the audio-video localization back to the vision subsystem to allow it to refine its location and trajectory estimates.

6 Acknowledgments

We would like to thank the NIST Smart Spaces Laboratory for the use of their microphone array infrastructure, upon which our microphone array is based. We would also like to thank Corinna Sherman for her help in conducting the experiments.

References

1. D. J. Beymer and K. Konolige. Real-time tracking of multiple people using stereo. In *Frame-Rate Workshop*, 1999.
2. U. Bub, M. Hunke, and A. Waibel. Knowing who to listen to in speech recognition: Visually guided beamforming. In *1995 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1995.
3. M. Casey, W. Gardner, and S. Basu. Vision steered beam-forming and transaural rendering for the artificial life interactive video environment, (alive). In *99th Convention of the Audio Engineering Society*, 1995.
4. M. Collobert, R. Feraud, G. LeTourneur, O. Bernier, J. E. Viallet, Y. Mahieux, and D. Collobert. Listen: a system for locating and tracking individual speakers. In *2nd International Conference on Face and Gesture Recognition*, 1996.
5. T. Darrell, D. Demirdjian, N. Checka, and P. Felzenszwalb. Plan-view trajectory estimation with dense stereo background models. In *2001 International Conference on Computer Vision*, 2001.
6. T. Darrell, G. G. Gordon, M. Harville, and J. Woodfill. Integrated person tracking using stereo, color, and pattern detection. *IJCV*, (37(2)):199–207, June 2000.
7. R. Duraiswami, D. Zotkin, and L. S. Davis. Active speech source localization by a dual course-to-fine search. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2001.
8. Y. A. Ivanov, A. F. Bobick, and J. Liu. Fast lighting independent background subtraction. *IJCV*, 2000.
9. J. Krumm, S. Harris, B. Meyers, B. Brummit, M. Hale, and S. Shafer. Multi-camera multi-person tracking for easyliving. In *3rd IEEE Workshop on Visual Surveillance*, 2000.
10. H. F. Silverman, W. R. Patterson, and J. L. Flanagan. The huge microphone array. *IEEE Concurrency*, pages 36–46, October 1998.
11. Barry D. Van Veen and Kevin M. Buckley. Beamforming: A versatile approach to spatial filtering. *IEEE ASSP Magazine*, April 1988.
12. M. Viberg and H. Krim. Two decades of statistical array processing. In *31st Asilomar Conference on Signals, Systems, and Computers*, 1997.
13. C. Wang and M. Brandstein. Multi-source face tracking with audio and visual data. In *IEEE International Workshop on Multimedia Signal Processing*, 1999.

Proceedings of the International
Conference on Computer Vision,
Vancouver, Canada, 2001.

Reducing Drift in Parametric Motion Tracking

Ali Rahimi, Louis-Philippe Morency and Trevor Darrell

MIT Artificial Intelligence Laboratory
{`rahimi, lmorency, trevor`}`@ai.mit.edu`

Abstract. We develop a class of differential motion trackers that automatically stabilize when in finite domains. Most differential trackers compute motion only relative to one previous frame, accumulating errors indefinitely. We estimate pose changes between a set of past frames, and develop a probabilistic framework for integrating those estimates. We use an approximation to the posterior distribution of pose changes as an uncertainty model for parametric motion in order to help arbitrate the use of multiple base frames. We demonstrate this framework on a simple 2D translational tracker and a 3D, 6-degree of freedom tracker.

1 Introduction

Tracking the pose of an object requires that image transformation parameters be recovered for each frame of a video sequence. A common class of approaches for estimating these parameters involves accumulating motion parameters between pairs of temporally adjacent frames. These differential techniques suffer from accumulated drift which limits their effectiveness when dealing with long video sequences. The proposed method reduces this drift by anchoring each frame to many past frames. We then use a maximum likelihood formalism to fuse these pose change estimates to obtain poses which exhibits less error.

Various methodologies for avoiding drift have been proposed. For example, [2] and [5] compute the pose of an object by bringing it into registration with the first frame in the video sequence. This approach restricts the range of appearances to be near the initial pattern unless complicated model acquisition techniques are employed. Another approach is to use subject-independent models that are refined over time

([1, 9]), but the accuracy of these methods is often limited by the coarseness of their models, though strong prior motion models can sometimes be used to obtain better accuracy (eg, [14]).

In this paper we show how typical differential tracking algorithms can be stabilized without changing the core structure of the tracker. We relax the restriction that only temporally adjacent frames will be used for differential tracking, allowing high-quality pose change measurements to compensate for poor quality ones. We compute pose changes between each frame and several anchor frames that are close in pose and appearance to it. These differential motion estimates are then combined to provide a robust estimate of pose for each frame. Conceptually, previous frames are used as an image-based model of the object being tracked, alleviating the need to construct an explicit model of the scene as is done in [11] and [4], for example.

The next section provides a maximum likelihood framework for differential tracking. We then augment this model to incorporate additional anchor frames. In order to find the maximum likelihood poses in this augmented model, it is necessary to measure the uncertainty in each pose estimate, so we develop an error measure for parametric pose estimation. We then discuss details involved in implementing our algorithm and apply our framework to a simple 2D tracking problem where camera motion is restricted to fronto-parallel translation over a synthetic planar object. Expeirments in sections 4.1 and 4.2 show how to augment the 6-DOF tracker of [3] with our framework and demonstrate its use in tracking heads through large rotations and computing egomotion in long sequences.

2 Differential tracking as maximum likelihood

We propose a measurement model suitable for representing differential trackers. We then frame our drift-reduced tracker in this model by adding additional measurement nodes. In order to cast tracking as a maximum likelihood problem, we develop an error model for estimating parametric pose change.

2.1 A measurement model

Consider a sequence of images $y_0 \cdots y_t$ with associated object poses $\xi_0 \cdots \xi_T$. Let $\delta_0^1 = d(\xi_1, \xi_0)$ be the pose change between frames with pose ξ_0 and ξ_1. If the parametrization is additive, d just subtracts ξ_0 from ξ_1. In the affine case, d computes $A^{-1}[A(\xi_1)A(\xi_0)^{-1}]$, where A

returns a 3x3 affine matrix given a 6 dimensional vector, and A^{-1} returns the six parameters of the affine transformation given an affine matrix. We also define d^{-1} such that $d^{-1}(d(\xi_1, \xi_0), \xi_0) = \xi_1$. Estimating the pose change between frames y_{t-1} and y_t results in a pose difference δ_{t-1}^{t} with distribution $p(\delta_{t-1}^{t}|y_{t-1}, y_t)$.

Assuming that pose governs everything about appearance, δ_{t-1}^{t} is conditionally independent of y_{t-1} and y_t given ξ_{t-1} and ξ_t, so $p(\delta|y_{t-1}, y_t) = p(\delta|\xi_{t-1}, \xi_t)$[1]. Figure 1 depicts the resulting independence diagram for a differential tracker. The joint density of measurements $\{\delta\}$ and poses $\{\xi\}$ is

$$p(\{\xi\}, \{\delta\}) = p(\{\xi\}) \prod_{t=1}^{T} p(\delta_{t-1}^{t}|\xi_{t-1}, \xi_t)$$

Finding the set of ML poses $\{\xi\}$ involves computing

$$\begin{aligned} &\arg\max_{\{\xi\}} p(\{\xi\}|\{\delta\}) \\ &= \arg\max_{\{\xi\}} \sum_{t=1}^{T} \ln p(\delta_{t-1}^{t}|\xi_{t-1}, \xi_t) \end{aligned} \tag{1}$$

We can show that the traditional method of computing pose changes and updating pose estimates is in fact the ML solution by assuming that the performance of the tracker depends only on pose change and not on absolute pose. As a result, $p(\delta|y_{t-1}, y_t) = p(\delta|d(\xi_t, \xi_{t-1}))$. Making a final Gaussianity assumption on the posterior, we obtain:

$$p(\delta|\xi_t, \xi_{t-1}) = N(\delta_{t-1}^{t}; d(\xi_t, \xi_{t-1}), \Lambda_{t,t-1}). \tag{2}$$

Equation (1) can now be rewritten as

$$\arg\min_{\{\xi\}} \sum_{t=1}^{T} \|\delta_{t-1}^{t} - d(\xi_t, \xi_{t-1})\|_{\Lambda_{t,t-1}}. \tag{3}$$

The minimum value for this problems is 0, and occurs when

$$\begin{aligned} \delta_{t-1}^{t} &= d(\xi_t, \xi_{t-1}) \\ \xi_t &= d^{-1}(\delta_{t-1}^{t}, \xi_{t-1}), \end{aligned} \tag{4}$$

---

[1] This implies that given the pose, there is no other source of uncertainty in the appearance of a frame. As will be shown later, imager noise is funnelled into $p(\delta|y_{t-1}, y_t)$ by other means, alleviating the need for a cumbersome integration step here.

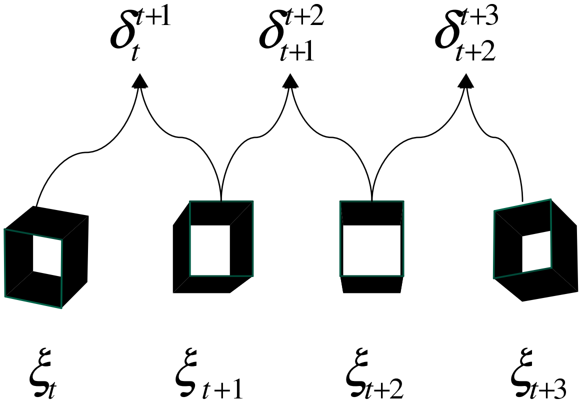


Fig. 1. Independence diagram for a simple pose tracker. The tracker measures pose differences $\{\delta\}$ between adjacent frames.

confirming that the traditional update equation does indeed maximize likelihood given the simplifying assumptions we've made. Note that $\Lambda_{t,t-1}$ drops out of the optimization, and so it is not necessary to compute the error in pose changes.

2.2 Using multiple base frames to reduce drift

To improve pose estimation, we invoke two principal insights:

1. When the trajectory comes close to crossing itself (ie, $\xi_t \approx \xi_s, t > s$), tracking should be performed between frames y_t and y_s as well.
2. Information about the pose of future frames can be used to adjust the pose estimate of past frames.

Proposition 1) provides redundant reliable information which allows us to better estimate pose. Proposition 2) is appealing since returning near a previously visited point can disambiguate measurements if information from the future is allowed to affect the past. Hence, in figure 2, we would do well to compute a pose change estimate between y_t and all frames that lie in the shaded region, and allow these measurements to influence the pose of frames $y_1 \cdots y_t$.

We augment the measurement model laid out in the previous section to incorporate these additional measurements. To improve performance, we can also incorporate knowledge about the dynamics of the pose parameters. Figure 3 shows how to update the graphical model of

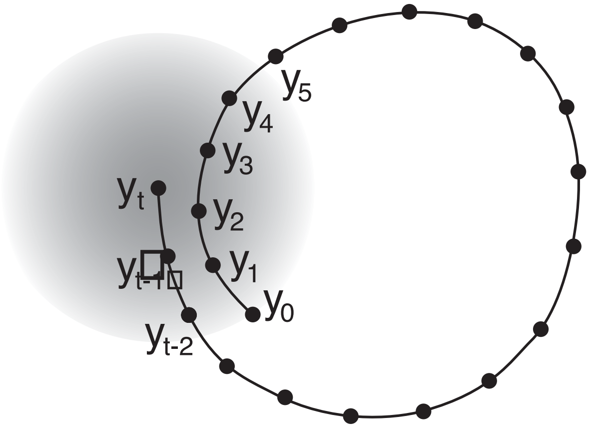


Fig. 2. When estimating the pose of frame y_t, we should take into account the pose change between y_t and y_{t-1} as well as all other frames which are in the shaded region.

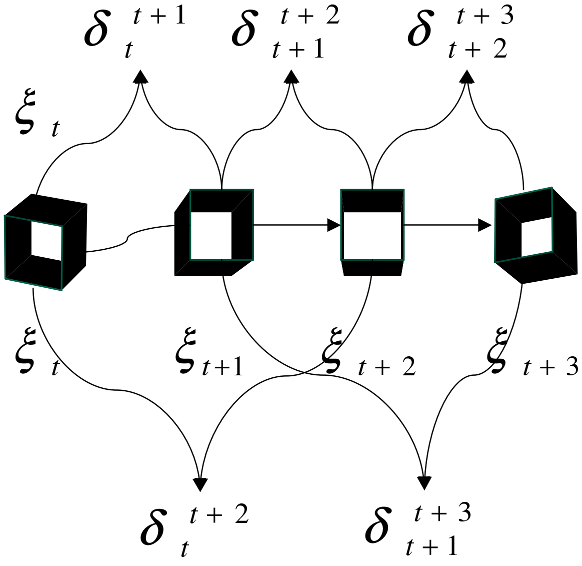


Fig. 3. The measurement model when multiple base frames are used. A dynamical model for pose change is also added (horizontal arrows).

the differential tracker to incorporate the added information. The joint of the poses and observations becomes

$$p(\{\xi\},\{\delta\}) = p(\xi_0)\prod_{t=1}^{T} p(\xi_t|\xi_{t-1}) \prod_{(f,g)\in D} p(\delta_f^g|\xi_f,\xi_g)$$

where D is the set of pairs of frames between which we have calculated the pose change. Using the Gaussian uncertainty model of (2), the ML poses are

$$\begin{aligned}\arg\min_{\{\xi\}} \quad & \sum_{(f,g)\in D} \|\delta_{t-1}^t - d(\xi_t,\xi_{t-1})\|_{\Lambda_{t,t-1}} \\ & + \sum_{t=1}^{T} \|d(\xi_t,\xi_{t-1})\|_{\Lambda_d} \qquad (5)\end{aligned}$$

where we have assumed that the pose dynamics are Brownian with covariance Λ_d. The optimization problem can be thought of as relaxing a spring system where the natural length of a spring between nodes ξ_f and ξ_g is δ_f^g and its stiffness is $\Lambda_{f,g}^{-1}$.

Unlike the minimization problem of the traditional tracker, we now need to know $\Lambda_{f,g}$. An approximation to $\Lambda_{f,g}$ is derived in the following two sections.

2.3 Estimating pose change

The simplest pose change tracker computes the maximum likelihood pose difference $\hat{\delta}_{t-1}^t$ by assuming that y_t can be warped back to y_{t-1}. Camera noise and any change in appearance that is not modelled by warping is modelled with identically distributed and independent Gaussian noise of unspecified variance added to every pixel. The generative model of y_{t-1} is then:

$$\begin{aligned} y_{t-1}(x) &= y_t(x - u(x;\delta_{t-1}^t)) + \mathbf{w}(x) \\ p(y_{t-1}(x)|y_t(x),\delta_{t-1}^t) &= \mathcal{N}(y_{t-1}(x); \\ &\qquad y_t(x - u(x;\delta_{t-1}^t)),\sigma_w^2) \end{aligned} \qquad (6)$$

where $\mathbf{w}(\mathbf{x})$ is Gaussian and white over space and time, and has constant variance σ_w^2 over the image. $\mathcal{N}(x;\mu,\sigma^2)$ is a Gaussian distribution with means μ and variance σ^2. $u(x;\delta)$ is the warping function: it is used to displace a pixel at location x to location $x + u(x;\delta)$ in the target image. The ML estimate, $\hat{\delta}$, maximizes the posterior $p(\delta|y_t, y_{t-1})$. This is equivalent to minimizing a sum-of-squared error function over δ:

$$\begin{aligned}\hat{\delta} &= \operatorname*{argmax}_{\delta} p(\delta|y_t, y_{t-1}) \\ &= \operatorname*{argmin}_{\delta} \sum_x [y_{t-1}(x) - y_t(x - u(x;\delta))]^2\end{aligned} \tag{7}$$

This is the traditional least squares formulation for tracking, derived in a probabilistic framework. Various total-least squares formulations which allow y_t to be noisy as well have been proposed [13, 8]. We have demonstrated that pose change estimation computes the mode of the distribution $p(\delta|y_t, y_{t-1})$. To fully qualify this distribution, we still need to compute its covariance $\Lambda_{t,t-1}$.

2.4 Uncertainty in motion estimates

Probabilistic methods for computing uncertainty in optical flow have been proposed in [12, 8]. We approximate the posterior $p(\delta|y_t, y_{t-1})$ by fitting a Gaussian distribution at the mode $\hat{\delta}$ computed by the pose estimator. The derivation is based on the approximation made in Laplace's method (see [6] for a note on the subject).

Using Bayes rule, we can rewrite the log-posterior:

$$\begin{aligned}\log p(\delta|y_t, y_{t-1}) = &\log p(y_{t-1}|\delta, y_t) \\ &+ \log p(\delta|y_t) - \log p(y_{t-1}|y_t)\end{aligned} \tag{8}$$

Since $\hat{\delta}$ is taken to be the ML estimate, the first derivative of (8) vanishes at $\hat{\delta}$. Assuming uniform $p(\delta|y_t)$ (this is the case if $p(\delta)$ is itself uniform, since we can glean nothing about future poses from a single image), the Hessian of (8) becomes

$$\mathbf{H} = \frac{\partial^2}{\partial\delta^2} \log p(\delta|y_t, y_{t-1}) = \frac{\partial^2}{\partial\delta^2} \log p(y_{t-1}|y_t, \delta)$$

The Taylor expansion of (8) about its mode is therefore:

$$\begin{aligned}\log p(\delta|y_t, y_{t-1}) \approx &\log p(\hat{\delta}|y_t, y_{t-1}) \\ &+ \frac{1}{2}\Delta\delta^T \frac{\partial^2}{\partial\delta^2} \log p(y_{t-1}|y_t, \hat{\delta})\Delta\delta \\ &+ \text{H.O.T.},\end{aligned}$$

where $\Delta\delta = \delta - \hat{\delta}$. Dropping high order terms and exponentiating, we obtain a Gaussian approximation to the posterior:

$$p(\delta|y_t, y_{t-1}) \approx \alpha \exp\left(\frac{1}{2}(\delta - \hat{\delta})^T \mathbf{H}(\delta - \hat{\delta})\right)$$

This Gaussian has mean $\hat{\delta}$ as expected, and its variance is $\Lambda_{t,t-1} = -\mathbf{H}^{-1}$. In this case, $\mathbf{H}$ is the Hessian of the log of

$$\begin{aligned} p(y_{t-1}|y_t,\hat{\delta}) &= \prod_x p(w_t(x)) \\ &= \prod_x N(y_{t-1}(x); y_t(x-u(x;\hat{\delta})), \sigma_w^2), \end{aligned}$$

which is found to be

$$\mathbf{H} = \frac{1}{\sigma_w^2}\sum_x \frac{\partial u}{\partial \delta}^T (\tilde{y}_{t-1}\nabla_x^2 \hat{y}_{t-1} - \nabla \hat{y}_t \nabla \hat{y}_t^T)\frac{\partial u}{\partial \delta},$$

where $\hat{y_{t-1}} = y_t(x-u(x;\hat{\delta}))$ is the reconstructed y_{t-1} and $\tilde{y}_{t-1}(x) = y_{t-1}(x) - \hat{y}_{t-1}(x)$ is the reconstruction residual[2]. Since in practice the reconstruction error is small, we can further approximate $\mathbf{H}$ by:

$$\mathbf{H} = -\frac{1}{\sigma_w^2}\sum_x \frac{\partial u}{\partial \delta}^T \nabla y_{t-1}(x)[\nabla y_{t-1}(x)]^T)\frac{\partial u}{\partial \delta},$$

Finally, σ_w^2 can be estimated as

$$\hat{\sigma}_w^2 = \frac{1}{N}\sum_x [y_{t-1}(x) - y_t(x-u(x;\hat{\delta}))]^2. \tag{9}$$

Our final estimate of the variance of $p(\delta|y_t, y_{t-1})$ is:

$$\Lambda_{t,t-1} = \hat{\sigma}_w^2 \left[\sum_x \frac{\partial u}{\partial \delta}^T \nabla y_{t-1}(x) \nabla y_{t-1}(x)^T \frac{\partial u}{\partial \delta}\right]^{-1}.$$

This expression has an intuitive interpretation which makes it suitable as an approximation of the posterior covariance. $\hat{\sigma}_w^2$ can be interpreted as the RMS reconstruction error after warping according to the recovered pose change. $\mathbf{H}$ can be interpreted as the average sensitivity of each component of δ, weighted by the strength of the features in the image. This is because $\nabla y(x) \nabla y(x)^T$ represents the strength of a feature at location x (see [10]), and $\frac{\partial u}{\partial \delta}(x;\delta)$ is a measure of the sensitivity of δ at various points in the image.

To illustrate this point, we compute the sensitivity of a translational and an affine tracker. In the translational case, $u(x;\delta) = \delta$. So

---

[2] In deriving this expression, we have assumed that $\partial^2 u/\partial \delta^2 = 0$. ie, u is linear wrt δ.

$\frac{\partial}{\partial\delta}u(x;\delta) = \mathbf{I}$. The covariance becomes

$$\mathbf{\Lambda}_{translation} = \hat{\sigma}_w^2 \left[\sum_x \nabla y \nabla y^T \right]^{-1}, \tag{10}$$

which is just the reconstruction error weighted by a measure of how textured the image is.

In the case of an affine tracker, the partial of u is:

$$\frac{\partial}{\partial\delta}u(x;\delta) = \begin{bmatrix} x\; y\; 1\; 0\; 0\; 0 \\ 0\; 0\; 0\; x\; y\; 1 \end{bmatrix}.$$

If we set $\nabla y_{t-1}(x) \nabla y_{t-1}(x)^T = \mathbf{I}$, effectively assigning to all points the same feature properties, the covariance becomes

$$\mathbf{\Lambda}_{affine} = \hat{\sigma}_w^2 \left[\sum_x \begin{bmatrix} \begin{matrix} x^2 & xy & x \\ xy & y^2 & y \\ x & y & 1 \end{matrix} & \mathbf{0} \\ \mathbf{0} & \begin{matrix} x^2 & xy & x \\ xy & y^2 & y \\ x & y & 1 \end{matrix} \end{bmatrix} \right]^{-1}.$$

According to this expression, points away from the center of the coordinate system reduce the uncertainty in the multiplicative portion of the affine transformation more than the central points. In addition all points contribute equally to the translation parameters. Both observations are consistent with our expectation.

3 Results: a simple 2D tracker

We first show results when tracking the position of an aperture moving over an image. ξ_t represents the current pixel location of the aperture and y_t denotes the image captured through the aperture. Since ξ only parametrizes translation, a simple motion model with $u(x;\delta) = \delta$ is adequate. Figure 4 shows the pose estimates from a differential tracker which finds pose changes by minimizing (8) using gradient descent. The update is according to (4) and is additive.

The algorithm estimates the pose change between consecutive 50x50 pixel windows which translate by an average of 5.6 pixels each step along a spiral path. The average error in estimating δ is around 0.66 pixels, which after 626 iterations, results in approximately 55 pixels of drift.

To measure the uncertainty of the pose change estimator, we used the pose covariance from equation (10). Figure 4 displays tracking performance on the same aperture trajectory. The previous frame was always used as an anchor frame, along with the 3 past frames which were closest in pose to the previous frame. In 626 frames, tracking drifts by at most 2.44 pixels and is off by 0.11 pixels at frame 623. Figure 5 compares the pose error of the two trackers over time. The the drift-reduced tracker stops accumulating error after about 50 frames, while the unenhanced tracker continues drifting.

To find the poses which maximize equation (5), we computed the derivatives of the log-likelihood with respect to each pose:

$$\begin{aligned} 0 &= \frac{\partial}{\partial \xi_i} p(\{\delta\}|\{\xi\}) \\ &= - \sum_{(f=i,g)\in D} \Lambda_{i,g}(\delta_i^g - \xi_i + \xi_g) \\ &+ \sum_{(f,g=i)\in D} \Lambda_{f,i}(\delta_f^i - \xi_f + \xi_i) \end{aligned} \tag{11}$$

Equation (11) is a sparse linear system in terms of the poses. Given a fixed value for ξ_0, this system can be solved very efficiently (Matlab's backslash operator, which uses simple Gaussian elimination solves the above 626 frame problem in less than a second).

4 Stabilized 3D tracking

Our method can also be applied to 3D tracking. We show results using a rigid motion tracker with integrated intensity and depth constraints, but our method is applicable to any parametric motion formulation, with or without depth constraints.

Depth constraints have been shown to increase the accuracy of gradient-based rigid motion tracking [3]. A depth constancy constraint analogous to the traditional brightness constancy constraint can be derived and yields:

$$\begin{aligned} -I_t &= \left[I_x \; I_y \right] u(x;\delta) \\ -Z_t &= \left[Z_x \; Z_y \right] u(x;\delta) - V_z(x,\delta) \end{aligned} \tag{12}$$

where I_x, Z_x, etc, are the partials of y_t or y_{t-1}. Together, these equations constrain the local motion δ_{t-1}^t by using the image gradients.

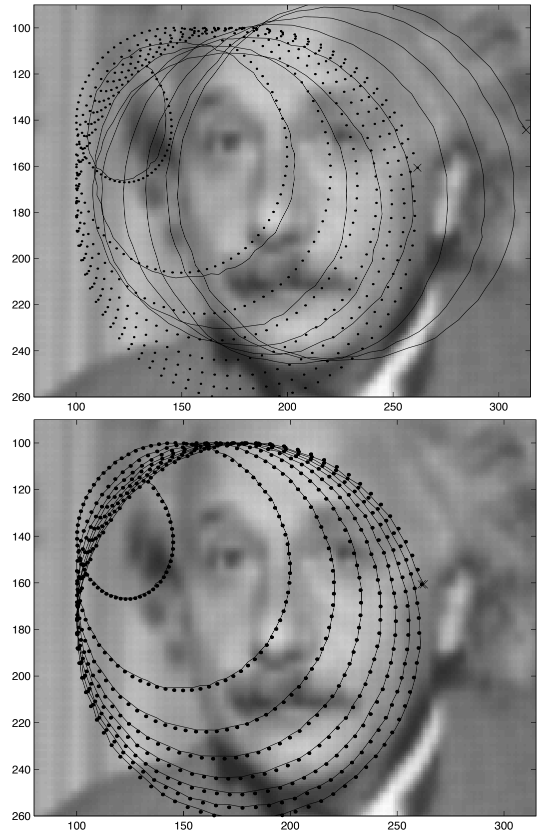


Fig. 4. Estimates of the position of a 50x50 pixel aperture as it follows a spiral path on the image. The position estimate is based solely on the image acquired through the aperture. Top: traditional tracker. The estimated trajectory (solid) terminates (marked by '×') with more than 55 pixels of error relative to ground truth (dotted). Bottom: drift-reduced tracker, using at most 4 past frames. The estimated trajectory ends less than 1 pixels from the ground truth.

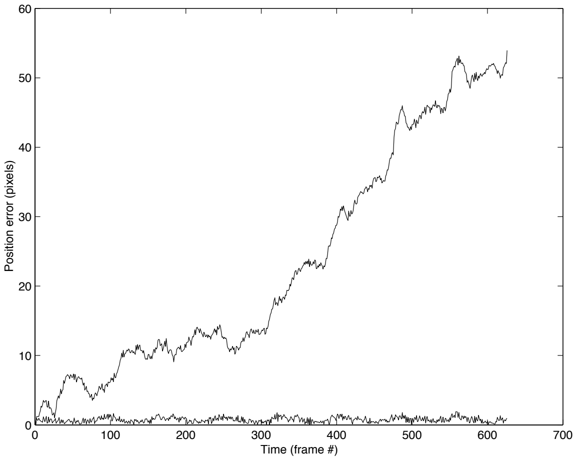


Fig. 5. Comparison of position error between simple tracker and drift-reduced tracker.

When the camera model is perspective, a velocity $[V_X, V_Y, V_Z]^T$ at a location in the real world results in image flow

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} f & 0 & -x \\ 0 & f & -y \end{bmatrix} \begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} .$$

In the case of 3D motion, we define $\delta = [\delta_\omega \delta_\Delta]$ where the three components of δ_ω specify infinitesimal rotation and the three components of δ_Δ specify translation. The warping function becomes:

$$u(x;\delta) = \frac{1}{Z} \begin{bmatrix} f & 0 & -x_1 \\ 0 & f & -x_2 \end{bmatrix} (\delta_\omega \times X + \delta_\Delta)$$

where X is the world coordinate of the image point x. Isolating δ and plugging u back into (12):

$$\begin{aligned} -Z_t &= \frac{1}{Z} \begin{bmatrix} fZ_x & fZ_y & -(Z + xZ_x + yZ_y) \end{bmatrix} \mathbf{Q}\delta \\ -I_t &= \frac{1}{Z} \begin{bmatrix} fI_x & fI_y & -(xI_x + yI_y) \end{bmatrix} \mathbf{Q}\delta \end{aligned} \tag{13}$$

with

$$\mathbf{Q} = \begin{bmatrix} \mathbf{I} & -\hat{X} \end{bmatrix} ,$$

where $\hat{X}$ is the 3x3 skew symmetric matrix formed by the real-world coordinates corresponding to x and $\mathbf{I}$ is the 3x3 identity matrix. The system of equation (13) is linear and highly overconstrained and can be easily solved for δ.

For infinitesimal 3D updates, $d(\xi_1, \xi_0)$ should be the real eigenvector of $e^{\hat{\xi}_1} e^{-\hat{\xi}_0}$ [7], but we have found that $d(\xi_1, \xi_0) = \xi_1 - \xi_0$ is adequate in practice. Drift reduction then consisted in solving equation (11) using a sparse linear system solver.

4.1 Results: 6-DOF head tracker

We demonstrate the performance of the drift reduction algorithm on this 3D tracker. Figure 6 describes the direction of a head as the subject looks around the room. The nose moves by at most 20 cm throughout the sequence and the head yaws by up to 80 degrees in each direction and pitches by up to a total of 55 degrees. The sequence is 800 frames long and corresponds to about 1.2 minutes of video. The face was segmented from the background using the depth information only. Pose changes were computed using the combined constraints of (13). As shown in figure 7, after about 600 frames, the traditional tracker has accumulated noticeable drift in its estimate of rotation, whereas the drift-reduced tracker shows the pointer on the subject's nose whenever he returns to a near-frontal pose. Only appearance was used in finding suitable anchor frames. Figure 8 plots the index of anchor frames used for each frame. The protrusions from the diagonal line are produced as the subject returns from a rotation. Note that the first frame is never reused. The robustness is entirely due to recovering from drift accumulated during each rotation by using frames observed while going into the rotation.

4.2 Results: egomotion

The sequence summarized in figure 9 demonstrates that the drift reduced tracker can also be used for computing ego-motion. The task is to hold the pointer in the same location relative to the real world as the camera scans the room. Between frames 400 and 600, almost none of the original scene is visible. By frame 610, the drift-reduced tracker shows significant improvement over the traditional tracker, despite the dearth of back frames before frame 630. The superior performance in the early frames demonstrates the benefits of the batch/non-causal nature of the drift-reduction algorithm and of allowing information in the future influence the past. By frame 1050 the unenhanced tracker has

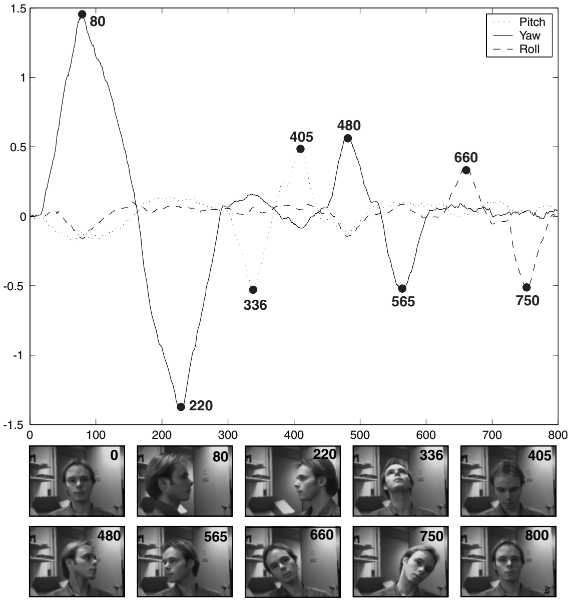


Fig. 6. The sequence is 1.2 minutes long. The subject looks in all directions, by up to 80 degrees from frontal in some directions. The sequence was captured at ∼11 fps. The graph above provides an intuitive feel for the relative magnitude of the rotations. It plots δ_ω over time.

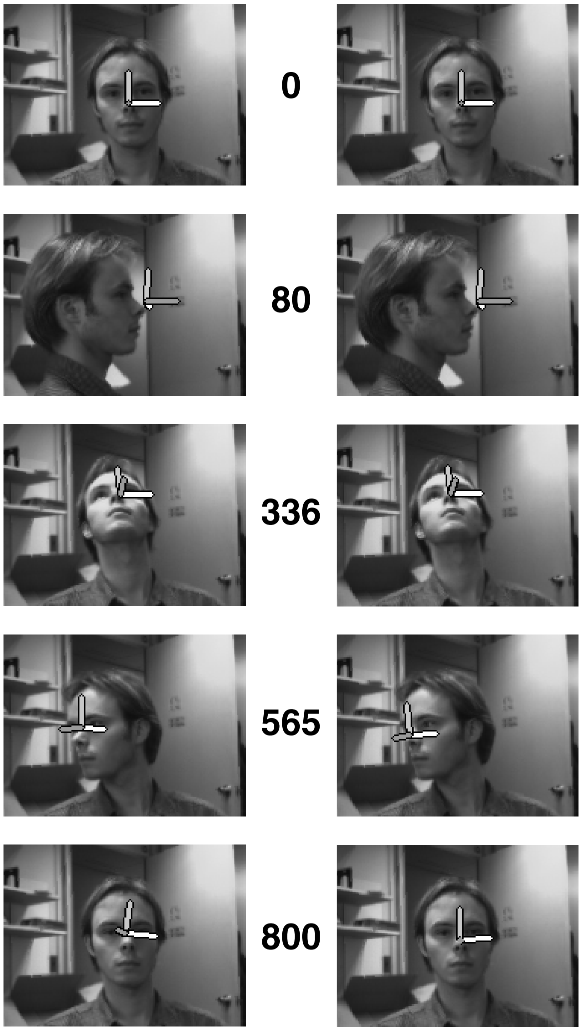


Fig. 7. Left column: traditional tracker. Poses are updated according to (4). Right column: drift-reduced tracker.

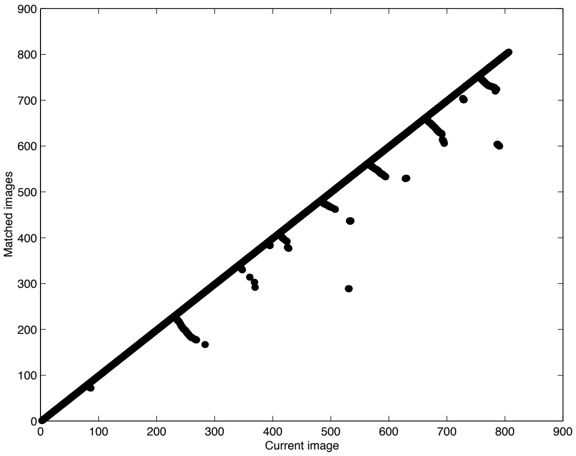


Fig. 8. Anchor frames used by drift-reduced tracker. Each frame on the horizontal axis is matched by appearance with 3 previous frame. The protrusions show that as the subject returns from a rotation, frames on the way into the rotation are used as anchor.

drifted far enough that all subsequent pose changes throw it even further off track. Figure 10 shows a quantitive version of the results. After 600 frames, the traditional tracker starts to accumulate considerable drift. During the same period, the drift-reduced tracker keeps track of the real movement by using information prom similar previous frames as shown in figure 12.

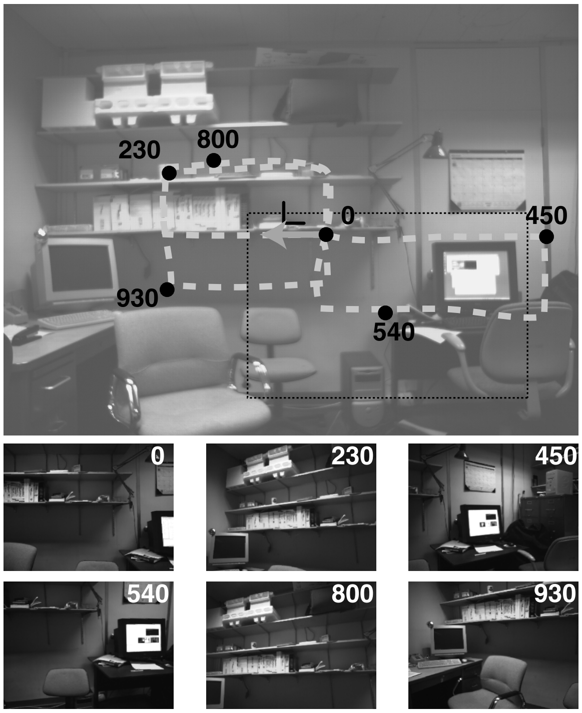


Fig. 9. The camera begins panning from the center dot, in the direction of the arrow. The dashed path marks the approximate trajectory of the center of the camera (drawn by hand). Only the interior of the black rectangle is visible to the camera (approximate), so that the intial pose is completely out of view between frames 420 and 530.

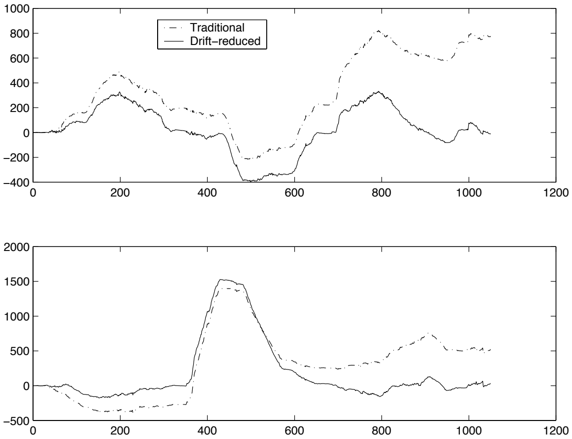


Fig. 10. Top: Horizontal translation. Bottom: Vertical Translation. The traditional tracker exhibits continual drift with respect to the drift-reduced tracker.

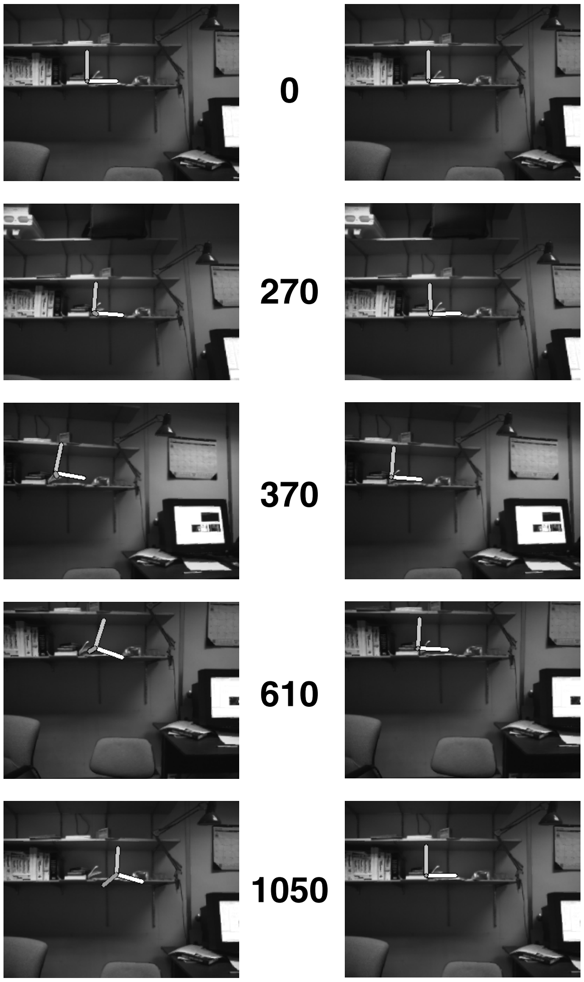


Fig. 11. Left column: traditional tracker. Right column: drift-reduced tracker. Beyond 1050 frames, the traditional tracker is no longer effective.

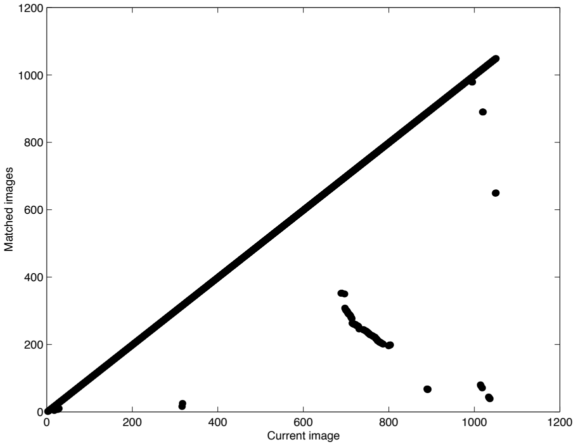


Fig. 12. Anchor frames used by drift-reduced tracker. Previously visited poses are used effectively (eg, frames 300, 700, 1020).

5 Conclusion

We have developed a framework for stabilizing parametric motion trackers in closed environments. Our method measures pose change between frames which are similar in pose and appearance, and uses these measurements to compute robust pose estimates. This improves stability since additional pose change measurements provide robustness and ground the tracking against commonly revisited sites. We derived an uncertainty model for motion estimation and used it to frame the problem of incorporating these additional measurements into a non-causal estimation framework. We demonstrated the benefits of using multiple base frames in our maximum likelihood framework on a synthetic 2D motion tracking problem and on 3D ego-motion computation and pose estimation.

6 Acknowledgement

Many thanks to Tom Minka for suggesting the Hessian approximation used in section 2.4.

References

1. S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *CVPR*, pages 232–237, 1998.
2. G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *PAMI*, 20(10):1025–1039, 1998.
3. M. Harville, A. Rahimi, T. Darrell, G. Gordon, and J. Woodfill. 3d pose tracking with linear depth and brightness constraints. In *Proceedings of CVPR 99*, pages 206–213, Corfu, Greece, 1999.
4. T. Jebara and A. Pentland. Parametrized structure from motion for 3d adaptive feedback tracking of faces. In *CVPR*, 1997.
5. M. LaCascia, S. Sclaroff, and V. Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of textured-mapped 3d models. *PAMI*, 22(4):322–336, April 2000.
6. T.P. Minka. Using lower bounds to approximate integrals. Technical report, Media Lab, http://www.media.mit.edu/~tpminka/papers/rem.html, 2001.
7. Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, 1994.
8. O. Nestares, D.J. Fleet, and D.J. Heeger. Likelihood functions and confidence bounds for total-least-squares problems. In *CVPR*, 2000.
9. N. Oliver, A. Pentland, and F. Berard. Lafter: Lips and face real time tracker. In *Computer Vision and Patt. Recog.*, 1997.

10. J. Shi and C. Tomasi. Good features to track. In *CVPR94*, pages 593–600, 1994.
11. H.-Y. Shum and R. Szeliski. Construction of panoramic mosaics with global and local alignment. In *IJCV*, pages 101–130, February 2000.
12. E.P. Simoncelli, E.H. Adelson, and D.J. Heeger. Probability distributions of optical flow. In *CVPR91*, pages 310–315, 1991.
13. J. Weber and J. Malik. Robust computation of optical flow in a multiscale differential framework. *IJCV*, 14(1):67–81, 1995.
14. C. Wren and A. Pentland. Dynamic models of human motion. In *Proceedings of Face and Gestures*, 1998.

Proceedings of the 2001 Workshop on
Perceptual User Interfaces (PUI'01),
Orlando, FL, May 2001.

Signal Level Fusion for Multimodal Perceptual User Interface

John W. Fisher III and Trevor Darrell

MIT Artificial Intelligence Laboratory
{`fisher, trevor`}`@ai.mit.edu`

Abstract. Multi-modal fusion is an important, yet challenging task for perceptual user interfaces. Humans routinely perform complex and simple tasks in which ambiguous auditory and visual data are combined in order to support accurate perception. By contrast, automated approaches for processing multi-modal data sources lag far behind. This is primarily due to the fact that few methods adequately model the complexity of the audio/visual relationship. We present an information theoretic approach for fusion of multiple modalities. Furthermore we discuss a statistical model for which our approach to fusion is justified. We present empirical results demonstrating audio-video localization and consistency measurement. We show examples determining where a speaker is within a scene, and whether they are producing the specified audio stream.

1 Introduction

Multi-modal fusion is an important, yet challenging task for perceptual user interfaces. Humans routinely perform complex and simple tasks in which ambiguous auditory and visual data are combined in order to support accurate perception. In contrast, automated approaches for processing multi-modal data sources lag far behind. This is primarily due to the fact that few methods adequately model the complexity of the audio/visual relationship. Classical approaches to multi-modal fusion either assume a statistical relationship which is too simple (e.g. jointly Gaussian) or defer fusion to the decision level when many of the joint (and useful) properties have been lost. While such pragmatic choices may lead to simple statistical measures, they do so at the cost of modeling capacity.

We discuss a nonparametric statistical approach to fusion which jointly models audio-visual phenomena. Using principles from information theory we show an approach for learning maximally informative joint subspaces for multi-modal fusion. Specifically, we simultaneously learn projections of images in the video sequence *and* projections of sequences of periodograms taken from the audio sequence. The projections are computed adaptively such that the video and audio projections have maximum mutual information (MI). The approach uses the methodology presented in [2, 6, 4] which formulates a learning approach by which the entropy, and by extension the MI, of a differentiable map may be optimized. We also discuss a statistical model for which the approach can be shown to be optimal.

Combining audio and video signals for dialog interface applications is an important goal for perceptual user interfaces. There has been substantial progress on feature-level integration of speech and vision. However, many of these systems assume that no significant motion distractors are present and that the camera was "looking" at the user who was uttering the audio signal.

Indeed, speech systems (both those that integrate viseme features and those that do not) are easily confused if there are nearby speakers also making utterances, either directed at the speech recognition system or not. If a second person says "shut down" near a voice-enabled workstation, the primary user may not be pleased with the result! In general, it is clear that multi-modal cues can aid the segmentation of multiple speakers into separate channels (e.g. the "cocktail party" effect)..

In this paper we show how signal level fusion of audio and video data using nonparametric models can capture useful joint structure. Specifically, we show results on two tasks, one localizing a speaker in a video stream, and the second measuring audio/video consistency–whether the audio and video came from the same source.

1.1 Related work

As mentioned above, there has been much work on feature level audio-visual speech recognition. For example, Meier *et al* [9] and Stork [12] (and others) have built visual speech reading systems that can improve speech recognition results dramatically. It is not clear whether these systems could be used to localize the speaker as they implicitly rely on localization having already been performed. In theory, these systems could be modified to verify if the sequence of observed visemes was consistent with the detected phonemes. We are not aware of a system which has been reported to do this to date, though it may be a successful approach. The method we will present works at a pre-feature level and does not presume detection of phonemes or visemes, so it may be advantageous in cases where a person-independent viseme model is hard to obtain. Also, since our method is not dependent on speech content, it would have the advantage of working on non-verbal utterances.

Other work which is more closely related to ours is that of Hershey and Movellan [7] which examined the per-pixel correlation relative to an audio track, detecting which pixels have related variation. An inherent assumption of this method was that the joint statistics were gaussian. Slaney and Covell [11] looked at optimizing temporal alignment between audio and video tracks, but did not address the problem of detecting whether two signals came from the same person or not. Their technique was more general than [7] in that pixels changes were considered jointly, although there is also an implicit Gaussian assumption. Furthermore, this technique makes use of training data.

The idea of simply gating audio input with a face detector is related to ours, but would not solve our target scenerio above where the primary user is facing the screen and a nearby person makes an utterance that can be mistakenly interpreted as a system command. We are not aware of any prior work in perceptual user interfaces which addresses signal-processing level estimators to do both video localization and classify audio-visual synchrony among individuals.

2 Informative subspaces

We now give a brief description of our information theoretic fusion approach. While the algorithm has been described in previous work [3], that discussion focused primarily on the information theoretic intuition which motivated the method. In this section we also present a statistical model from which the method can be derived and the conditions under which our fusion approach is optimal.

2.1 Information theoretic fusion

Figure 1 illustrates our audio/visual fusion approach. Each image in the measured video sequence is treated as a single sample of a high-dimensional random variable (i.e. the dimension equals the number of pixels) . We denote ith image as V_i. The audio signal is converted to a sequence of periodograms (i.e. magnitude of windowed FFTs). Peridograms are computed at the video frame rate using a window equal to twice the frame period. Similarly to the video sequence, each periodogram "frame" is also treated as a sample of a high dimensional random variable (whose dimension is equal to the number of frequency bins) and whose ith frame is denoted U_i.

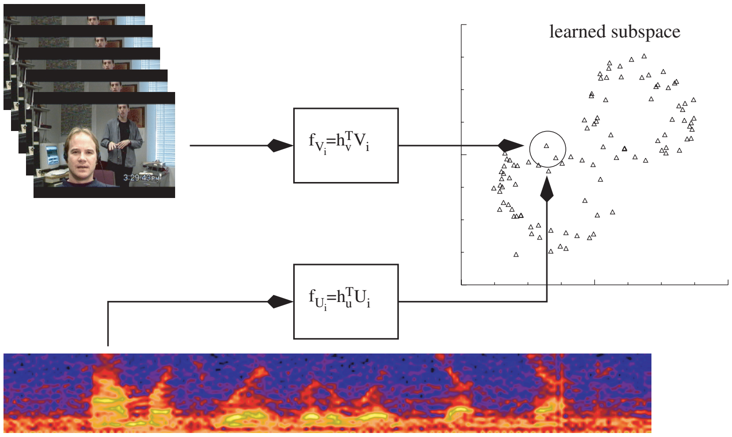


Fig. 1. Maximally Informative Joint Subspace

Using the approach described in [3] we learn projections of the audio/video frames, denoted,

$$f_{V\,i} = h_V^T V_i \tag{1}$$
$$f_{U\,i} = h_U^T U_i \tag{2}$$

resulting in samples of low-dimensional features $f_{V\,i}$ and $f_{U\,i}$ (whose dimensionality is determined by the matrices h_V and h_U, respectively). The criterion for learning the projection vectors, h_V and h_U, is to maximize the MI between the resulting audio and video features$f_{V\,i}$ and $f_{U\,i}$.

Mutual information (in the case of continuous features) is defined as [1]

$$\begin{aligned} I\left(f_V, f_U\right) &= h\left(f_U\right) + h\left(f_V\right) - h\left(f_U, f_V\right) \qquad (3) \\ &= \int_{R_U} p_{f_U}\left(x\right) \log\left(p_{f_U}\left(x\right)\right) dx + \\ &\quad \int_{R_U} p_{f_V}\left(x\right) \log\left(p_{f_V}\left(x\right)\right) dx - \end{aligned}$$

$$\int\int_{R_U \times R_V} p_{f_U,f_V}(x,y) \log\left(p_{f_U,f_V}(x,y)\right) dxdy$$

The difficulty of MI as a criterion for adaptation is that it is an integral function of probability densities. Furthermore, in general we are not given the densities themselves, but samples from which they must be inferred. Consequently, we replace equation 3 with the approximation of [6]

$$\begin{aligned}\hat{I}\left(f_V, f_U\right) &= \hat{H}(f_U) + \hat{H}(f_V) - \hat{H}(f_V, f_U) \\ &= \int_{R_U} \left(\hat{p}_{f_U}(x) - p_u(x)\right)^2 dx \\ &\quad + \int_{R_V} \left(\hat{p}_{f_V}(x) - p_u(x)\right)^2 dx \\ &\quad - \int_{R_U \times R_V} \left(\hat{p}_{f_V,f_U}(x,y) - p_u(x,y)\right)^2 dxdy\end{aligned} \tag{4}$$

where R_U is the support of one feature output, R_V is the support of the other, p_u is the uniform density over that support, and $\hat{p}(x)$ is the Parzen density [10] estimate computed from the projected samples:

$$\hat{p}\left(x\right) = \frac{1}{N} \sum_i \kappa\left(x - x_i, \sigma\right) \tag{5}$$

where $k\left(\ \right)$ is a gaussian kernel in our case and σ is the standard deviation.

Note that this is essentially an integrated squared error comparison between the density of the projections to the uniform density (which has maximum entropy over a finite region). The consequence of using this approximation is that its gradient with respect to the projection coefficients can be computed *exactly* by evaluating a finite number of functions at a finite number of sample locations in the output space as shown in [5, 6]. The update term for the individual entropy terms in 4 of the ith feature vector at iteration k as a function of the value of the feature vector at iteration $k-1$ is (where f_i denotes a sample of either f_U or f_V or their concatenation depending on which term of 4 is being computed)

$$\begin{aligned}\Delta {f_i}^{(k)} = b_r({f_i}^{(k-1)}) - \\ \frac{1}{N} \sum_{j \neq i} \kappa_a \left({f_i}^{(k-1)} - {f_j}^{(k-1)}, \sigma\right)\end{aligned} \tag{6}$$

$$b_r(f_i)_l \approx \frac{1}{d^M} \prod_{j \neq l} \left(\kappa_1 \left(f_{ij} + \frac{d}{2}, h \right) - \kappa_1 \left(f_{ij} - \frac{d}{2}, h \right) \right) \tag{7}$$

$$\kappa_a \left(f_i, \sigma \right) = \kappa(f_i, \sigma) * \kappa'(f_i, \sigma) \tag{8}$$

$$= -\frac{\exp\left(-\frac{{f_i}^T f_i}{4h^2} \right)}{\left(2^{M+1} \pi^{M/2} h^{M+2} \right)} f_i \tag{9}$$

where M is the dimensionality of the feature vector f_i. Both $b_r(f_i)$ and $\kappa_a(f_i, \sigma)$ are M-dimensional vector-valued functions and d is the support of the output of the mapping (i.e. a hyper-cube with sides of length d centered at the origin). The notation $b_r(y_i)_l$ indicates the lth element of $b_r(f_i)$ [6].

The process is repeated iteratively until a local maximum is reached using the update rule above. In the experiments that follow the dimensionality of f_U and f_V are set to unity while the number iterations is typically 150 to 300 iterations.

Capacity control The method of [6] requires that the projection be differentiable, which it is in this case. Additionally some form of capacity control is necessary as the method results in a system of underdetermined equations. In practice we impose an L_2 penalty on the projection coefficients of h_U and h_V. Furthermore, we impose the criterion that if we consider the projection h_V as a filter, it has low output energy when convolved with images in the sequence (on average). This constraint is the same as that proposed by Mahalanobis *et al* [8] for designing optimized correlators the difference being that in their case the projection output was designed explicitly while in our case it is derived from the MI optimization in the output space.

The adaptation criterion, which we maximize in practice, is then a combination of the approximation to MI (equation 4) and the regularization terms:

$$J = \hat{I}\left(f_V, f_U \right) - \alpha_v h_V^T h_V - \alpha_u h_U^T h_U - \beta h_V \bar{R}_V^{-1} h_V \tag{10}$$

where the last term derives from the output energy constraint and $\bar{R}_V^{-1}$ is average autocorrelation function (taken over all images in the sequence). This term is more easily computed in the frequency domain (see [8]) and is equivalent to pre-whitening the images using the inverse of the average power spectrum. The scalar weighting terms α_v, α_u, β, were set using a data dependent heuristic for all experiments.

The interesting thing to note is that computing h_v can be decomposed into three stages:

1. Pre-whiten the images **once** (using the average spectrum of the images) followed by iterations of
2. Updating the feature values, and
3. Solving for the projection coefficients using least squares and the L_2 penalty.

The pre-whitening interpretation makes intuitive sense in our case as it accentuates edges in the input image. It is the moving edges (lips, chin, etc.) which we expect to convey the most information about the audio. The projection coefficients related to the audio signal, h_U, are solved in a similar (and simultaneously) without the initial pre-whitening step.

2.2 The implicit statistical model

From the perspective of information theory, estimating separate projections of the audio video measurements which have high mutual information with respect to each other makes intuitive sense as such features will be predictive of each other. The advantage being that the form of those statistics are not subject to strong assumptions (e.g. joint gaussianity).

However, we now show that there is a statistical model for which such fusion is optimal. Consider the graphical models shown in figure 2. Figure 2a shows an independent cause model, where $\{A, B, C\}$ are unobserved random variables representing the causes of our (high-dimensional) observations $\{U, V\}$. In general there may be more causes and more measurements, but this simple case can be used to illustrate our algorithm. An important aspect is that the measurements have dependence on only one common cause. The joint statistical model consistent with the graph of figure 2a is

$$P(A, B, C, U, V) = P(A)P(B)P(C)P(U|A, B)P(V, B, C) \ .$$

Given the independent cause model a simple application of Bayes' rule (or the equivalent graphical manipulation) yields the graph of figure 2b which is consistent with

$$P(A, B, C, U, V) = P(U)P(C)P(A, B|U)P(V|B, C) \ ,$$

which shows that information about U contained in V is conveyed through the *joint* statistics of A and B. The consequence being that, in general, we cannot disambiguate the influences that A and B have on the measurements. A similar graph is obtained by conditioning on V. Suppose decompositions of the measurement U and V *exist* such that the following joint densities can be written:

$$\begin{aligned} P(A,B,C,U,V) = {} & P(A)P(B)P(C)P(U_A|A)P(U_B|B) \\ & P(V_B|B)P(V_C|C) \end{aligned}$$

where $U = [U_A, U_B]$ and $V = [V_B, V_C]$. An example for our specific application would be segmenting the video image (or filtering the audio signal). In this case we get the graph of figure 2c and from that graph we can extract the Markov chain which contains elements related only to B. Figure 2d shows equivalent graphs of the extracted Markov chain. As a consequence, there is no influence due to A or C.

Of course, we are still left with the formidable task of finding a decomposition, but given the decomposition it can be shown, using the data processing inequality [1], that the following inequality holds:

$$I(f_U, f_V) \leq I(f_U, B) \tag{11}$$

$$I(f_U, f_V) \leq I(f_V, B) \tag{12}$$

So, by maximizing the mutual information between $I(f_U, f_V)$ we must necessarily increase the mutual information between f_U and B and f_V and B. The implication is that fusion in such a manner discovers the underlying cause of the observations, that is, the joint density of $P(f_U, f_V)$ is strongly related to B. Furthermore, with an approximation, we can optimize this criterion without estimating the separating function directly. In the event that a perfect decomposition does not exist, it can be shown that the method will approach a "good" solution in the Kullback-Leibler sense.

3 Empirical results

We now present experimental results in which the general method described previously is used to first to localize the speaker in the video and second to measure whether the audio signal is consistent with the video signal. We collected audio-video data from eight subjects. In all cases the video data was collected at 29.97 frames per second at a resolution of 360x240. The audio signal was collected at 48000 KHz, but only 10Khz of frequency content was used. All subjects were asked to utter the phrase "How's the weather in Taipei?". This typically yielded 2-2.5 seconds of data. Video frames were processed as is, while the audio signal was transformed to a series of periodograms. The window length of the periodogram was 2/29.97 seconds (i.e. spanning the width of two video frames). Upon estimating projections the mutual information between the projected audio and video data samples is used as the measure of consistency. All values for mutual information are in terms of the maximum possible value, which is the value obtained (in the limit) if the two variables are uniformly distributed and perfectly predict one another. In all cases we assume that there is not significant head movement on the part of the speaker. While this assumption might be violated in practice one might account for head movement using a tracking algorithm, in which case the algorithm as described would process the images after tracking.

3.1 Video localization of speaker

Figure 3a shows a single video frame from one sequence of data. In the figure there is a single speaker and a video monitor. Thoughout the sequence the video monitor exhibits significant flicker. Figure 3c shows an image of the pixel-wise standard deviations of the image sequence. As can be seen, the energy associated with changes due to monitor flicker is greater than that due to the speaker. Figure 3b shows the absolute value of the output of the pre-whitening stage for the video frame in the same figure. Note that the output we use is signed. The absolute value is shown instead because it illustrates the enhancements of edges in the image.

Figure 5a shows the associated periodogram sequence where the horizontal axis is time and the vertical axis is frequency (0-10 Khz). Figure 3d shows the coefficients of the learned projection when fused with the audio signal. As can be seen the projection highlights the region about the speaker's lips.

Figure 4a shows results from another sequence in which there are two people. The person on the left was asked to utter the test phrase, while the person on the right moved their lips, but did not speak. This sequence is interesting in that a simple face detector would not be sufficient to disambiguate the audio and video stream. Furthermore, viseme based approaches might be confused by the presence of two faces.

Figures 4b and 4c show the pre-whitened images as before. There are significant changes about both subjects lips. Figure 4d shows the coefficients of the learned projection when the video is fused with the audio and again the region about the correct speaker's lips is highlighted.

3.2 Quantifying consistency between the audio and video

In addition to localizing the audio source in the image sequence we can also check for consistency between the audio and video. Such a test is useful in the case that the person to which a system is visually attending is not the person who actually spoke. Having learned a projection which optimizes MI in the output feature space, we can then estimate the resulting MI and use that estimate to quantify the audio/video consistency.

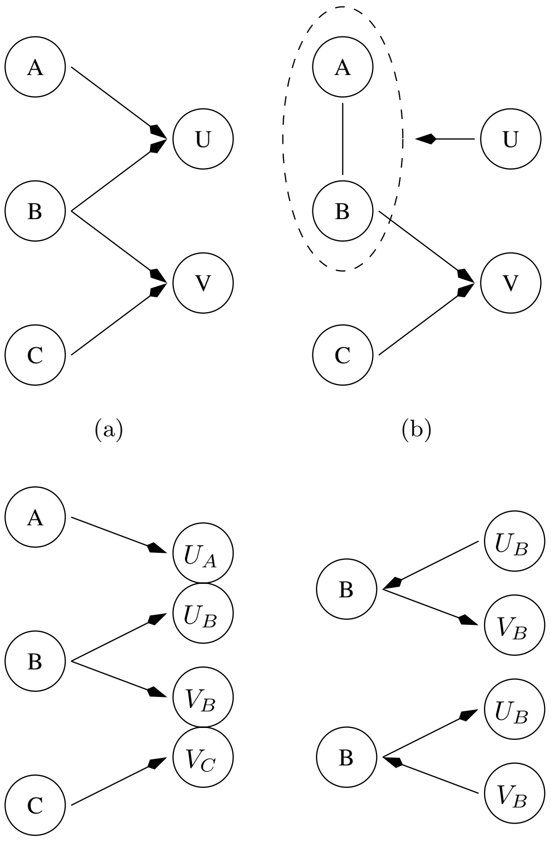
Using the sequence of figure 3 we compared the fusion result when using separately recorded audio sequence from another speaker. The periodogram of the alternate audio sequence is shown in figure 5b. Figure 6a (correct audio) and 6b (alternate audio) compares the resulting projections h_v. In the case that the alternate audio was used we see that coefficients related to the video monitor increase significantly. The estimate of mutual information was 0.68 relative to the maximum possible value for the correct audio sequence. In contrast when compared to the periodogram of 5b , the value drops to 0.08 of maximum. We repeat the same experiment with two speaker video sequence, shown in figure 6c (correct audio) and 6d (alternate audio) and again we see, not surprisingly, the speaker is not localized. The estimate of mutual information for this correct sequence was 0.61 relative to maximum, while it drops to 0.27 when the alternate audio is used.

3.3 Eight-way test

Finally, data was collected from six additional subjects. These data were used to perform an eight-way test. Each video sequence was compared to each audio sequence. No attempt was made to optimally align the mismatched audio sequences. Table 1 summarizes the results. The previous sequences correspond to subjects 1 and 2 in the table. In every case the matching audio/video pairs exhibited the highest mutual information after estimating the projections.

	a1	a2	a3	a4	a5	a6	a7	a8
v1	0.68	0.19	0.12	0.05	0.19	0.11	0.12	0.05
v2	0.20	0.61	0.10	0.11	0.05	0.05	0.18	0.32
v3	0.05	0.27	0.55	0.05	0.05	0.05	0.05	0.05
v4	0.12	0.24	0.32	0.55	0.22	0.05	0.05	0.10
v5	0.17	0.05	0.05	0.05	0.55	0.05	0.20	0.09
v6	0.20	0.05	0.05	0.13	0.14	0.58	0.05	0.07
v7	0.18	0.15	0.07	0.05	0.05	0.05	0.64	0.26
v8	0.13	0.05	0.10	0.05	0.31	0.16	0.12	0.69

Table 1. Summary of results over eight video sequences. The columns indicate which audio sequence was used while the rows indicate which video sequence was used. In all cases the correct audio/video pair have the highest relative MI score.



(a) (b)

(c) (d)

Fig. 2. Graphs illustrating the various statistical models exploited by the algorithm: (a) the independent cause model - U and V are independent of each other conditioned on $\{A, B, C\}$, (b) information about U contained in V is conveyed through *joint* statistics of A and B, (c) the graph implied by the existence of a separating function, and (d) two equivalent Markov chains which can be extracted from the graphs *if* the separating functions can be found.

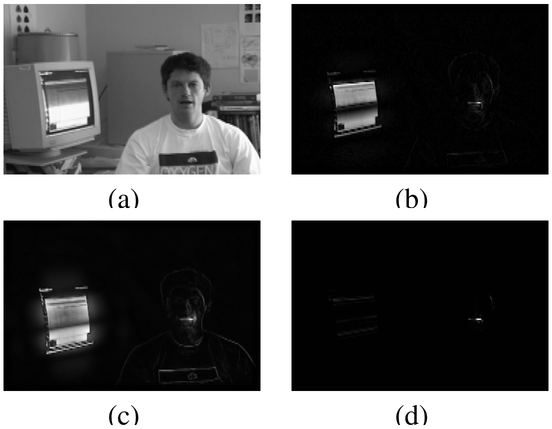


Fig. 3. Video sequence contains one speaker and monitor which is flickering: (a) one image from the sequence, (b) magnitude of the image after pre-whitening, (c) pixel-wise image of standard deviations taken over the entire sequence, (d) image of the learned projection, h_V.

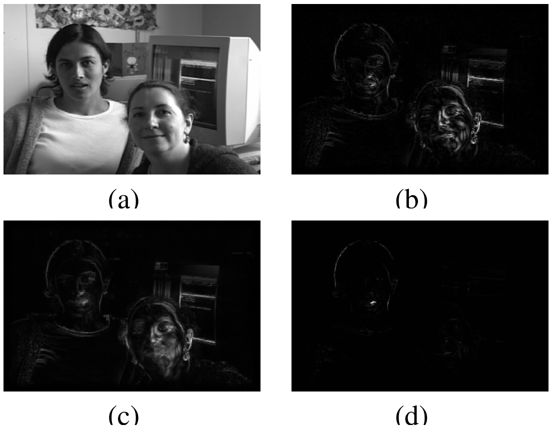


Fig. 4. Video sequence containing one speaker (person on left) and one person who is randomly moving their mouth/head (but not speaking): (a) one image from the sequence, (b) magnitude of the image after pre-whitening, (c) pixel-wise image of standard deviations taken over the entire sequence, (d) image of the learned projection, h_V.

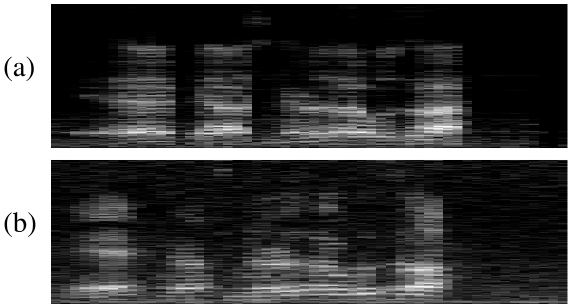


Fig. 5. Gray scale magnitude of audio periodagrams. Frequency increases from bottom to top, while time is from left to right. (a) audio signal for image sequence of figure 3. (b) alternate audio signal recorded from different subject.

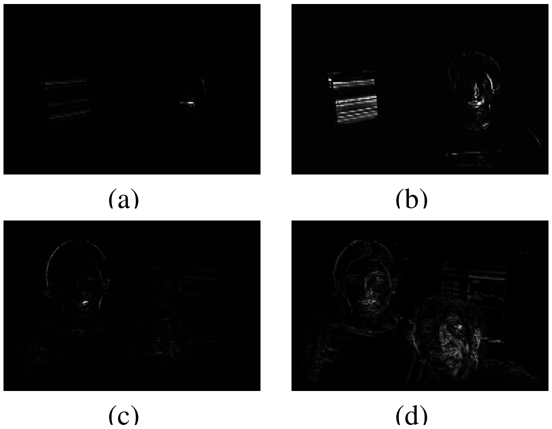


Fig. 6. Comparison of learned video projections when correct (left) and incorrect (right)(audio is compared to image sequences of figure 3 (top) and figure 4 (bottom). When correct audio is used energy is concentrated on subject, when incorrect audio is used it is distributed throughout the image.

4 Discussion and future work

We have presented a method for information theoretic fusion of audio and video data. We have demonstrated over a small set of data, that the method shows promise for detecting audio-video consistency. We are not aware of equivalent results in the literature, although previous multi-modal methods might also work for this application. However, in contrast to previous approaches our method does not make strong assumptions about the underlying joint properties of the modalities being fused (e.g. Gaussian statistics). Consequently, it has the capacity to represent more complex structure which may be present in the data. Furthermore, our method makes no use of training data. While there is an adaptive element to the method, the adaptation occurs in an online fashion over a short sequence (approximately 2-2.5 seconds) of audio-video data. Consequently, the method is applicable when a prior model cannot be trained. As might happen when a multi-modal interface is moved to a new environment. Future work will address the robustness of the method over a larger corpus of data. Another area of interest is to determine the relationship between camera resolution, audio signal-to-noise ratio, and sampling rates for which the method maintains reliability.

5 Acknowledgements

This research was supported by MIT Project Oxygen.

References

1. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., New York, 1991.
2. J.W. Fisher and J.C. Principe. Unsupervised learning for nonlinear synthetic discriminant functions. In D. Casasent and T. Chao, editors, *Proc. SPIE, Optical Pattern Recognition VII*, volume 2752, pages 2–13, 1996.
3. John W. Fisher III, Trevor Darrell, William T. Freeman, and Paul. Viola. Learning joint statistical models for audio-visual fusion and segregation. In *Advances in Neural Information Processing Systems 13*, 2000.
4. John W. Fisher III, Alex T. Ihler, and Paul A. Viola. Learning informative statistics: A nonparametric approach. In S. A. Solla, T. K. Leen, and K-R. Mller, editors, *Advances in Neural Information Processing Systems 12*, 1999.
5. John W. Fisher III and Jose C. Principe. Entropy manipulation of arbitrary nonlinear mappings. In J.C. Principe, editor, *Proc. IEEE Workshop, Neural Networks for Signal Processing VII*, pages 14–23, 1997.

6. John W. Fisher III and Jose C. Principe. A methodology for information theoretic feature extraction. In A. Stuberud, editor, *Proceedings of the IEEE International Joint Conference on Neural Networks*, 1998.
7. John Hershey and Javier Movellan. Using audio-visual synchrony to locate sounds. In S. A. Solla, T. K. Leen, and K-R. Mller, editors, *Advances in Neural Information Processing Systems 12*, pages 813–819, 1999.
8. A. Mahalanobis, B. Kumar, and D. Casasent. Minimum average correlation energy filters. *Applied Optics*, 26(17):3633–3640, 1987.
9. Uwe Meier, Rainer Stiefelhagen, Jie Yang, and Alex Waibel. Towards unrestricted lipreading. In *Second International Conference on Multimodal Interfaces (ICMI99)*, 1999.
10. E. Parzen. On estimation of a probability density function and mode. *Ann. of Math Stats.*, 33:1065–1076, 1962.
11. Malcolm Slaney and Michele Covell. Facesync: A linear operator for measuring synchronization of video facial images and audio tracks. In T. K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, 2000.
12. G. Wolff, K. V. Prasad, D. G. Stork, and M. Hennecke. Lipreading by neural networks: Visual preprocessing, learning and sensory integration. In *Proc. of Neural Information Proc. Sys. NIPS-6*, pages 1027–1034, 1994.

Human Computer Interaction
International Proceedings, pp. 687–691,
2001.

Preserving the Freedom of Paper in a Computer-Based Sketch Tool

Christine J. Alvarado and Randall Davis

MIT Artificial Intelligence Laboratory
{`calvarad, davis`}`@ai.mit.edu`

Abstract. The tradeoff between the ease of sketching a mechanical system on paper and the power of representing it on a computer is too great, as evidenced by the fact that mechanical designers typically begin by sketching their designs on paper, then transfer them to the computer only after the initial design process is complete. To bring early design to the computer, we must develop a tool that allows the user to sketch freely, yet still interprets and understands the user's drawing as a mechanical system. We present a tool that allows users to sketch simple 2-D mechanisms and report on early reactions to it.

1 Introduction

User interfaces to current computer aided design (CAD) and simulation tools offer little assistance at the conceptual stage of design. As a result, designers typically begin by sketching with paper and pencil and transfer their designs to the computer only when nearly complete.

Our goal is to eliminate this two-stage design process by creating a computer-based sketching tool that is both powerful and natural. The tool should allow the user to sketch freely, without modifying her drawing style, yet still be able to interpret the user's drawing and allow the user to interact with it, in a manner illustrated below.

Free sketching is not the only route to ease of interaction with design tools. As one example, sketch interpretation can be simplified by constraining the user's drawing style, for example by requiring that the user draw each object with a single stroke [9]. [8] explores another approach, creating shapes by carving larger shapes with a carving tool rather than sketching them directly.

We chose instead to allow users to sketch as they would on paper, attempting to alter their style as little as possible. There is an interesting, open question of whether free sketching is indeed the best overall interaction metaphor, but detailed comparisons of sketching vs. menus (or other approaches) cannot be addressed until an adequately powerful sketch system has been built. This work is a first step in building such a system.

Our tool, called ASSIST (A Shrewd Sketch Interpretation and Simulation Tool), enables sketching simple two-dimensional mechanical systems in a natural fashion, i.e., without explicitly informing the system what is being drawn. ASSIST interprets the sketch as the user draws and can simulate the design at any time during the design process. This paper describes the tool, discusses a user study that evaluated its utility in its current form, and suggests directions for future work and improvements.

We begin with a description of a typical interaction with ASSIST, then introduce our model for sketch interpretation and interaction, including a brief discussion of our recognition algorithm. This is followed by a discussion of the user test we performed, the feedback we gathered, and the implications for future design. We conclude with a discussion of related and future work.

2 Interacting with assist

Users interact with the system through a digitizing LCD tablet that allows them to draw directly onto the computer screen and see their strokes appear under their pen as they draw them.

We describe a session in which a user draws the car on a hill seen in Figure 1. The user begins by drawing one side of the polygon that constitutes the car body on an otherwise blank canvas. As she is drawing, she sees her stroke appear under her pen in grey. As she lifts her pen, the system replaces her stroke with a black line to indicate its recognition. At this point the system believes that the user's stroke could possibly be a rod[1] or a line that will become part of some other part (in this case a mechanical body). Next she draws the rest of the body with one stroke. Again, she sees her strokes in gray as she draws; when she lifts her pen the system replaces all of her strokes with a blue polygon that closely matches the strokes she has drawn (Figure 1, center). Note that the system has "cleaned up" her raw strokes to indicate its recog-

---

[1] A rod is a thin rigid body.

nition. We discuss the advantages and disadvantages of this behavior in Section 4.

The user continues to draw the other pieces of the drawing. By mistake, she draws one of the wheels too large. To delete the wheel, she circles it; the wheel turns red to indicate it has been selected. She then draws a line through it to delete it.

After she finishes the drawing, she draws the hill for the car to sit on, and anchors it to the background by drawing an "X"[2]. At this point the car is floating above the hill. To move it, she circles it to select it, and then drags it down to rest on the hill by putting her pen on one of the selected pieces and dragging it downward until it rests on the hill. As she drags her pen, the whole car moves with it.

If at any point in the drawing process the system misinterprets one of the user's strokes, she can tap the "Try Again" button at the bottom of the screen, and ASSIST will present a list of alternative interpretations for her stroke.

Finally, when she finishes her drawing (or at any time during the design process), she can tap the "Run" button to see a simulation of the system: another window appears (Figure 1 right), showing the car running down the hill. The original drawing remains unchanged, allowing for a convenient sketch-modify-simulate loop.

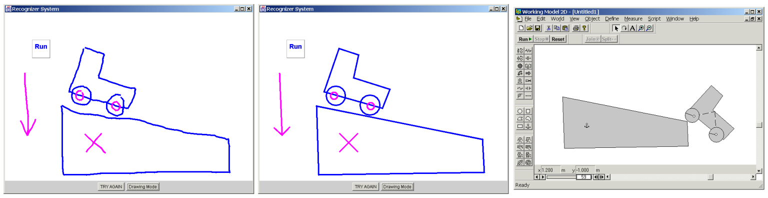


Fig. 1. A car on a hill: As drawn by the user (left), as interpreted and displayed by ASSIST (center) and as simulated in Working Model (right).

3 Approach

3.1 Recognition

Although our recognition algorithm is not the focus of this paper, understanding it will help explain some of our interface design decisions. For a more complete description of the algorithm, see [1].

---

[2] Anything not anchored can fall.

The first step to recognizing a user's sketch is to identify the patterns in the sketch that represent mechanical parts. To identify these patterns without forcing the user to significantly change her drawing style, ASSIST relies on the fact that mechanical engineering has a fairly concrete visual vocabulary for representing components [2].

Simply recognizing these patterns is not enough, however, because there is a great deal of ambiguity in any mechanical sketch. For example, consider wheels of the car in Figure 1. Are the small circles in the middle of the large circles pin joints or circular bodies themselves? The recognition process must also choose the correct representation for a set of strokes, given multiple patterns to choose from.

After each stroke, ASSIST uses a three-stage procedure to choose the most likely interpretation for that stroke. First, it matches the user's strokes (including the most recently drawn) to a series of templates, producing all possible interpretations of the strokes. Next, the system ranks the interpretations using a series of heuristics (described in [1]) about drawing style and mechanical engineering. As one example, consider one of the the small circles that represent the pin joints in Figure 1. The system was able to recognize it as a pin joint instead of a circular body because in two-dimensional systems bodies do not typically overlap without some means of interconnection (i.e. the pin joint). Finally, the system chooses the best consistent overall set of interpretations from among those produced in step 1, and displays it to the user.

3.2 Level of aggressiveness

One design issue we faced was how "aggressive" the system should be in interpreting the sketch. Interpreting after every stroke keeps the user informed of the system's current understanding, allowing her to intervene promptly to correct any misinterpretations[3].

We could alternatively have chosen to delay interpretation, interpreting only after the user had finished drawing, or at least paused substantially. This approach has the advantage that the user would not have to deal with the system's interpretations after every stroke. But errors are more difficult to correct: The user may have to go back and fix interpretation errors after finishing part of the drawing, as an early mistake can lead subsequent interpretations down the wrong path.

---

[3] Lack of intervention is, conversely, taken as tacit approval: the system becomes more sure of an interpretation the longer it remains unchallenged by the user.

We decided to build the system with a constant interaction between the system and the user, to keep the user's expectations from becoming too far removed from the system's interpretations. This interaction also offers guidance that keeps the system from getting lost in what would otherwise be a very large space of possible interpretations, as the drawing becomes more complex[4].

4 Evaluation and results

To judge how natural the system is we ran a test in which subjects were asked to sketch three simple mechanical systems, both on paper and using ASSIST via an active matrix LCD tablet. We observed their behavior and asked them to explain their experience with the system, describing in what ways it felt natural and in what ways it felt awkward. Since this is an early-stage design, we were looking for qualitative feedback not quantitative results. We use the feedback to guide the system into its next implementation.

4.1 Method

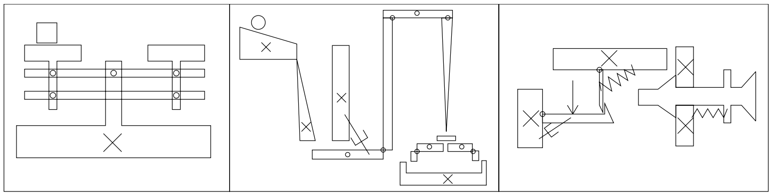


Fig. 2. Three test examples: a scale (left), a Rube-Goldberg machine (center) and a circuit breaker (right).

Eleven subjects from the MIT AI Lab took part in the study. All of the subjects had a background in computer science; two were mechanical engineers.

Sessions with the subjects lasted between 30 minutes and an hour and were videotaped.

---

[4] Results from our study (Section 4), however, indicate that it might be more natural for the system to display its interpretation less often, even at the cost of a few more mistakes.

We first showed them the diagrams in Figure 2 and told them they would be asked to drawn them twice, once on paper and once using our system. They were told not to worry about replicating the details exactly. After they drew the figures on paper, we gave them a 5 minute introduction to ASSIST that included drawing simple parts, such as a pair of bodies connected by a spring. We also told them how to select, move and delete objects. We then asked them to use ASSIST to draw the same systems they had drawn on paper.

After they were done sketching we asked them a series of questions (Table 1). Although we were observing their behavior as they sketched, we also wanted them to articulate what they felt was particularly intrusive or particularly helpful in working in ASSIST. The first few questions measure the perceived number of mistakes the system made and how tolerant users were of these mistakes; the last two are open ended.

1. How often did you feel the system got the correct interpretation for your strokes?
2. When the system misinterpreted your stroke, how reasonable was its misinterpretation?
3. How clear was the system's interpretation for your strokes?
4. How easy was it to modify the drawing?
5. Compare using this system to drawing on paper.
6. Compare using this system to using a menu-based interface.

Table 1. Questions we asked the subjects.

4.2 Results

Our first observation was that the learning curve for our system is low. All subjects were able to successfully draw all of the systems in Figure 2. While they had to modify slightly the way they drew certain objects, such as polygons and springs, for the most part they were either able to use their natural drawing style or quickly learn the small changes required to work successfully with ASSIST[5].

Because of the slight modifications to their drawing styles, it took subjects slightly longer to draw system with ASSIST than it did on paper. The three examples took the subjects about 10 minutes total to sketch on paper (approximately 2 minutes for the scale, 4 minutes

---

[5] Since this study, another member of our group has implemented more robust low-level recognizers to be more accurate with less-constrained drawing styles. (See [10].)

for the Rube-Goldberg machine, and 4 minutes for the circuit breaker). The scale took subjects about 5 minutes to draw using ASSIST, while subjects spent about 10 minutes on the Rube-Goldberg machine and about 10 minutes on the circuit breaker.

In response to question 1, almost all subjects reported the that the system got the correct interpretation of their strokes between 70 and 80 percent of the time. One subject reported that the system got the correct interpretation for his strokes only 50 percent of the time, while another said he felt the system interpreted his strokes correctly about 95 percent of the time. Most recognition errors were polygon recognition errors.

When the system did make a mistake, subjects overwhelmingly preferred to delete the mistaken interpretation and redraw it, rather than using the "Try Again" button. A few subjects tried using "Try Again", but did not find the correct interpretation of their piece and ended up deleting the piece they had just drawn anyway. The bias against the "Try Again" button is likely due to the fact that when the system misinterprets a user's stroke, it is often a result of never identifying the correct interpretation for that stroke, in which case the correct interpretation does not appear in the "Try Again" list. It may also be the case that our users are too used to paper, where re-drawing is the common reaction to a misunderstanding.

All subjects reported that it was clear when the system had made a mistake that they needed to correct, but also reported that the feedback was at times distracting. The primary source of distraction was the way the system replaced the user's strokes with the icon indicating its interpretation. Even though the icon is fitted to match the user's stroke as closely as possible, the match is never exact, so the stroke appears to "jump" a little when it is replaced. This behavior bothered some people because they felt that they no longer had total control of the strokes they were putting on the page and it was therefore difficult to have any sort of precision in their drawing. In our current implementation we have removed the replacement of the user's strokes and instead use a color change to indicate interpretation. Further investigation is necessary to determine whether the color change gives adequate feedback.

People were quite pleased with the power that the system brought to the drawing process in the realm of mechanical engineering. Subjects found the ability to simulate the drawings to be quite appealing. It is especially interesting to note that subjects were intrigued when the behavior of their system did *not* match the intended behavior. For example, unless drawn with great attention to symmetry, the scale will not balance when empty (by default all bodies have the same mass so the scale will tilt toward the side with the longer lever arm). Subjects often tweaked their drawings until they got the desired behavior.

People also liked the idea that the system could be a more complete editing tool. For example, although paper does not give users the ability to cut and paste or rotate their images, most people said that having these features in the program would not make the interface feel any less natural, but would simply add power they have come to expect from a computer program. Also, people liked the idea that ASSIST was able to clean up their drawings, even though they would have preferred for this cleanup to take place after they were done drawing. Several subjects expressed interest in a more controlled way to clean up their drawings. They wanted their strokes to be left rough at first, with the option to go back and touch up the drawing later.

5 Related work

Other sketch-based design tools include the Electronic Cocktail Napkin [3, 4] and SILK (Sketching Interfaces Like Krazy) [5]. Our work explores many similar issues as theirs, but in the domain of mechanical engineering where the task of recognition involves more ambiguity resolution.

Our work deals with many of the issues presented in [6]. Their work acknowledges the idea that in any naturally based interface will have inherent ambiguity in the input. She presents a method for representing these ambiguities and then applying a series of “mediators” to the possible recognitions. Our approach is similar; the second step of our recognition algorithm is similar to the “mediation” stage in her architecture.

6 Future work and conclusions

We are exploring our free sketch interface further by incorporating the feedback received from users into the current version of our interface. We have been working to improve low-level recognition algorithms because most of the interpretation mistakes made by the system were low-level mistakes. We have changed our feedback on the interpretation of the design to simply change the color of the strokes, rather than replace the strokes themselves.

We would also like to produce more intuitive simulations for the mechanical systems sketched by users. We noted that users tweaked their drawings until the simulator gave them the desired behavior. In many cases they were confused as the why the simulator did not produce the correct behavior, not understanding that a literal interpretation of a hand-drawn sketch rarely produces the desired qualitative simulation of the system. Other work in our group has developed methods that allow users to describe the intended behavior of sketched devices through spoken explanations and sketched gestures [7]. Our goal is to have ASSIST use the behavioral descriptions to adjust parameters such that the simulation gives the desired qualitative behavior.

We have suggested that engineers currently do not use computers in early design because they do not allow for quick and natural sketching of ideas. To be useful in early design, computers must allow the designer to sketch as on paper, and provide benefits, such as the ability to simulate the system, that are not available with paper. We believe the work described here takes several small but important steps in that direction.

References

1. Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. In *Proceeding of IJCAI-2001*, August 2001.
2. Ivan I. Artobolevsky. *Mechanisms in Modern Engineering Design*, volume 1. MIR Publishers, Moscow, 1976.
3. Ellen Yi-Luen Do and Mark D. Gross. Drawing as a means to design reasoning. *AI and Design*, 1996.
4. Mark D. Gross. The electronic cocktail napkin - a computational environment for working with design diagrams. *Design Studies*, 17:53–69, 1996.
5. James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3):56–64, March 2001.
6. Jennifer Mankoff, Scott E Hudson, and Grefory D. Abowd. Providing intergrated toolkit-level support for ambiguity in recogntiion-based interfaces. In *Proceedings of the CHI 2000 conference on Human factors in computing systems*, pages 368–375, 2000.
7. Michael Oltmans. Understanding natually conveyed explanations of device behavior. Master's thesis, Massachusetts Institute of Technology, 2000.
8. Roope Raisamo. An alternative way of drawing. In *Proceedings of CHI 1999*, pages 175–182, 1999.
9. Dean Rubine. Specifying gestures by example. *Computer Graphics*, pages 329–337, July 1991.
10. Metin Sezgin. Early processing in sketch understanding. Unpublished Master's Thesis, Massachusetts Institute of Technology.

Proceedings of IJCAI-01,
pp. 1365–1371, August, 2001.

Resolving Ambiguities to Create a Natural Computer-Based Sketching Environment

Christine J. Alvarado and Randall Davis

MIT Artificial Intelligence Laboratory
{`calvarad, davis`}`@ai.mit.edu`

Abstract. Current computer-based design tools for mechanical engineers are not tailored to the early stages of design. Most designs start as pencil and paper sketches, and are entered into CAD systems only when nearly complete. Our goal is to create a kind of "magic paper" capable of bridging the gap between these two stages. We want to create a computer-based sketching environment that feels as natural as sketching on paper, but unlike paper, understands a mechanical engineer's sketch as it is drawn. One important step toward realizing this goal is resolving ambiguities in the sketch— determining, for example, whether a circle is intended to indicate a wheel or a pin joint—and doing this as the user draws, so that it doesn't interfere with the design process. We present a method and an implemented program that does this for freehand sketches of simple 2-D mechanical devices.

1 Sketching conceptual designs

Engineers typically make several drawings in the course of a design, ranging from informal sketches to the formal manufacturing drawings created with drafting tools. Drawing is far more than an artifact of the design process; it has been shown to be essential at all stages of the design process [16]. Yet almost all early drawings are still done using pencil and paper. Only after a design is relatively stable do engineers take the time to use computer aided design or drafting tools, typically because existing tools are too difficult to use for the meager payoff they provide at this early stage.

Our aim is to allow designers to sketch just as they would on paper, e.g., without specifying in advance what component they are drawing, yet have the system understand what has been sketched. We want to have the input be as unconstrained as possible, in order to make interaction easy and natural; our route to accomplishing this is to build a sufficiently powerful sketch recognizer.

It is not yet obvious that a freehand sketching interface will be more effective in real use than a carefully designed menu-based system. In order to do the comparison experiments, however, we must first build powerful sketch-based systems. It is the construction of such a system that is the focus of this paper.

The value of sketching as an interface and the utility of intelligent sketch understanding has gained increasing attention in recent years (e.g., [6]). Some early research was concerned with single stroke classification ([11]), while more recent work ([4, 7]) puts groups of strokes together to form larger components. A number of efforts (e.g., [3], [8]) have acknowledged the necessity of representing ambiguities that arise in interpreting strokes, but have not substantially addressed how to resolve those ambiguities.

Given the frequency of ambiguities in a sketch, a tool that constantly interrupts the designer to ask for a choice between multiple alternatives would be cumbersome. Our work is thus focused, in part, on creating a framework in which to both represent and use contextual (top-down) knowledge to resolve the ambiguities. We built a program called ASSIST (A Shrewd Sketch Interpretation and Simulation Tool) that interprets and understands a user's sketch as it is being drawn, providing a natural-feeling environment for mechanical engineering sketches.

The program has a number of interesting capabilities.

- The basic input to the program is a sketch, i.e., a sequence of strokes drawn "while the system watches," not a finished drawing to be interpreted only after it is complete.
- Sketch interpretation happens in real time, as the sketch is being created.
- The program allows the user to draw mechanical components just as on paper, i.e., as informal sketches, without having to pre-select icons or explicitly identify the components.
- The program uses a general architecture for both representing ambiguities and adding contextual knowledge to resolve the ambiguities.
- The program employs a variety of knowledge sources to resolve ambiguity, including knowledge of drawing style and of mechanical engineering design.
- The program understands the sketch, in the sense that it recognizes patterns of strokes as depicting particular components, and

illustrates its understanding by running a simulation of the device, giving designers a way to simulate their designs as they sketch them.

We describe the system and report on a pilot user study evaluating the naturalness of the program's interface and the effectiveness of its interpretations.

2 Designing with ASSIST

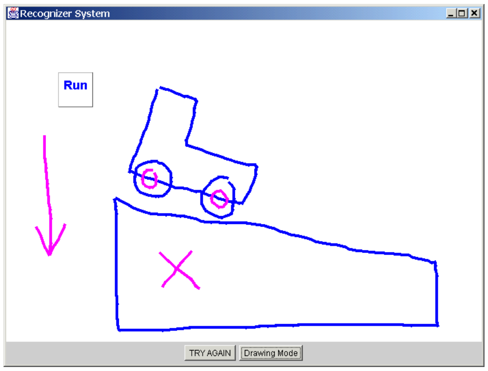


Fig. 1. A car on a hill, as drawn by the user in ASSIST.

Figure 1 shows a session in which the user has drawn a simple car on a hill. The user might begin by drawing the body of the car, a free-form closed polygon. As the user completes the polygon, the system displays its interpretation by replacing the hand-drawn lines (shown in Figure 1) with straight blue lines. Next the user might add the wheels of the car, which also turn blue as they are recognized as circular bodies. The user can then "attach" the wheels with pin joints that connect wheels to the car body and allow them to rotate. The user might then draw a surface for the car to roll down, and anchor it to the background (the "x" indicates anchoring; anything not anchored can fall). Finally, the user can add gravity by drawing a downward pointing arrow not attached to any object. The user's drawing as re-displayed by ASSIST is shown in Figure 2.

The system recognizes the various components in the drawing by their form and context; when the "Run" button is tapped, it transfers

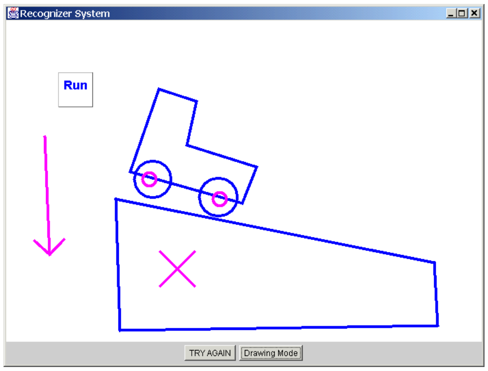


Fig. 2. The sketch as displayed by ASSIST.

the design to a two-dimensional mechanical simulator which shows what will happen (Figure 3).[1]

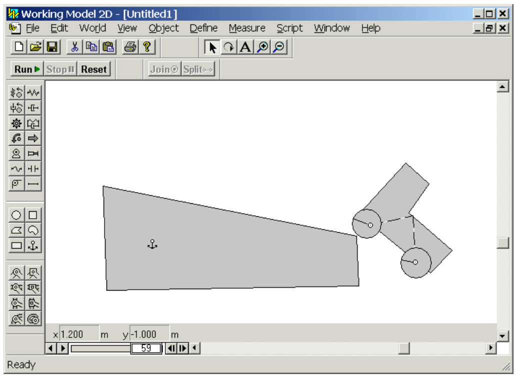


Fig. 3. The sketch simulated, showing the consequences.

Note that the user drew the device without using icons, menu commands, or other means of pre-specifying the components being drawn. Note, too, that there are ambiguities in the sketch, e.g., both the wheels of the car and pin joints are drawn using circles, yet the system was

---

[1] We use Working Model 2D from Knowledge Revolution, a commercial mechanical simulator; any simulator with similar capabilities would do as well.

able to select the correct interpretation despite these ambiguities, by using the knowledge and techniques discussed below. The automatic disambiguation allowed the user to sketch without interruption.

Figure 4 shows a session in which the user has drawn a more interesting device, a circuit breaker, and run a simulation of its behavior.

Note that ASSIST deals only with recognizing the mechanical components in the drawing and is, purposely, literal-minded in doing so. Components are assembled just as the user drew them, and component parameters (e.g. spring constants, magnitudes of forces, etc) are set to default values. The car in Figures 1–3, for example, wobbles as it runs down the hill because the axles were not drawn in the center of the wheels. The combination of literal-minded interpretation and default parameter values can produce device behavior other than what the user had in mind. Other work in our group has explored the interesting and difficult problem of communicating and understanding the *intended* behavior of a device once it has been drawn using ASSIST [10].

3 Embedding intelligent assistance

We created a model for sketch understanding and ambiguity resolution inspired by the behavior of an informed human observer, one that recognizes the sketch by relying on both low-level (i.e., purely geometric) routines and domain specific knowledge.

One interesting behavior of an informed observer is that interpretation begins as soon as the designer begins sketching. While not a required strategy—people can obviously interpret a finished sketch—there are advantages in ease of use and in speed from having the program do its interpretation in parallel with drawing. Ease of use arises because the program can provide an indication of its interpretation of parts of the sketch as soon as they are drawn, making it easier for the user to correct a misinterpretation. Interpretation is faster because incremental interpretation effects a divide and conquer strategy: parts of the drawing interpreted correctly can provide useful context when interpreting parts drawn subsequently.[2]

A second interesting behavior of an informed observer is the ability to accumulate multiple interpretations and defer commitment. Consider for example the objects in Figure 5. Are the strokes in 5a going to become part of a ball and socket mechanism (5b), or are they the

---

[2] The program also seems faster because it is working while the user is drawing, reducing the user's wait.

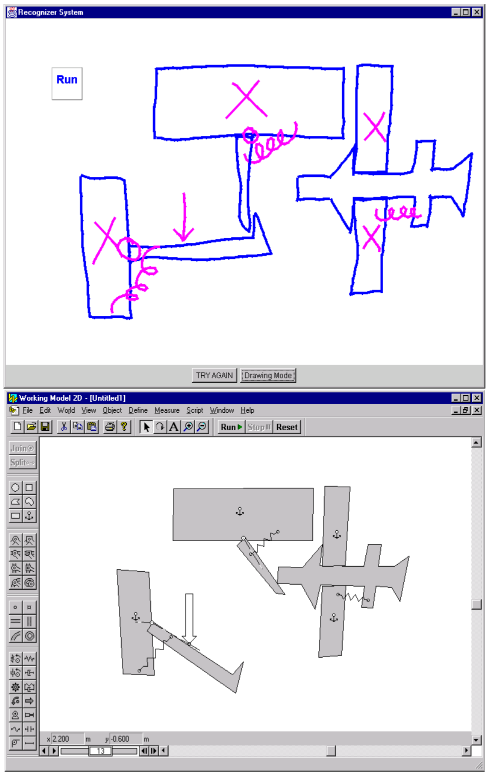


Fig. 4. A sketch of a circuit breaker (left) and its simulation (right).

beginning of a gear (5c)? Committing too soon to one interpretation precludes the other. Hence interpretation must be capable of revision in the face of new information.

There is clearly a need to balance out the desire for interpretation occurring in parallel with drawing, and the need to avoid premature commitment. We discuss below how our system accomplishes this.

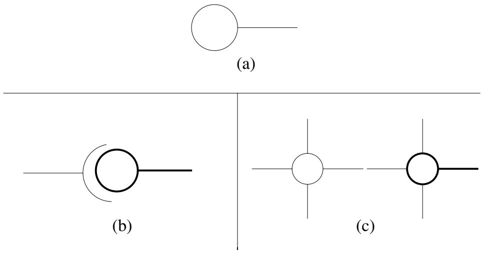


Fig. 5. An example of ambiguity: The bold strokes in (b) and (c) are identical to the strokes in (a).

Third, while commitment should be deferred, it must of course be made eventually, and determining when to make that commitment is not easy. Timing information can assist. Consider the case of circles: Because circles are low-level structures, it is likely that they will be used in higher-level structures, as for example when a circle turns out to be part of a pulley system. One way of dealing with this is to use timing data: the system gets to "watch" the sketch being drawn and knows when each stroke was made. If, some time after the circle has been drawn, it has still not been used in any other structure, the observer can plausibly guess that it will not be incorporated into another piece and should be interpreted as an independent circular body.[3]

Finally, parts may remain ambiguous even when a piece of the drawing is finished. To resolve these residual ambiguities, the observer uses his knowledge of mechanical engineering components and how they combine. Consider, for example, the small circles inside the larger circles in Figure 2; ASSIST determines that these are more likely to be pivot joints than additional circular bodies, both because small circles

---

[3] A *body* is any hunk of material not otherwise interpreted as a more specialized component (like a spring, pin joint, etc.). The car body is a polygonal body; its wheels are circular bodies.

typically indicate pin joints and because bodies do not typically overlap without some means of interconnection (i.e., the pin joint).

Our system incorporates each of these observations: it begins interpreting the sketch as soon as the user starts drawing; it accumulates multiple interpretations, deferring commitment until sufficient evidence (e.g., stroke timing) accumulates to suggest a component has been finished, and it resolves ambiguities by relying on knowledge from the domain about how components combine.

4 ASSIST's interpretation and disambiguation process

ASSIST's overall control structure is a hierarchical template-matching process, implemented in a way that produces continual, incremental interpretation and re-evaluation as each new stroke is added to the sketch. Each new stroke triggers a three stage process of recognition, reasoning and resolution. Recognition generates all possible interpretations of the sketch in its current state, reasoning scores each interpretation, and resolution selects the current best consistent interpretation. After each pass through the three stages the system displays its current best interpretation by redrawing the sketch.

4.1 Recognition

In the recognition stage, ASSIST uses a body of recognizers, small routines that parse the sketch, accumulating all possible interpretations as the user draws each stroke. A recognizer takes as input raw strokes and previously recognized objects, and if the input fits its template, produces a new object. For example, the circle recognizer reports a circle if all the points on a stroke lie at roughly the same distance from the average X and Y coordinate of the stroke.[4] The circle is then available to other recognizers, e.g., the pulley recognizer.

4.2 Reasoning

In the second stage the system scores each interpretation using a variety of different sources of knowledge that embody heuristics about how people draw and how mechanical parts combine.

---

[4] In other work we describe recognizers that use more sophisticated early processing of basic geometry [14].

Temporal evidence People tend to draw all of one object before moving to a new one. Our system considers interpretations that were drawn with consecutive strokes to be more likely than those drawn with non-consecutive strokes.

Additional evidence comes from “longevity:” the longer a figure stays unchanged, the stronger its interpretation becomes, because as time passes it becomes more likely that the figure is not going to be turned into anything else by additional strokes.

Simpler is netter We apply Occam’s razor and prefer to fit the fewest parts possible to a given set of strokes. For example, any polygonal body (e.g., the car body in Figure 2) could have been interpreted as a set of (connected) individual rods, but the system prefers the interpretation “body” because it fits many strokes into a single interpretation.

More specific is better Our system favors the most specific interpretation. Circles, for example, (currently) have three interpretations: circular bodies, pin joints, and the “select” editing gesture. The selection gesture is the most specific interpretation, in the sense that every circle can be a circular body or pin joint, but not every circle can be a selection gesture (e.g., if it does not encircle any objects). Hence when a circle contains objects inside of it, the system prefers to interpret it as a selection gesture.

Domain knowledge ASSIST uses basic knowledge about how mechanical components combine. For example, a small circle drawn on top of a body is more likely to be a pin joint than a circular body.

User feedback User feedback also supplies guidance. The “Try Again” button (see the bottom of Figure 1) permits the user to indicate that something was recognized incorrectly, at which point the system discards that interpretation and offers the user an ordered list of alternative interpretations. Conversely the system can be relatively sure an interpretation is correct if the user implicitly accepts it by continuing to draw.

Combining evidence The heuristics described above all independently provide evidence concerning which interpretation is likely to be correct. Our method of combining these independent sources involves distinguishing between two categories of evidence: categorical and situational.

Categorical evidence ranks interpretations relative to one another based on the first four knowledge sources described above. Each source is implemented in the system as a set of rules that takes two interpretations as input, and outputs an ordering between them. In processing Figure 1, for example, the interpretation "body" is ranked higher than the interpretation "connected rods," based on the "Simpler is Better" heuristic.

Situational evidence comes from implicit and explicit feedback from the user. Explicit feedback is provided by use of the "Try Again" button; implicit feedback arises when the user keeps drawing after the system displays an interpretation, suggesting that the user is satisfied that the system has understood what has been drawn so far.

The system gives each interpretation two numeric scores, one from each category of evidence. The categorical score is an integer from 0 to 10; the situational score is an integer from -11 to 11. These values are chosen so that the situational dominates the categorical, because we want user feedback to dominate general ranking rules. An interpretation's total score is simply the sum of its two scores.

To convert categorical evidence to a numerical score (so it can be combined it with the situational score), we generate a total ordering of all the interpretations consistent with the partial orders imposed by the categorical evidence. We do a topological sort of the graph of partial orders produced by the evidence and distribute scores evenly, from 0 to 10, over all the interpretations in the sorted graph.[5]

Situational scores start out at 0 and are strengthened or weakened by evidence that can raise of lower the current value by 1 or by 11. Situational evidence thus either modifies an interpretation's value by a small amount (1 unit) or declares it to be certainly correct or certainly incorrect. The system declares an interpretation to be certainly correct or certainly incorrect when the user explicitly accepts or rejects the interpretation using the "Try Again" dialog box. The system strengthens

---

[5] The system first removes cycles in the graph by collapsing strongly connected components. Conceptually, this step indicates that the system will give an equal score to all interpretations that have inconsistent ordering given the evidence (i.e., one rule says A is more likely than B, while another says B is more likely than A). In addition, if there are more than 11 interpretations, the top ten are assigned scores of 10 through 1; the remaining interpretations all receive a score of 0.

an interpretation by a small amount each time strokes added by the user are consistent with that interpretation.[6]

We developed this approach to accumulating and combining evidence, and implemented our knowledge sources as a rule based system, in order to provide a degree of modularity to the system. Our overall approach to the problem is to take into account as many sources of knowledge as prove useful in interpreting the sketch. We knew that it would be impossible to identify and implement them all at the outset, hence our design put a high premium on the ability to add and remove sources of evidence easily.

4.3 Resolution

The third stage in the interpretation process involves deciding which interpretation is currently the most likely. Our system uses a greedy algorithm, choosing the interpretation with the highest total score, eliminating all interpretations inconsistent with that choice, and repeating these two steps until no more interpretations remain to be selected.

The process is illustrated by the interpretation graph in Figure 6, which shows in graphical form all of the possible interpretations of four strokes (the top row of ovals): 4 separate lines, 4 rods, a quadrilateral, rectangle, or square. The rod on the left has the highest score, so it is chosen as a correct interpretation for stroke A. Choosing that interpretation eliminates the interpretations of quadrilateral, rectangle or square, because stroke A is needed in any of these interpretations. In this context the other strokes are interpreted as rods because that interpretation has the highest score of any remaining interpretation.

Recall that our interpretation process is continuous: all three stages of processing occur after every new stroke is added to the sketch, and the current best interpretation as selected by the greedy algorithm is presented to the user. The process tends to settle down reasonably quickly, in part because, as noted, we reward longevity. Hence once an interpretation has been presented to the user and unchanged for some period of time, it becomes increasingly unlikely to change.

5 Evaluation and results

Our initial evaluation of ASSIST has focused on its naturalness and effectiveness. We asked subjects to sketch both on paper and using

---

[6] The system does not yet weaken an interpretation by a small amount; we have included this possibility for symmetry and possible future use.

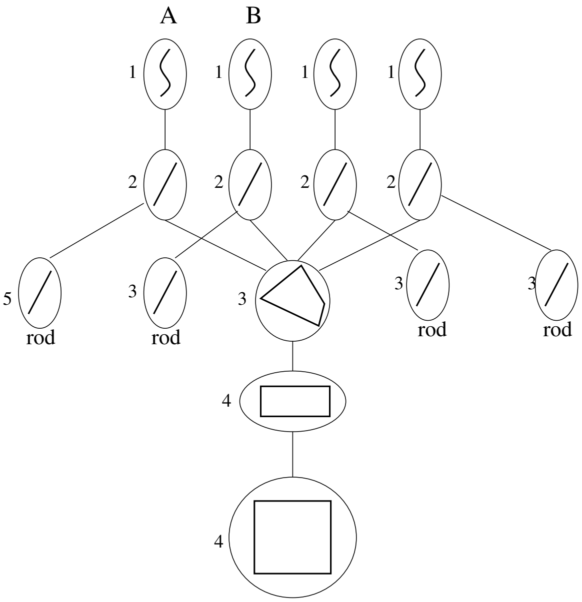


Fig. 6. A recognition graph for four strokes; scores are shown at the left of each interpretation.

ASSIST. We observed their behavior and asked them to describe how ASSIST felt natural and what was awkward about using it.

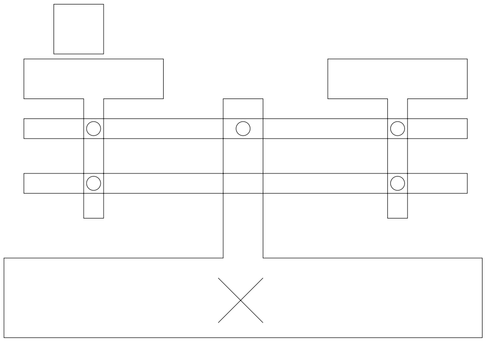


Fig. 7. A scale.

We tested the system on eleven people from our the laboratory, two of whom had mechanical engineering design experience. All were asked first to draw a number of devices on paper (Figures 7, 8, 9), to give them a point of comparison and to allow use to observe differences in using the two media.

They were then asked to draw the same systems using ASSIST (they drew with a Wacom PL-400 tablet, an active matrix LCD display that allows users to sketch and see their strokes appear directly under the stylus). We asked them how often they felt the system got the correct interpretation and how reasonable the misinterpretations were, and asked them to compare using our system to drawing on paper and to using a menu-based interface.

The system was successful at interpreting the drawings despite substantial degrees of ambiguity, largely eliminating the need for the user to specify what he was drawing. As a consequence, a user's drawing style appeared to be only mildly more constrained than when drawing on paper.

People reported that the system usually got the correct interpretation of their sketch. Where the system did err, examination of its performance indicated that in many cases the correct interpretation had never been generated at the recognition step, suggesting that our reasoning heuristics are sound, but we must improve the low-level recognizers. This work is currently under way.

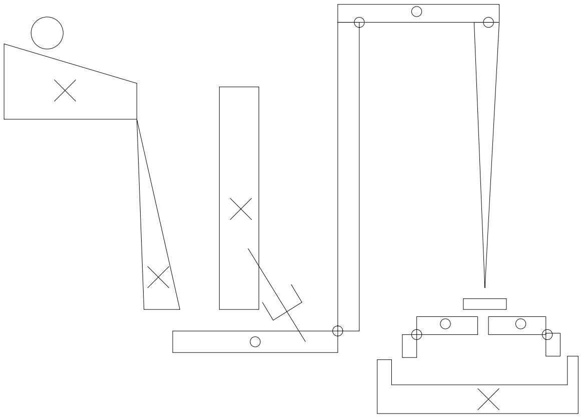


Fig. 8. A Rube-Goldberg machine. The ball rolling down the incline sets in motion a sequence of events that eventually pushes the block at the right into the receptacle at bottom right. The device is an adaptation of the one found in [Narayanan, 1995].

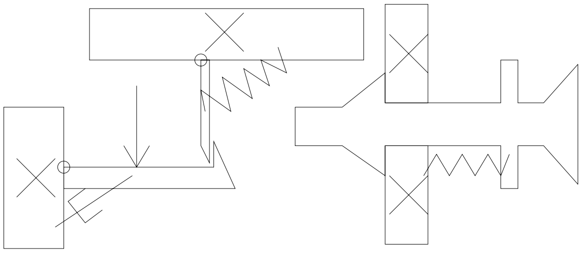


Fig. 9. A circuit breaker.

Users tended to draw more slowly and more precisely with ASSIST than they did on paper. The most common complaint was that it was difficult to do an accurate drawing because the system changed the input strokes slightly when it re-drew them (to indicate its interpretations). Users felt that the feedback given by ASSIST was effective but at times intrusive. Our next generation of the system leaves the path of the strokes unchanged, changing only their color to indicate the interpretation.

For a more complete discussion responses to the system from a user interface perspective, see [1].

6 Related work

The Electronic Cocktail Napkin (ECN) project [2, 5] attacks a similar problem of sketch understanding and has a method for representing ambiguity. Our system takes a more aggressive approach to ambiguity resolution and as a result can interpret more complicated interactions between parts. In order for ECN to to resolve ambiguity, the user must either inform the system explicitly of the correct interpretation, or the system must find a specific higher-level pattern that would provide the context to disambiguate the interpretation of the stroke. Our system, in contrast, takes into account both drawing patterns and knowledge of drawing style.

[8] presents a general framework for representing ambiguity in recognition-based interfaces. This work is similar in using a tree-like structure for representing ambiguity, but touches only briefly on ambiguity resolution. Our work pushes these ideas one step further within the domain of mechanical engineering by providing a framework and set of heuristics for ambiguity resolution.

SILK [7] allows a user to sketch out rough graphical user interface designs, then transform them into more polished versions. SILK addresses the notion of ambiguity, but limits its handling of it to single parts, e.g., is this group of strokes a radio button or a check box? This does not in general affect the interpretation of the other strokes in the sketch. In contrast, our system can resolve ambiguities that affect the interpretation of many pieces of the sketch.

A theoretical motivation to our work was provided by work in [13], which outlines several goals in interpreting ambiguous sketches. Our work implements many of the multiple representation and disambiguation techniques suggested in their work.

We have also been motivated by work in mechanical system behavior analysis, especially in the field of qualitative behavior extraction and representation [12, 15]. The work by Stahovich aims to extract the important design constraints from the designer's rough sketch and is less focused on the interface or sketch recognition process. It was nevertheless the inspiration for our work in this area.

7 Future work

The work presented in this paper is a first step toward creating a natural interface. It can usefully be expanded in several areas.

First, our current formulation of recognition and evidential reasoning is of course quite informal. This is a consequence of our focus at this stage on the knowledge level, i.e., trying to determine what the program should know and use to evaluate interpretations. Once the content has become more stable and better understood, a more formal process of evaluation and control (e.g., Bayes' nets) may prove useful both for speed and scaling.

Second, in our efforts to combine the best properties of paper and the digital medium we have yet to find many of the appropriate tradeoff points. How aggressive should the system be in its interpretations? Forcing the user to correct the system immediately when it makes a mistake greatly aids recognition, but may distract the designer by forcing her to focus on the system's recognition process rather than on the design. In addition, some ambiguities are resolved as more of the sketch is drawn, yet if the system waits for the sketch to be finished, unraveling an incorrect interpretations can be a great deal of work.

In the same vein, it will be important to calibrate how important true freehand sketching is to designers. The obvious alternative is a icon-based system with graphical editing capabilities (e.g., moving and resizing the standard components). Freehand drawing can be powerful, but alternative interface styles need to be considered as well.

The system should also adapt to new users and their sketching style. For example, one of our heuristics was that people draw all of one object before moving onto the next, but there are of course exceptions. The system should be able to adjust to this type of behavior and learn to override its default heuristic.

8 Conclusion

CAD systems are rarely used in early design because they do not allow for quick and natural sketching of ideas. To be useful here, computers must allow the designer to sketch as on paper, yet provide benefits not available with paper, such as the ability to simulate the system.

To provide an interface that feels natural yet interprets sketches as the user draws, the system must be able to resolve ambiguities without interrupting the user. This work provides one solution to problem of ambiguity resolution in a framework of reasonable generality.

Acknowledgments

Luke Weisman and Mike Oltmans helped extensively in the development of these ideas and this system. The work was supported in part by the National Science Foundation, the MIT/Ford Motor Company Collaboration, and MIT's Project Oxygen.

References

1. Christine Alvarado and Randall Davis. Preserving the freedom of sketching to create a natural computer-based sketch tool. In *Human Computer Interaction International Proceedings*, 2001.
2. Ellen Yi-Luen Do and Mark D. Gross. Drawing as a means to design reasoning. *AI and Design*, 1996.
3. Mark Gross and Ellen Yi-Luen Do. Ambiguous intentions: a paper-like interface for creative design. In *Proceedings of UIST 96*, pages 183–192, 1996.
4. Mark D. Gross. Recognizing and interpreting diagrams in design. In *2nd Annual International Conference on Image Processing*, pages 308–311, 1995.
5. Mark D. Gross. The electronic cocktail napkin - a computational environment for working with design diagrams. *Design Studies*, 17:53–69, 1996.
6. Marti Hearst. Sketching intelligent systems. *IEEE Intelligent Systems*, pages 10–18, May/June 1998.
7. James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3):56–64, March 2001.
8. Jennifer Mankoff, Scott E Hudson, and Grefory D. Abowd. Providing intergrated toolkit-level support for ambiguity in recogntiion-based interfaces. In *Proceedings of the CHI 2000 conference on Human factors in computing systems*, pages 368–375, 2000.
9. N. Hari Narayanan, Masaki Suwa, and Hiroshi Motoda. *Behavior Hypothesis from Schematic Diagrams*, chapter 15, pages 501–534. The MIT Press, Cambridge, Massachusetts, 1995.
10. Michael Oltmans. Understanding natually conveyed explanations of device behavior. Master's thesis, Massachusetts Institute of Technology, 2000.

11. Dean Rubine. Specifying gestures by example. *Computer Graphics*, pages 329–337, July 1991.
12. Elisha Sacks. Automated modeling and kinematic simulation of mechanisms. *Computer-Aided Design*, 25(2):107–118, 1993.
13. Eric Saund and Thomas P. Moran. Perceptual organization in an interactive sketch editing application. In *ICCV 1995*, 1995.
14. Tevik Metin Sezgin. Early processing in sketch understanding. Master's thesis, Massachusetts Institute of Technology, 2001.
15. T. Stahovich, R. Davis, and H.Shrobe. Generalting multiple new designs from a sketch. *Artificial Intelligence*, 104(1-2):211–264, 1998.
16. David G. Ullman, Stephan Wood, and David Craig. The importance of drawing in mechanical design process. *Computer & Graphics*, 14(2):263–274, 1990.

Proceedings of the 2001 Workshop on
Perceptual User Interfaces (PUI'01),
Orlando, FL, May 2001.

Sketch Based Interfaces: Early Processing for Sketch Understanding

Tevfik Metin Sezgin,[1] Thomas Stahovich[2] and Randall Davis[1]

[1] MIT Artificial Intelligence Laboratory
`{mtsezgin, davis}@ai.mit.edu`
[2] CMU Department of Mechanical Engineering
`stahov@andrew.cmu.edu`

Abstract. Freehand sketching is a natural and crucial part of everyday human interaction, yet is almost totally unsupported by current user interfaces. We are working to combine the flexibility and ease of use of paper and pencil with the processing power of a computer, to produce a user interface for design that feels as natural as paper, yet is considerably smarter. One of the most basic steps in accomplishing this is converting the original digitized pen strokes in a sketch into the intended geometric objects. In this paper we describe an implemented system that combines multiple sources of knowledge to provide robust early processing for freehand sketching.

1 Introduction

Freehand sketching is a familiar, efficient, and natural way of expressing certain kinds of ideas, particularly in the early phases of design. Yet this archetypal behavior is largely unsupported by user interfaces in general and by design software in particular, which has for the most part aimed at providing services in the later phases of design. As a result designers either forgo tool use at the early stage or end up having to sacrifice the utility of freehand sketching for the capabilities provided by the tools. When they move to a computer for detailed design, designers usually leave the sketch behind and the effort put into defining the rough geometry on paper is largely lost.

We are working to provide a system where users can sketch naturally and have the sketches understood. By "understood" we mean that sketches can be used to convey to the system the same sorts of information about structure and behavior as they communicate to a human engineer.

Such a system would allow users to interact with the computer without having to deal with icons, menus and tool selection, and would exploit direct manipulation (e.g., specifying curves by sketching them directly, rather than by specifying end points and control points). We also want users to be able to draw in an unrestricted fashion. It should, for example, be possible to draw a rectangle clockwise or counterclockwise, or with multiple strokes. Even more generally, the system, like people, should respond to how an object looks (e.g., like a rectangle), not how it was drawn. This will, we believe, produce a sketching interface that feels much more natural, unlike Graffiti and other gesture-based systems (e.g., [9], [14]), where pre-specified motions (e.g., an L-shaped stroke or a clockwise rectangular stroke) are required to specify a rectangular shape.

The work reported here is part of our larger effort aimed at providing natural interaction with software, and with design tools in particular. That larger effort seeks to enable user to interact with automated tools in much the same manner as they interact with each other: by informal, messy sketches, verbal descriptions, and gestures. Our overall system uses a blackboard-style architecture [6], combining multiple sources of knowledge to produce a hierarchy of successively more abstract interpretations of a sketch.

Our focus in this paper is on the very first step in the sketch understanding part of that larger undertaking: interpreting the pixels produced by the user's strokes and producing low level geometric descriptions such as lines, ovals, rectangles, arbitrary polylines, curves and their combinations. Conversion from pixels to geometric objects is the first step in interpreting the input sketch. It provides a more compact representation and sets the stage for further, more abstract interpretation (e.g., interpreting a jagged line as a symbol for a spring).

2 The sketch understanding task

Sketch understanding overlaps in significant ways with the extensive body of work on document image analysis generally (e.g., [2]) and graphics recognition in particular (e.g., [16]), where the task is to go from a scanned image of, say, an engineering drawing, to a symbolic description of that drawing.

Differences arise because sketching is a realtime, interactive process, and we want to deal with freehand sketches, not the precise diagrams found in engineering drawings. As a result we are not analyzing careful, finished drawings, but are instead attempting to respond in real time to noisy, incomplete sketches. The noise is different as well: noise in a freehand sketch is typically not the small-magnitude randomly distributed variation common in scanned documents. There is also an additional source of very useful information in an interactive sketch: as we show below, the timing of pen motions can be very informative.

Sketch understanding is a difficult task in general as suggested by reports in previous systems (e.g., [9]) of a recognition rate of 63%, even for a sharply restricted domain where the objects to be recognized are limited to rectangles, circles, lines, and squiggly lines (used to indicate text).

Our domain–mechanical engineering design–presents the additional difficulty that there is no fixed set of shapes to be recognized. While there are a number of traditional symbols with somewhat predictable geometries (e.g., symbols for springs, pin joints, etc.), the system must also be able to deal with bodies of arbitrary shape that include both straight lines and curves. As consequence, accurate early processing of the basic geometry–finding corners, fitting both lines and curves–becomes particularly important.

3 System description

Sketches can be created in our system using any of a variety of devices that provide the experience of freehand drawing while capturing pen movement. We have used traditional digitizing tablets, a Wacom tablet that has an LCD-display drawing surface (so the drawing appears under the stylus), and a Mimio whiteboard system. In each case the pen motions appear to the system as mouse movements, with position sampled at rates between 30 and 150 points/sec, depending on the device and software in use.

In the description below, by a single stroke we mean the set of points produced by the drawing implement between the time it contacts the surface (mouse-down) and the time it breaks contact (mouse-up). This single path may be composed of multiple connected straight and curved segments (see, Fig. 1).

Our approach to early processing consists of three phases *approximation*, *beautification*, and *basic recognition*. Approximation fits the most basic geometric primitives–lines and curves–to a given set of pixels. The overall goal is to approximate the stroke with a more compact and abstract description, while both minimizing error and avoiding over-fitting. Beautification modifies the output of the approximation layer, primarily to make it visually more appealing without changing its meaning, and secondarily to aid the third phase, basic recognition. Basic recognition produces interpretations of the strokes, as for example, interpreting a sequence of four lines as a rectangle or square. (Subsequent recognition, at the level of mechanical components, such as springs, and pin joints is accomplished by another of our systems [1]).

3.1 Stroke approximation

Stroke processing consists of detecting vertices at the endpoints of linear segments of the stroke, then detecting and characterizing curved segments of the stroke.

Vertex detection We use the sketch in Fig. 1 as a motivating example of what should be done in the vertex detection phase. Points marked in Fig. 1 indicate the corners of the stroke, where the local curvature is high.

Note that the vertices are marked only at what we would intuitively call the corners of the stroke (i.e., endpoints of linear segments). There are, by design, no vertices marked on curved portions of the stroke because we want to handle these separately, modeling them with curves (as described below). This is unlike the well studied problem of piecewise linear approximation [13].

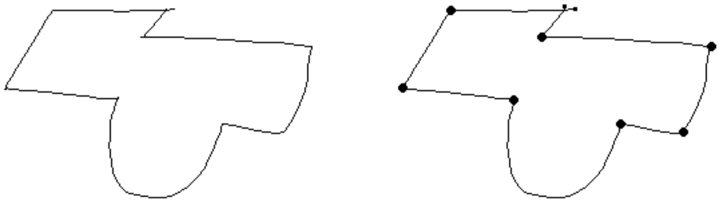


Fig. 1. The stroke on the left contains both curves and straight line segments. The points we want to detect in the vertex detection phase are indicated with large dots in the figure on the right. The beginning and the end points of the stroke are indicated with smaller dots.

Our approach takes advantage of the interactive nature of sketching, combining information from both stroke direction and speed data.

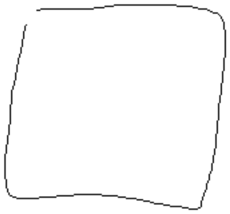


Fig. 2. Stroke representing a square.

Consider as an example the square in Fig. 2; Fig. 3 shows the direction, curvature (change in direction with respect to arc length) and speed data for this stroke. We locate vertices by looking for points along the stroke that are minima of speed (the pen slows at corners) or maxima of the absolute value of curvature.[1]

While extrema in curvature and speed typically correspond to vertices, we cannot rely on them blindly because noise in the data introduces many false positives. To deal with this we use average based filtering.

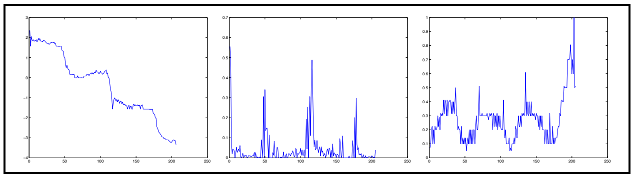


Fig. 3. Direction, curvature and speed graphs for the stroke in Fig. 2

Average based filtering

We want to find extrema corresponding to vertices while avoiding those due to noise. To increase our chances at doing this, we look for extrema in those portions of the curvature and speed data that lie beyond a threshold. Intuitively, we are looking for maxima of curvature only where the curvature is already high and minima of speed only where the speed is already low. This will help to avoid selecting false positives

---

[1] From here on for ease of description we use curvature to mean the absolute value of the curvature data.

of the sort that would occur say, when there is a brief slowdown in an otherwise fast section of a straight stroke.

To avoid the problems posed by choosing a fixed threshold, we set the threshold based on the mean of each data set.[2] We use these thresholds to separate the data into regions where it is above/below the threshold and select the global extrema in each region that lies above the threshold.

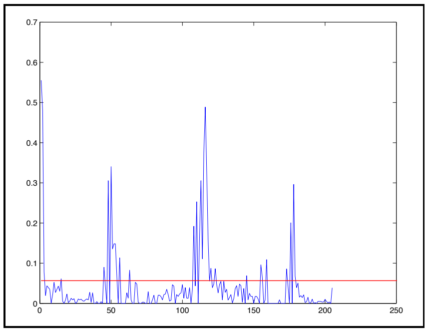


Fig. 4. Curvature graph for the square in Fig. 2 with the threshold dividing it into regions.

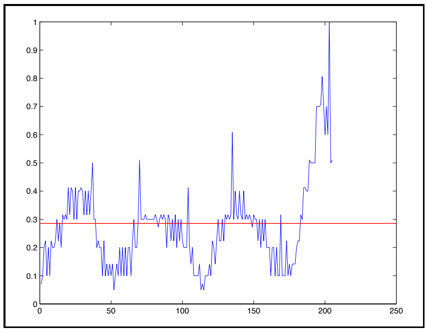


Fig. 5. Speed graph for the stroke in Fig. 2 with the threshold dividing it into regions.

---

[2] The exact threshold has been determined empirically; for curvature data the threshold is the mean, while for the speed the threshold is 90% of the mean.

Application to curvature data

Fig. 4 shows the curvature graph partitioned into regions of high and low curvature. Note that this reduces but doesn't eliminate the problem of false positives introduced by noise in the stroke. We deal with the false positives using the hybrid fit generation scheme described below.[3]

While average based filtering performs better than simply comparing the curvature data against a hard coded threshold, it is still clearly not free of empirical constants. As we explain when considering future work, scale space provides a better approach for dealing with noisy data without having to make a priori assumptions about the scale of relevant features.

Application to speed change

Our experience is that curvature data alone rarely provides sufficient reliability. Noise is one problem, but variety in angle changes is another. Fig. 6 illustrates how curvature fit alone misses a vertex (at the upper right) because the curvature around that point was too small to be detected in the context of the other, larger curvatures. We solve this problem by incorporating speed data into our decision as an independent source of guidance.

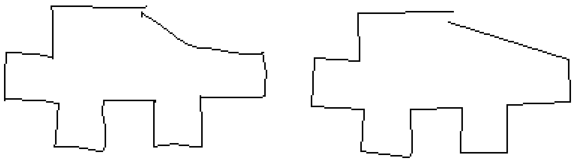


Fig. 6. At left the original sketch of a piece of metal; at right the fit generated using only curvature data.

---

[3] An alternative approach is to detect consecutive almost-collinear edges (using some empirical threshold for collinearity) and combine them into one edge, removing the vertex in between. Our hybrid fit scheme deals with the problem without the need to decide what value to use for "almost-collinear."

Just as we did for the curvature data, we reduce the number of false extrema by average based filtering, then look for speed minima. The intuition here is simply that pen speed drops when going around a corner in the sketch. Fig. 7 shows (at left) the speed data for the sketch in Fig. 6, along with the polygon drawn from the speed-detected vertices (at right).

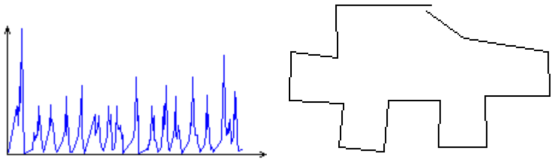


Fig. 7. At left the speed graph for the piece; at right the fit based on only speed data.

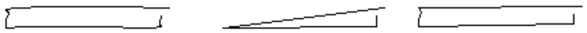
Using speed data alone has its shortcomings as well. Polylines formed from a combination of very short and long line segments can be problematic: the maximum speed reached along the short line segments may not be high enough to indicate the pen has started traversing another edge, with the result that the entire short segment is interpreted as the corner. This problem arises frequently when drawing thin rectangles, common in mechanical devices. Fig. 8 illustrates this phenomena. In this figure, the speed fit misses the upper left corner of the rectangle because the pen failed to gain enough speed between the endpoints of the short vertical segment. The curvature fit, by contrast, detects all corners, along with some other vertices that are artifacts due to hand dynamics during freehand sketching. This illustrates the utility of having both fits available.

We use information from both sources, generating hybrid fits by combining the set of candidate vertices derived from curvature data F_d with the candidate set from speed data F_s, taking into account the system's certainty that each candidate is a real vertex.

Generating hybrid fits

Hybrid fit generation occurs in three stages: computing vertex certainties, generating a set of hybrid fits, and selecting the best fit.

Our certainty metric for a curvature candidate vertex v_i is the scaled magnitude of the curvature in a local neighborhood around the point, computed as $|d_{i-k} - d_{i+k}|/l$. Here l is the curve length between points



(a) Input, 63 points

(b) Using speed data, 4 vertices

(c) Using curvature data, 7 vertices

Fig. 8. Average based filtering using speed data misses a vertex. The curvature fit detects the missed point (along with vertices corresponding to the artifact along the left edge of the rectangle).

S_{i-k}, S_{i+k} and k is a small integer defining the neighborhood size around v_i. The certainty metric for a speed fit candidate vertex v_i is a measure of the pen slowdown at the point, $1 - v_i/v_{max}$, where v_{max} is the maximum pen speed in the stroke. The certainty values are normalized to $[0, 1]$.

While both of these metrics are designed to produce values in $[0, 1]$, they have different scales. As the metrics are used only for ordering within each set, they need not be numerically comparable across sets. Candidate vertices are sorted by certainty within each fit.

The initial hybrid fit H_0 is the intersection of F_d and F_s. A succession of additional fits is then generated by appending to H_i the highest scoring curvature and speed candidates not already in H_i.

To do this, on each cycle we create two new fits: $H_i' = H_i + v_s$ (i.e., H_i augmented with the best remaining speed fit candidate) and $H_i'' = H_i + v_d$ (i.e., H_i augmented with the best remaining curvature candidate). We use least squares error as a metric of the goodness of a fit: the error ε_i is computed as the average of the sum of the squares of the distances to the fit from each point in the stroke S:

$$\varepsilon_i = \frac{1}{|S|} \sum_{s \in S} ODSQ(s, H_i)$$

Here $ODSQ$ stands for *orthogonal distance squared*, i.e., the square of the distance from the stroke point to the relevant line segment of the polyline defined by H_i. We compute the error for H_i' and for H_i''; the higher scoring of these two (i.e., the one with smaller least squares error) becomes H_{i+1}, the next fit in the succession. This process continues until all points in the speed and curvature fits have been used. The result is a set of hybrid fits.

In selecting the best of the hybrid fits the problem is as usual trading off more vertices in the fit against lower error. Here our approach is simple: We set an error upper bound and designate as our final fit H_f, the H_i with the fewest vertices that also has an error below the threshold.

Handling curves The approach described thus far yields a good approximation to strokes that consists solely of line segments, but as noted our input may include curves as well, hence we require a means of detecting and approximating them.

The polyline approximation H_f generated in the process described above provides a natural foundation for detecting areas of curvature: we compare the Euclidean distance l_1 between each pair of consecutive vertices in H_f to the accumulated arc length l_2 between those vertices in the input S. The ratio l_2/l_1 is very close to 1 in the linear regions of S, and significantly higher than 1 in curved regions.

We approximate curved regions with Bézier curves, defined by two end points and two control points. Let $u = S_i$, $v = S_j$, $i < j$ be the end points of the part of S to be approximated with a curve. We compute the control points as:

$$c_1 = k\hat{t}_1 + v$$

$$c_2 = k\hat{t}_2 + u$$

$$k = \frac{1}{3} \sum_{i \le k < j} |S_k - S_{k+1}|$$

where $\hat{t}_1$ and $\hat{t}_2$ are the unit length tangent vectors pointing inwards at the curve segment to be approximated. The 1/3 factor in k controls how much we scale $\hat{t_1}$ and $\hat{t_2}$ in order to reach the control points; the summation is simply the length of the chord between S_i and S_j.[4]

As in fitting polylines, we want to use least squares to evaluate the goodness of a fit, but computing orthogonal distances from each S_i in the input stroke to the Bézier curve segments would require solving a fifth degree polynomial. (Bézier curves are described by third degree polynomials, hence computing the minimum distance from an arbitrary point to the curve involves minimizing a sixth degree polynomial, equivalent to solving a fifth degree polynomial.) A numerical solution is both

---

[4] The 1/3 constant was determined empirically, but works very well for freehand sketches. As we discovered subsequently, the same constant was independently chosen in [15].

computationally expensive and heavily dependent on the goodness of the initial guesses for roots [12], hence we resort to an approximation. We discretize the Bézier curve using a piecewise linear curve and compute the error for that curve. This error computation is $O(n)$ because we impose a finite upper bound on the number of segments used in the piecewise approximation.

If the error for the Bézier approximation is higher than our maximum error tolerance, the curve is recursively subdivided in the middle, where middle is defined as the data point in the original stroke whose index is midway between the indices of the two endpoints of the original Bézier curve. New control points are computed for each half of the curve, and the process continues until the desired precision is achieved.

Examples of the capability of our approach is shown in Fig. 9, a hastily-sketched mixture of lines and curves. Note that all of the curved segments have been modeled curves, rather than the piecewise linear approximations that have been widely used previously.

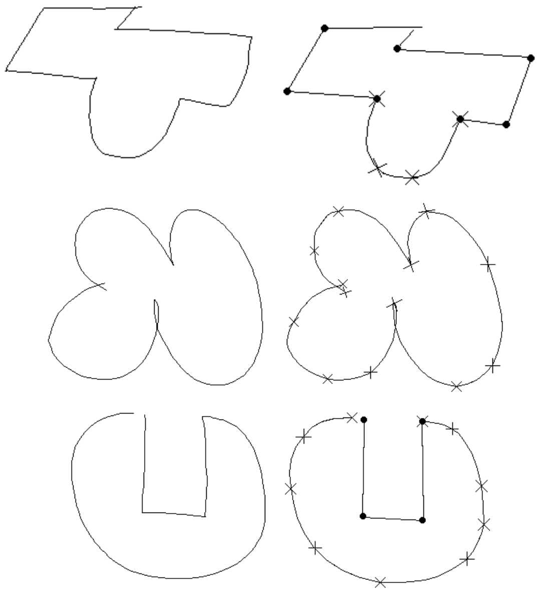


Fig. 9. Examples of arbitrary stroke approximation. Boundaries of Bézier curves are indicated with crosses, detected vertices are indicated with dots.

3.2 Beautification

Beautification refers to the (currently minor) adjustments made to the approximation layer's output, primarily to make it look as intended. We adjust the slopes of the line segments in order to ensure the lines that were apparently meant to have the same slope end up being parallel. This is accomplished by looking for clusters of slopes in the final fit produced by the approximation phase, using a simple sliding-window histogram. Each line in a detected cluster is then rotated around its midpoint to make its slope be the weighted average of the slopes in that cluster. The (new) endpoints of these line segments are determined by the intersections of each consecutive pair of lines. This process (like any neatening of the drawing) may result in vertices being moved; we chose to rotate the edges about their midpoints because this produces vertex locations that are close to those detected, have small least square errors when measured against the original sketch, and look right to the user. Fig. 10 shows the original stroke for the metal piece we had before, and the output of the beautifier. Some examples of beautification are also present in Fig. 13.

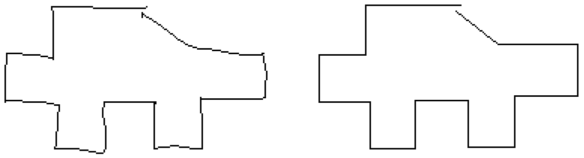


Fig. 10. At left the original sketch of a piece of metal revisited, and the final beautified output at right.

3.3 Basic object recognition

The final step in our processing is recognition of the most basic objects that can be built from the line segments and curve segments produced thus far, i.e., simple geometric objects (ovals, circles, rectangles, squares).

Recognition of these objects is done with hand-tailored templates that examine various simple properties. A rectangle, for example, is recognized as a polyline with 4 segments all of whose vertices are within a specified distance of the center of the figure's bounding box; a stroke will be recognized as an oval if it has a small least squares error when compared to an oval whose axes are given by the bounding box of the stroke.

3.4 Evaluation

We have conducted a user study to measure the degree to which the system is perceived as easy to use, natural and efficient. Study participants were asked to create a set of shapes using our system and Xfig, a Unix tool for creating diagrams. Xfig is a useful point of comparison because it is representative of the kinds of tools that are available for drawing diagrams using explicit indication of shape (i.e., the user indicates explicitly which parts of the sketch are supposed to be straight lines, which curves, etc.) As in other such tools, XFig has a menu and toolbar interface; the user selects a tool (e.g., for drawing polygons), then creates the shapes piece by piece.

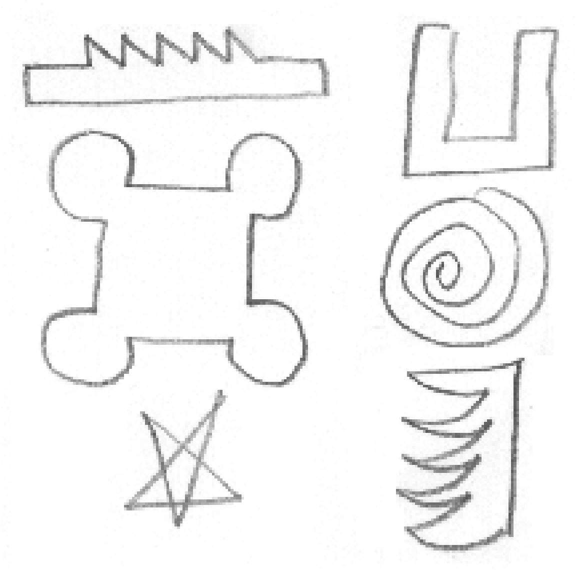


Fig. 11. Examples of the shapes used in the user study.

Thirteen subjects participated in our study, including computer science graduate students, computer programmers and an architecture student. Subjects were given sufficient time to get familiar with each system and then asked to draw a set of 10 shapes (examples given in Fig 11). All of the subjects reported our system being easier to use, efficient and more natural feeling. The subjects were also asked which system they would prefer when drawing these sort of informal shapes on a computer. All but one subject preferred our system; the sole dissenter preferred a tablet surface that had the texture and feel of paper.

Overall users praised our system because it let them draw shapes containing curves and lines directly and without having to switch back and forth between tools. We have also observed that with our system, users found it much easier to draw shapes corresponding to the gestures they routinely draw freehand, such as a star.

While the central point of this comparison was to determine how natural it felt to use each system, we also evaluated our system's ability to produce a correct interpretation of each shape (i.e., interpret strokes appropriately as lines or curves). Overall the system's identification of the vertices and approximation of the shapes with lines and curves was correct 96% of the time on the ten figures.

In addition to the user studies we have conducted, we wrote a higher level recognizer for evaluation purposes. The higher level recognizer takes the geometric descriptions generated by the basic object recognition module of our system and combines them into domain specific objects.

Fig. 13 shows the original input and the program's analysis for a variety of simple but realistic mechanical devices drawn as freehand sketches. The last two of them are different sketches for a part of the direction reversing mechanism for a tape player. Recognized domain specific components include gears (indicated by a circle with a cross), springs (indicated by wavy lines), and the standard fixed-frame symbol (a collection of short parallel lines). Components that are recognized are replaced with standard icons scaled to fit the sketch.

An informal comparison of the raw sketch and the system's approximations shows whether the system has selected vertices where they were drawn, fit lines and curves accurately, and successfully recognized basic geometric objects. While informal, this is an appropriate evaluation because the program's goal is to produce an analysis of the strokes that "looks like" what was sketched.

We have also begun to deal with overtracing, one of the (many) things that distinguishes freehand sketches from careful diagrams. Fig. 12 illustrates one example of the limited ability we have thus far embodied in the program.

Fig. 12. An overtraced oval and a line along with and the system's output.

4 Related work

In general, systems supporting freehand sketching lack one or more of the properties that we believe a sketching system should have:

- It should be possible to draw arbitrary shapes with a single stroke, (i.e., without requiring the user to draw objects in pieces).
- The system should do automatic feature point detection. The user should not have to specify vertex positions by hand.
- The system should not have sketching modes for drawing different geometric object classes (i.e., modes for drawing circles, polylines, curves etc.).
- The sketching system should feel natural to the user.

The Phoenix sketching system [15] had some of the same motivation as our work, but a more limited focus on interactive curve specification. While the system provided some support for vertex detection, its focus on curves led it to use Gaussian filters to smooth the data. While effective for curves, Gaussians tend to treat vertices as noise to be reduced, obscuring vertex location. As a result the user was often required to specify the vertices manually.

Work in [5] describes a system for sketching with constraints that supports geometric recognition for simple strokes (as well as a constraint maintenance system and extrusion for generating solid geometries). The set of primitives is more limited than ours: each stroke is interpreted as a line, arc or as a Bézier curve. More complex shapes can be formed by combinations of these primitives, but only if the user lifts the pen at the end of each primitive stroke, reducing the feeling of natural sketching.

The work in [3] describes a system for generating realtime spline curves from interactively sketched data. They focus on using knot removal techniques to approximate strokes known to be composed only of curves, and do not handle single strokes that contain both lines and curves. They do not support corner detection, instead requiring the user to specify corners and discontinuities by lifting the mouse button, or equivalently by lifting the pen. We believe our approach of automatically detecting the feature points provides a more natural and convenient sketching interface.

Zeleznik [7] describes a mode-based stroke approximation system that uses simple rules for detecting the drawing mode. The user has to draw objects in pieces, reducing the sense of natural sketching. Switching modes is done by pressing modifier buttons in the pen or in the keyboard. In this system, a click of the mouse followed by immediate dragging signals that the user is drawing a line. A click followed by a pause and then dragging of the mouse tells the system to enter the freehand curve mode. Our system allows drawing arbitrary shapes without any restriction on how the user draws them. There is enough information provided by the freehand drawing to differentiate geometric shapes such as curves, polylines, circles and lines from one another, so we believe requiring the user to draw things in a particular fashion is unnecessary and reduces the natural feeling of sketching. Our goal is to make computers understand what the user is doing rather than requiring the user to sketch in a way that the computer can understand.

Among the large body of work on beautification, Igarashi et al. [8] describes a system combining beautification with constraint satisfaction, focusing on exploiting features such as parallelism, perpendicularity, congruence and symmetry. The system infers geometric constraints by comparing the input stroke with previous ones. Because sketches are inherently ambiguous, their system generates multiple interpretations corresponding to different ways of beautifying the input, and the most plausible interpretation is chosen among these interpretations. The system is interactive, requiring the user to do the selection, and doesn't support curves. It is, nevertheless, more effective then ours at beautification, but beautification is not the main focus of our work and is present for the purposes of completeness.

The works in [15] and [3] describe methods for generating very accurate approximations to strokes known to be curves with precision several orders of magnitude below the pixel resolution. The Bézier approximations we generate are less precise but are sufficient for approximating free-hand curves. We believe techniques in [15] and [3] are excessively precise for free-hand curves, and the real challenge is detecting curved regions in a stroke rather than approximating those regions down to the numerical machine precision.

5 Future work

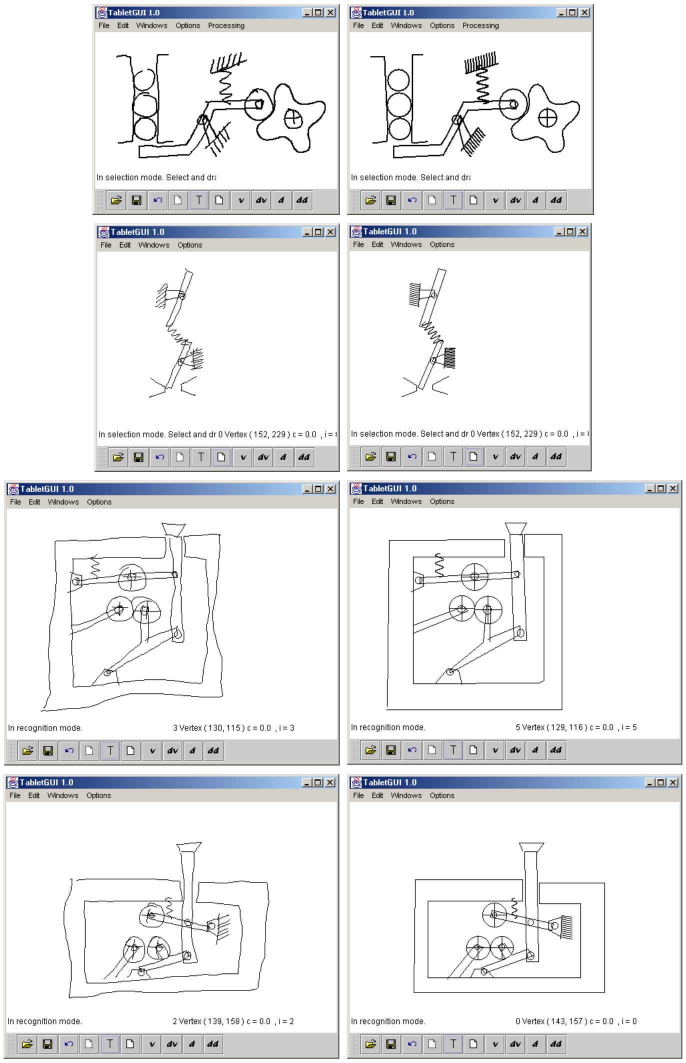
We are working to link this early processing to other work in our group that has focused on recognition [1] of higher level mechanical objects. This will provide the opportunity to add model-based processing of the stroke, in which early operations like vertex localization may be usefully guided by knowledge of the current best recognition hypothesis.

In addition, incorporating ideas from scale space theory looks like a promising way of detecting different scales inherent in the data and avoiding *a priori* judgments about the size of relevant features. In the pattern recognition community [4], [11] and [10] apply some of the ideas from scale space theory to similar problems. We are currently working on ways of applying these techniques to speed and curvature data. We believe this may allow us to deal more effectively with sketches that contain relevant details at a variety of scales. There is no guaranteed way of deciding which scales are important at the geometric level, so using constraints and/or information provided by the domain of application may help in scale selection.

Humans naturally seem to slow down when they draw things carefully as opposed to casually, so another interesting research direction would be to explore the degree to which one can use the time it takes to draw a stroke as an indication of how careful and precise the user meant to be.

6 Conclusion

We have built a system capable of using multiple sources of information to produce good approximations of freehand sketches. Users can sketch on an input device as if drawing on paper and have the computer detect the low level geometry, enabling a more natural interaction with the computer, as a first step toward more natural user interfaces generally, and toward earlier use of automated tools in the design cycle in particular.



In selection mode. Select and dr

In selection mode. Select and dr

In selection mode. Select and dr 0 Vertex (152, 229) c = 0.0 , i =

In selection mode. Select and dr 0 Vertex (152, 229) c = 0.0 , i =

In recognition mode. 3 Vertex (130, 115) c = 0.0 , i = 3

In recognition mode. 5 Vertex (129, 116) c = 0.0 , i = 5

In recognition mode. 2 Vertex (139, 158) c = 0.0 , i = 2

In recognition mode. 0 Vertex (143, 157) c = 0.0 , i = 0

Fig. 13. Performance examples: The first two pair are sketches of a marble dispenser mechanism and a toggle switch. The last two are sketches of the direction reversing mechanism in a tape player.

References

1. Christine Alvarado. A natural sketching environment: Bringing the computer into early stages of mechanical design. Master's thesis, Massachusetts Institute of Technology, 2000.
2. H. S. Baird, H. Bunke, and K. Yamamoto. Structured document image analysis. Springer-Verlag, Heidelberg, 1992.
3. M. Banks and E. Cohen. Realtime spline curves from interactively sketched data. In *SIGGRAPH, Symposium on 3D Graphics*, pages 99–107, 1990.
4. A. Bentsson and J. Eklundh. Shape representation by multiscale contour approximation. *IEEE PAMI 13, p. 85–93, 1992.*, 1992.
5. L. Eggli. Sketching with constraints. Master's thesis, University of Utah, 1994.
6. Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12:213–253, 1980. Reprinted in: *Readings in Artificial Intelligence*, Bonnie L. Webber and Nils J. Nilssen (eds.)(1981), pp 349-389. Morgan Kaufman Pub. Inc., Los Altos, CA.
7. R. Zeleznik et al. Sketch: An interface for sketching 3d scenes. In *Proceedings of SIGGRAPH'96*, pages 163–170, 1996.
8. T. Igarashi et. al. Interactive beautification: A technique for rapid geometric design. In *UIST '97*, pages 105–114, 1997.
9. James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer, vol. 34, no. 3, March 2001, pp. 56-64.*
10. T. Lindeberg. Edge detection and ridge detection with automatic scale selection. *ISRN KTH/NA/P–96/06–SE, 1996.*, 1996.
11. A. Rattarangsi and R. T. Chin. Scale-based detection of corners of planar curves. *IEEE Transactionsos Pattern Analysis and Machine Intelligence*, 14(4):430–339, April 1992.
12. N. Redding. Implicit polynomials, orthogonal distance regression, and closest point on a curve. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 191–199, 2000.
13. R. Rosin. Techniques for assessing polygonal approximations of curves. *7th British Machine Vision Conf., Edinburgh*, 1996.
14. Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25(4):329–337, 1991.
15. P. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, University of Washington, 1988.
16. K. Tombre. Analysis of engineering drawings. In *GREC 2nd international workshop*, pages 257–264, 1997.

MIT Student Oxygen Workshop, 2001.

Free-Hand Stroke Approximation for Intelligent Sketching Systems

Tevfik Metin Sezgin

MIT Artificial Intelligence Laboratory
`mtsezgin@ai.mit.edu`

Abstract. Free-hand sketching is used extensively during the early design phases as an important tool for conveying ideas, guiding the thought process, and serving as documentation [5]. It also provides a natural way of interaction desirable in the context of E21 and intelligent design spaces. Unfortunately there is little computer support for sketching. The first step in building a sketch understanding system is generating more meaningful descriptions of free-hand strokes. We describe a system that takes strokes described by an array of points along with timing data, and generates such concise descriptions in terms of geometric primitives such as circles, polylines, curves and their combinations.

1 System description

1.1 Issues

Loss of information due to digitization, noise in the data, imprecision due to freehand sketching and sparseness of sampled data points – which may be as low as 4-5 dpi as opposed to scanned drawings that may have 1200-2400 dpi – complicate the stroke approximation task.

1.2 Feature point detection

Stroke processing starts by looking for vertices. We want to avoid picking points on the curved regions as much as possible[1]. Vertex localization is a frequent subject in the extensive literature on graphics recognition (e.g., [4] compares 21 methods). However these methods produce piecewise linear approximations.

---

[1] Piecewise linear approximation algorithms don't satisfy this requirement.

Our approach takes advantage of the interactive nature of sketching by combining information from both curvature and speed data for detecting corners while avoiding a piecewise linear approximation. Feature points are indicated by the maxima of curvature[2] data and the minima of the pen speed. Both speed and curvature data are noisy and simply picking the extrema introduces many false positives. Below we describe two methods for selecting the extrema in each case.

Average based filtering The basic idea in average based filtering is confining our search for the vertices to only those regions where the curvature/speed data is above/below a threshold computed by the mean of the data. Then we search for the global extremum within each region to find the feature points.

Scale space filtering Our experiments show that the above method is satisfactory for settings where noise is relatively small but doesn't deal with extremely noisy data as in Fig. 1 very well, because it doesn't differentiate between vertices due to the fine scale structure of the noise and those due to the high level structure of the stroke. The scale space based method deals with this problem by looking at the number of feature points present at different scales in the scale space. As seen in Fig. 2, the feature count graph has two distinct regions where the feature count drops with different rates. This behavior is typical for freehand strokes.

Our task is selecting a scale where most of the noise is filtered out. This is done by modeling the feature count graph by fitting two lines to it: one to the region with the steep drop corresponding to places where the feature points due to noise disappear and the other to the flatter region where real feature points disappear. We take the scale corresponding to the intersection of these two lines as our scale.

The results obtained by applying this technique to curvature and speed data are in Fig. 3.

Generating hybrid fits The above methods separately may miss feature points, so we combine them to generate hybrid fits. We start the hybrid fit generation by letting the intersection of the generated fits

---

[2] From this point on, by curvature we will refer to the absolute value of the curvature data defined as $|\partial d/\partial s|$ where d is the angle between the tangent to the curve at a point and the x axis and s is the distance traveled along the curve.

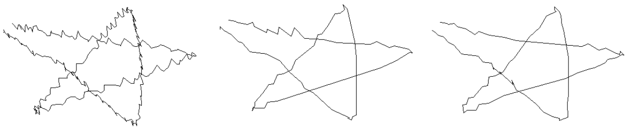


Fig. 1. A very noisy stroke along with the fits generated using curvature and speed data with average based filtering. Both fits have picked many false positives due to noise.

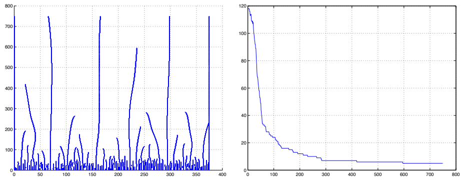


Fig. 2. The scale space and feature count graph for the noisy stroke in Fig. 1.

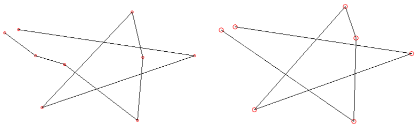


Fig. 3. Curvature and speed fits generated using the scale space approach. These fits contain 9 vertices for the curvature and 7 vertices for the speed fit compared to 69 and 82 vertices for the fits obtained using average based filtering in Fig. 1.

to be our initial hybrid fit. At each iteration, we form a list of candidate vertices by picking the vertex with the highest certainty from each of the original fits. We augment the most recent hybrid fit by the candidate from this list that reduces the least squares error of the fit by the largest amount.

1.3 Handling curves

Our system also supports strokes that may contain curves. First we detect curved regions by comparing the curve length l_1 between the consecutive detected feature points to the Euclidean distance l_2 between the two points. l_2/l_1 is significantly larger than 1 in curved regions. The curved regions are then approximated by Bézier curves.

2 Evaluation

For purposes of evaluation, we built a higher level recognizer that takes the output of our system and interprets strokes in the mechanical engineering domain. Fig. 4 shows a number of free hand sketches and the output of the recognizer[3].

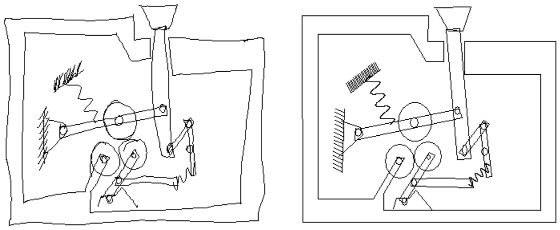


Fig. 4. Input-output figures for the rough sketch of the direction reversal mechanism of a walkman.

3 Related work

Related work can be found in [2, 1, 6, 3] but these systems either don't support drawing arbitrary shapes or don't do automatic vertex detec-

---

[3] Some domain specific objects such as springs and ground symbols are recognized by the recognizer.

tion. Some of them require drawing modes making them unnatural freehand sketching interfaces.

4 Future work

Future directions include potential improvements, user studies and integration with other systems including systems that can learn and classify strokes, patterns or objects taking the concise representation of strokes generated by our system as inputs.

5 Acknowledgements

I would like to thank my thesis advisor Prof. Randall Davis for his supervision.

References

1. L. Eggli. Sketching with constraints. Master's thesis, University of Utah, 1994.
2. James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer, vol. 34, no. 3, March 2001, pp. 56-64.*, 2001.
3. A. Rattarangsi and R. T. Chin. Scale-based detection of corners of planar curves. *IEEE Transactionsos Pattern Analysis and Machine Intelligence*, 14(4):430–339, April 1992.
4. R. Rosin. Techniques for assessing polygonal approximations of curves. *7th British Machine Vision Conf., Edinburgh*, 1996.
5. David G. Ullman, Stephen Wood, and David Craig. The importance of drawing in the mechanical design process. *Computers and Graphics*, 14(2):263–274, 1990.
6. R. Zeleznik. Sketch: An interface for sketching 3d scenes. In *Proceedings of SIGGRAPH'96*, pages 163–170, 1996.

Proceedings of the 2001 Workshop on
Perceptual User Interfaces (PUI'01),
Orlando, FL, May 2001.

Naturally Conveyed Explanations of Device Behavior

Michael Oltmans and Randall Davis

MIT Artificial Intelligence Laboratory
{`moltmans, davis`}`@ai.mit.edu`

Abstract. Designers routinely explain their designs to one another using sketches and verbal descriptions of behavior, both of which can be understood long before the device has been fully specified. But current design tools fail almost completely to support this sort of interaction, instead not only forcing designers to specify details of the design, but typically requiring that they do so by navigating a forest of menus and dialog boxes, rather than directly describing the behaviors with sketches and verbal explanations. We have created a prototype system, called ASSISTANCE, capable of interpreting multimodal explanations for simple 2-D kinematic devices. The program generates a model of the events and the causal relationships between events that have been described via hand drawn sketches, sketched annotations, and verbal descriptions. Our goal is to make the designer's interaction with the computer more like interacting with another designer. This requires the ability not only to understand physical devices but also to understand the means by which the explanations of these devices are conveyed.

1 Introduction

When a mechanical designer explains a device to a colleague, s/he does so with sketches and verbal explanations of the device's behavior. When specifying the same device in a CAD system, however, the interaction is not nearly as natural, in either the medium of expression or the content expressed. The designer must use a mouse and keyboard to specify a substantial body of detailed information (e.g., spring constants) that is not the primary concern in early design stages. This state of affairs remains true even a decade after work indicated that the formality and rigidity of CAD systems can significantly hinder the early stages of the design process [17].

We have been working to remove this barrier between designers and their tools by shifting the emphasis away from parametric specifications and towards multimodal explanations of behavior. We have constructed a system, called ASSISTANCE, that allows the user to describe a device's behavior using hand drawn sketches, sketched annotations, and verbal descriptions phrased in the same vocabulary designers routinely use when talking to one another. From this information ASSISTANCE generates a model that represents how the components move and what causal relationships exist between those movements. As we illustrate, this process both provides a more natural interface for the designer and allows the system to infer some useful details about the design of the device. In the near term the model constructed by the system will be used to inform a mechanical simulator, so the sketched device can be animated, while in the longer term we envision systems using ASSISTANCE's representations to reason about design rationale by tracking changes in both the design's structure and behavior during the early stages of design.

This paper reviews an example of the system in operation, explains what knowledge is required to support the inferences it makes, and examines both its capabilities and limitations.

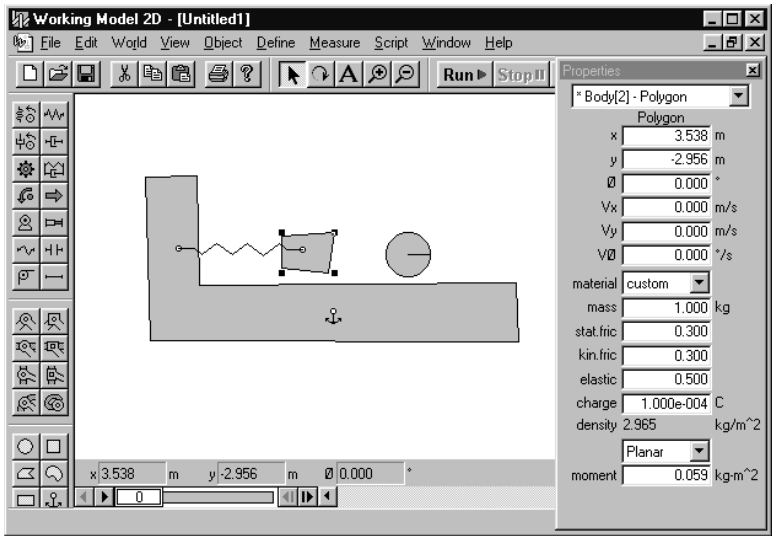
2 Describing structure and behavior

Enabling designers to describe devices to a computer as naturally as they would to a colleague requires understanding descriptions of both structure and behavior. For a mechanical device, the structure represents the device's components and their physical interconnections while the behavior represents how the components move and the relationships between these motions. The problem of understanding structural descriptions using natural media like hand drawn sketches has been explored in our group [1] and a few other efforts (e.g., [9]) but there has been very little work on understanding similarly natural descriptions of behavior.

To make description feel natural, we have to attend to both the medium of expression and the content being expressed. Designers of all sorts feel natural drawing and talking about their designs, particularly in the early, conceptual stages of the process. Designers also find a particular kind of content natural at this stage: they typically pay more attention to the behavior of the device than the properties of the individual components [2]. This demands that we enable descriptions to be phrased in the sort of language and vocabulary typically employed.

As a trivial yet instructive example, consider a spring attached to a block positioned next to a ball. In a traditional CAD system (Fig. 1) the designer would select the components from a tool bar and position them, and would then have to specify a variety of parameters, such as the rest length of the spring, the spring constant, etc.

Contrast this to the way someone would describe this device to a colleague. As we discovered in a set of informal experiments, the description typically consists of a quick hand drawn sketch (e.g., Fig. 2) and a brief spoken description, "the block pushes the ball." We have built a tool that augments structural descriptions by understanding these sorts of graphical and verbal descriptions of behavior.



* Body[2] - Polygon

Polygon

x	3.538	m
y	-2.956	m
Ø	0.000	°
Vx	0.000	m/s
Vy	0.000	m/s
VØ	0.000	°/s
material	custom	
mass	1.000	kg
stat.fric	0.300	
kin.fric	0.300	
elastic	0.500	
charge	1.000e-004	C
density	2.965	kg/m^2
	Planar	
moment	0.059	kg-m^2

x 3.538 m y -2.956 m Ø 0.000 °

0

Fig. 1. A block and spring described using a CAD tool.

3 Overview and capabilities

To use the system a designer first sketches the device, using a system called ASSIST [1], which interprets the sketch and generates a representation of device structure. The designer then switches to an explanation

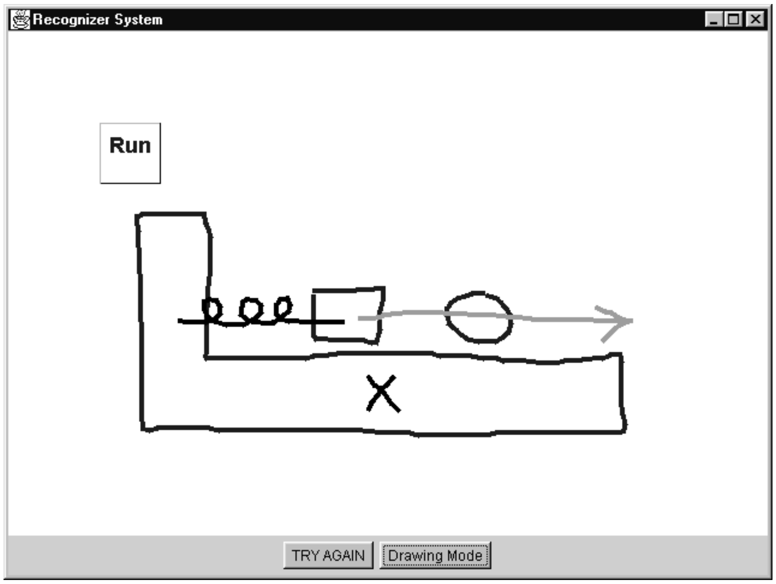


Fig. 2. A more natural medium of description.

mode and explains the device's behavior by drawing arrows, speaking, and pointing. After each of the designer's explanation fragments (i.e., each utterance and gesture) the system interprets that explanation fragment, updating its model of devices. At any time the designer can verbally ask the system to explain its causal model.

ASSISTANCE can currently understand descriptions of two dimensional kinematic devices that use rigid bodies, pin joints, pulleys, rods, and springs. It takes spoken natural language and hand-drawn sketches as input and generates a causal model that describes the actions the device performs and the causal connections between them.

We take "understanding" in this context to mean the ability to generate such a causal model, that accurately reflects the behavior description given by the designer. The system's task is thus to understand the designer, without attempting to determine whether the designer's description is physically accurate.

The representations ASSISTANCE generates are not a verbatim recording of the designer's description. To demonstrate that it has understood an explanation (and not just recorded it), ASSISTANCE can construct simple explanations about the role of each component in terms of the events that it is involved in and causal connections between events. Further evidence of the system's understanding is provided by its ability to infer from the behavior description what values some device parameters (e.g., spring constants) must take on in order to be consistent with the description. The query and parameter adjustment capabilities were designed to provide a mechanism for the system to describe its internal model and to suggest how such representations could be used in the future.

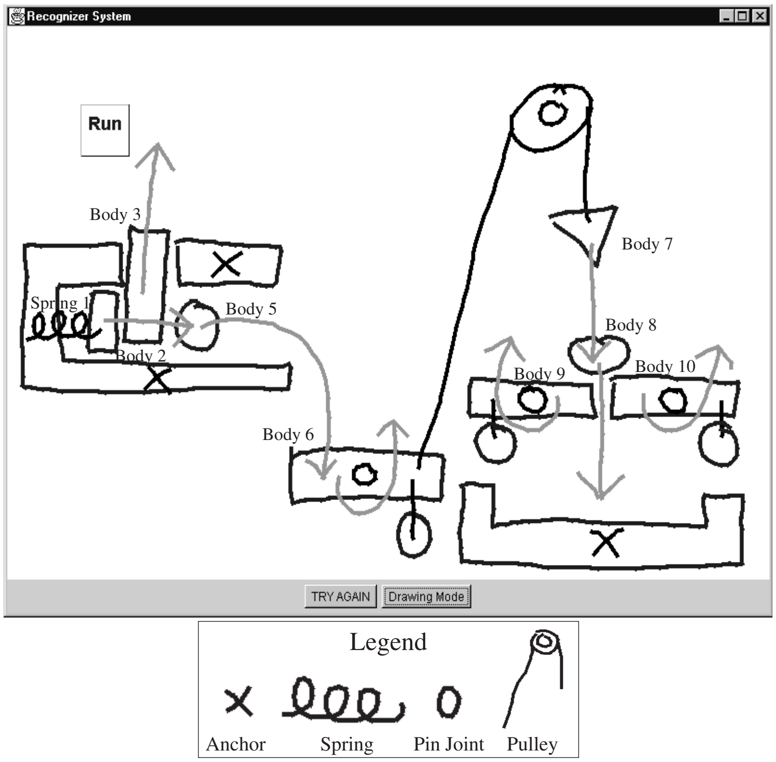
The current implementation of ASSISTANCE is made tractable by taking advantage of a number of sources of knowledge and focusing the scope of the task. We currently focus on two-dimensional kinematic devices, thereby limiting the vocabulary and grammar necessary to describe a device, making the language understanding problem in turn tractable. We then take advantage of two characteristics of informal behavior descriptions: they typically contain overlapping information and they are often expressed in stereotypical forms. We use the multiple, overlapping descriptions of an event–the same event described in a verbal explanation and in a sketched annotation–to help infer the meaning of the description. We also combine multiple descriptions to produce a richer description than either one provides alone. Finally, we use knowledge about the way designers describe devices to simplify the process of interpreting their descriptions (e.g., mechanical device behavior is frequently described in the order in which it occurs).

4 An example

An example will help demonstrate the types of multi-modal explanations ASSISTANCE understands and the types of inferences it can make. Fig. 3 shows a Rube Goldberg-style egg-cracking device (adapted from [12]), along with an explanation of its behavior (the arrows and verbal statements).

From this explanation the system generates a model of the device's behavior. The model (described in more detail below) consists of 8 events, one motion event for each of bodies 2, 3, and 5-10.

Note that in the absence of the information provided by the verbal and gestural annotations of Fig. 3, a simulation of the device produces useless behavior: body 3 will simply drop the small remaining distance to the frame, and sit there, preventing the spring from propelling the ball, while the ball will similarly drop the small distance and remain in place. Allowing the designer to explain how the device should work allows the system to construct a model that can be used to inform a simulator, so that intended behavior of the current design can be visualized.



“When body 3 moves up spring 1 releases.”
“Body 2 pushes body 5.”
“Body 6 rotates.”
“Body 7 falls.”

Fig. 3. The explanation of an egg cracking device. (Labels of the form “body 3” are created by the system for each component; some have been removed in this figure for clarity.)

4.1 Inferences from device structure

ASSISTANCE is given a description of the device's structure (supplied by another system we have developed [1]) that specifies each of the objects in the figure and their connections.[1] As its first step ASSISTANCE does a degree of freedom analysis based on the interconnection information (e.g., anchors prevent both rotation and translation while pin joints allow rotation).

4.2 Generating the behavioral model from the description

The bulk of the work of ASSISTANCE lies in parsing the user's verbal description and sketched annotations, and providing a causal model of the device behavior. We walk through several inputs to illustrate this process in action, detailing the knowledge required to understand the description. The example illustrates ASSISTANCE's ability to infer motions of bodies, identify multiple descriptions of the same event, disambiguate deictic references, and infer causal links between motions.

"When body 3 moves up spring 1 releases" ASSISTANCE begins by breaking the utterance into its constituent clauses, which it then translates into events. A straightforward interpretation of the first clause ("body 3 moves up") generates a representation for the motion of body 3. The system then infers the motion of body 2 from the second clause ("spring 1 releases"), based on the observation that spring 1 is connected on the left end to an anchored body (body 1), hence in order for the spring to "release," body 2 must be moving. This is an example of an inference based on the physical structure of the device.

ASSISTANCE then infers a causal connection between these two motions because the two clauses are linked by a conditional statement ("When body 3 moves...") suggesting causality, in which the motion of the first clause is a precondition for the motion in the second. This is an example of using linguistic properties to infer a causal link between events.

---

[1] We use "objects" to mean any of the things in the sketch. We refer to objects such as springs, pulleys, pin joints, etc., as "functional components", or "components." We use the term "body" to refer to any other hunk of material in the sketch (e.g., everything other than the spring, pulley, pin joints, and arrows in Fig. 3).

The arrow at body 3 From the arrow at body 3, ASSISTANCE generates a second event representation for the motion of this component, describing its path. ASSISTANCE then links this representation with the representation generated by the utterance above, based on its ability to recognize that the two descriptions describe the same type of motion and refer to the same object.

"Body 2 pushes body 5", the arrow at body 2 ASSISTANCE interprets the "pushes" phrase as two motion events with the first (the motion of body 2) causing the second (the motion of body 5). The causal link is inferred by the fact that *pushing* is interpreted as the act of one object causing another object to move.

From the arrow at body 2 the system generates a second representation the motion of that body (the first resulted from "the spring releases" utterance). Recognizing that they refer to motions of the same object, the system marks the two representations as describing the same motion. These equivalence links between event representations are used later to merge two descriptions of the same event into a single, more detailed representation.

At this point ASSISTANCE's behavioral model represents the fact that the motion of body 3 causes the motion of body 2, which in turn causes the motion of body 5.

"Body 6 rotates," and "body 7 falls" From these utterances ASSISTANCE infers a causal link between the motions of body 6 and body 7. This is based on its model of pulleys, which is simply that if two things are attached to either end of a pulley, and both of them are known to be moving, then one of the motions may have caused the other. ASSISTANCE uses this piece of physical reasoning and the topology of the device to hypothesize a causal link between the two motion events.

The arrows at bodies 8, 9, and 10 ASSISTANCE infers that there is a motion event associated with each of these 3 arrows. It also infers that the motion of the egg causes the motions of the two levers, based on the observation that the path followed by the egg brings it into contact with the levers.

Finally, ASSISTANCE observes that the motions of the two levers are rotations because the bodies have a single, rotational degree of freedom.

4.3 How ASSISTANCE demonstrates understanding

ASSISTANCE now has both a structural and behavioral model of the device. The system can demonstrate its understanding of the device by describing the events that a component is involved in and the immediate causes and effects of those events. An example is presented in Fig. 4.

Designer:	What is body 2 involved in?
ASSISTANCE:	The motion of body 3 causes the motion of body 2 which causes the motion of body 5

Fig. 4. ASSISTANCE demonstrates its understanding.

The system also demonstrates understanding by adjusting the parameters of the springs in the structural model so that the simulation of the device is closer to the behavior described by the designer. Consider the spring in Fig. 3: As drawn, is it currently compressed, stretched, or at its neutral position? With the knowledge that body 2 is propelled by spring 1, ASSISTANCE is able to infer that it must be compressed, allowing the program to modify the model, which then will produce the correct behavior when the device is simulated.

5 Implementation

The process by which ASSISTANCE builds its models can be split into four main components:

- Process sketch and speech input
- Translate these inputs into events and causal links
- Construct the causal structure
- Demonstrate the system's understanding of the explanation

We describe the overall architecture of the system, then discuss each of these components in detail.

5.1 System architecture

The basic structure and information flow in the system is depicted in Fig. 5. The sketch recognition system, ASSIST recognizes the raw sketch data and produces a description of objects in the sketch and their interconnections. IBM's ViaVoice recognizes the acoustic data and produces a decorated parse tree of the utterances. The input to ASSISTANCE is thus descriptions of physical bodies, descriptions of arrows, and parsed textual phrases, rather than raw pixel and acoustic data. The information from ASSIST and ViaVoice is converted into propositional statements and used as the foundation for the reasoning performed by ASSISTANCE.

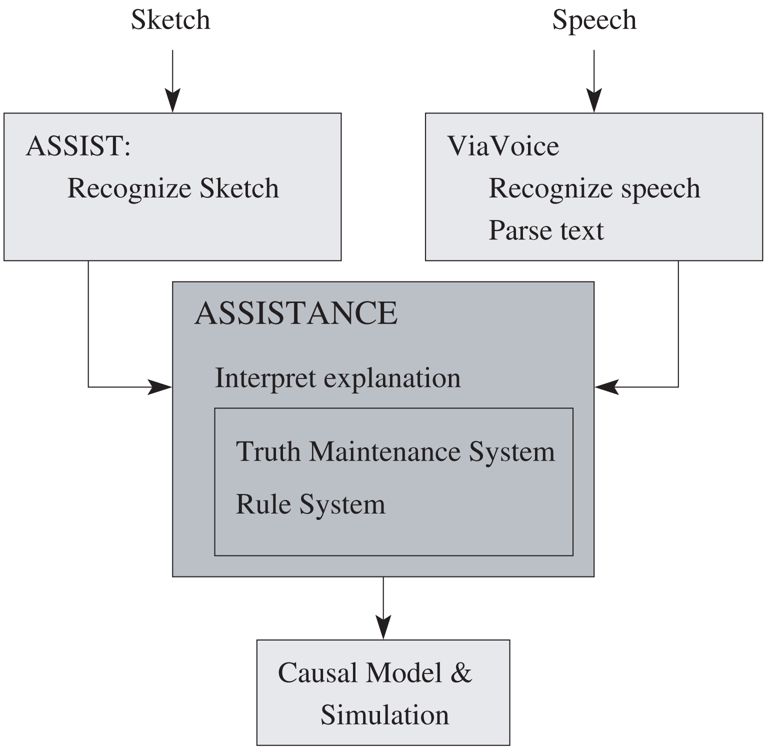


Fig. 5. The overall structure of the system.

ASSISTANCE is implemented with a forward-chaining rule-based system and truth maintenance system (TMS) taken from [7]. The rules represent the knowledge required to translate the parsed utterances and gestures into representations of events, and the knowledge required to infer causal relationships between events.

The role of the TMS is to maintain a record of inferences: When a rule fires, the TMS records the rule's preconditions as justifications for the inference. When the system later attempts to build a complete causal model of the device behavior, these records of inferences permit flexible and efficient exploration of alternative interpretations.

5.2 Process the inputs

There are three inputs to ASSISTANCE: the structural model generated by ASSIST, the utterances recognized by the speech recognition system, and the sketched arrows.

The Representation of the structural model The structure representation provided by ASSIST[1] contains shape and location information about every object in the sketch. For functional components (e.g., springs, pin joints, etc.) ASSIST also indicates which bodies they are attached to and whether they are attached to the fixed plane. All objects are assigned a unique English name (e.g. "body 1", "spring 1") so that they can be referred to unambiguously.

ASSISTANCE uses this model to perform a degree of freedom analysis on each body, determining from its connections (e.g., a pin joint) whether it can rotate, translate, or neither.

Speech recognition and processing Speech recognition is handled by IBM's ViaVoice software, which parses the utterances against a grammar containing phrases we found commonly used in an informal survey of several designers explanations of devices. The grammar abstracts from the surface level syntactic features to an intermediate syntactic representation that explicitly encodes grammatical relations such as subject and object. These intermediate representations are used by the rules described below to generate semantic representations of the utterances. This type of intermediate syntactic representation is similar to the approach taken in [15].

The grammar is written using the Java Speech Grammar Format, which provides a mechanism for annotating the grammar rules with tags. These tags decorate the parse tree generated by the speech recognition system with both the surface level syntactic features and the intermediate syntactic representations mentioned above.

The grammar handles several basic sentence types: motions (e.g. "the block moves"), conditionals (e.g. "If body 1 moves up then Spring 1 releases"), and propulsions (e.g. "Body 2 pushes body 5"). The system is also capable of handling deictic references to bodies, for example "This pushes Body 1." The interpretation of each of these is described in more detail below.

The grammar has intentionally been kept constrained, because our emphasis has not been on the language processing aspects of the system. Having a small grammar also helps the speech recognition system achieve a high level of accuracy. We are currently looking into the possibility of linking into a more powerful language processing system such as the START System [10].

Recognizing sketched gestures The sketched gestures currently handled by ASSISTANCE are arrows and pointing gestures. Both of these gesture types are recognized by ASSIST and converted into a symbolic representation that includes the object that they refer to; ASSISTANCE then reasons with the symbolic representations. For arrows, the referent is the object closest to the base of the arrow and for pointing gestures it is the object that is closest to the point indicated.

5.3 Translate inputs into events and causal links

In order to construct the causal structure of the device from the explanation, ASSISTANCE must first determine what events are mentioned in the explanation, then unify multiple representations of the same event. It then determines the causal relationships between pairs of events. The knowledge required to make these inferences is represented by rules that fit into several categories:

- Translate utterances into events
- Resolve deictic references
- Translate arrows into events
- Merge multiple representations of the same event
- Find causal connections between events

The rules are organized around knowledge of language patterns (the first two categories), knowledge of drawing conventions (the third category), and knowledge about physics and physical devices. By organizing the rules around such knowledge, and by writing them to be as general as possible within these categories, we achieve a degree of generality in the system's performance. As is common with rule-based systems, new rules must be added with care, but the task has to date been quite tractable.

Translate utterances into events There are currently three rules that translate utterances into motion events. These rules correspond to the three classes of verbs understood by the system: “moves,” “releases,” and “propels.” The rule for translating “propels” utterances, such as “Body 2 pushes body 3,” will illustrate how these rules work. The decorated parse tree for this sentence is shown in Fig. 6. The parse tree contains the structure of the sentence and syntactic tags that indicate the parts of speech and the roles played by each part of the parse (e.g. subject).

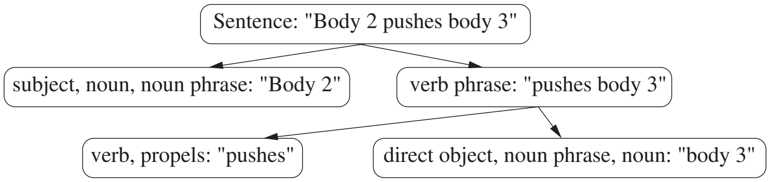


Fig. 6. Decorated parse tree for ”body 2 pushes body 3.”

To process this utterance the rule begins by identifying the sentence as a “propels” utterance, by the “propels” tag on the verb. Then it uses the structure of the parse tree to bind the “subject” to “body 2” and the “direct object” to “body 3.” Finally, it finds the physical objects corresponding to these bodies by matching the names to the bodies. The rule then asserts one motion event for the subject and one for the object.

The rule for “releases” utterances deals with springs, where “release” implies the motions of objects connected to either end of the spring. This rule is analogous to the one for “propels” except the bodies are those attached to the spring (which must be explicitly referenced by name or by a pointing gesture as described below).

The rule for interpreting “moves” utterances is analogous to the one for “propels” but does not have an object and only asserts one motion event.

Resolve deictic references In the current implementation, when objects are not referred to by name they must be accompanied by a pointing gesture. This allows the interpretation of phrases such as “when this moves up, body 1 pushes body 2.”

In addition to the pronouns “this” and “that” the system has a vocabulary of common component names like “block” and “ball” which the user can use to reference objects. However, since the system does not have representations for “balls” and “blocks” these references must also be disambiguated by pointing gestures. For example, the user can say “the block moves up” as long as she also points to the block in question.

The constraint that each reference has a corresponding pointing gesture means that the two can be matched in a straightforward manner, simply by keeping a list of references and pointing gestures and matching them in the order in which they occur.

While deictic references are common in explanations, our current requirement that the user disambiguate the referent is awkward and is one that we plan to eliminate in the future. There has been substantial work in both the literature on discourse theory and multi-modal interfaces [13] to indicate that this is possible. In particular the typical time delays between gestures and verbal utterances reported in [14] could be used to identify such multimodal references.

Translate arrows into events The translation of sketched arrows to motion events is straightforward because the recognition of the arrow includes the determination of the object it refers to: this is defined as the object at or near the tail of the arrow. A simple rule associates a motion event with this body and records the path depicted by the arrow.

Merge multiple representations of the same event We have found that in many explanations the motion of one body is described multiple times, often by multiple modalities. For example, in the egg cracker (see Fig. 3) the motion of body 2 is described three times: by the utterance “spring 1 releases”, by the utterance “body 2 pushes body 5,” and by the arrow. Initially each of these is represented individually.

To generate a unified causal structure, sets of equivalent events must be combined into a single, canonical event. This is done by unifying the properties of the individual events. In the example mentioned above, the representation generated from the arrow provides spatial information about the trajectory of the motion, while the verbal utterance indicates the causal connection between that motion and other motions.

Our current implementation assumes that each body can be involved in only a single motion event. This means that any two events that involve the same physical object are actually different descriptions of the same event[2]

The occurrence of overlapping descriptions from multiple modalities is well known (see [13] for example), however, our problem is slightly more complex than is typical: We not only have redundant descriptions due to multiple modalities, we also have redundant explanations within a modality, as, for example, the two utterances describing the motion of body 3. This is why we perform the merging of events based on the semantic interpretations instead of the input sources.

Find causal connections between events After identifying the motion events, the system attempts to find causal connections between them. There are currently two classes of causal links in our system: definite and plausible.

Definite Causal Links Definite causal links result from verbal utterances that unambiguously describe a causal relationship between two events. If, for example, the user says, "if this moves up spring 1 releases," we take that to be an unambiguous statement of a causal relationship between the two events. Definite causal links are also constructed from "propels" utterances, to relate the two motion events with a causal relationship. There are currently two rules for asserting definite causes, one for utterances of conditional statements and one for "propels" utterances.

Plausible Causal Links Plausible causal links arise from less explicit indications of causality. Currently these links are inferred from spatial information about the trajectories traced by bodies and from the motions of bodies connected by a rod or pulley system. There is one rule for each of these cases. As an example, spatial information is used in the egg-cracker description (Fig. 3) to infer that the motions of the two levers holding the egg are caused by the motion of the egg on its way into the pan. If the first body's trajectory brings it in contact with the second body, it is plausible that the second body's motion is caused by the first, but not guaranteed: the second motion may have commenced

---

[2] Future implementations will relax this restriction by adding reasoning to determine when an event description refers to a new event rather than another aspect of a previously mentioned event. The decision to keep the events separate could be based on evidence of conflicting properties (e.g. direction) or by examining the amount of time since the last reference to the event.

before the first. This rule makes use of the size of the bodies and the path traced by the arrow.

As as a second example, if two bodies are attached to a rope that passes through a pulley and they both move, it is plausible that one of the motions is the cause of the other. However, this is not definite because there may be slack in the rope.

5.4 Construct the causal structure

After finding all the events and the causal relationships between them, ASSISTANCE has two remaining tasks: (i) find the set of consistent causal structures, and (ii) choose the causal structure that is closest to the designer's description.

Find the set of consistent causal structures The constraints that must be satisfied in order for a causal ordering to be considered consistent are: (i) each event must have exactly one cause (but can have multiple effects), and (ii) causes precede effects.

The truth maintenance system and its constraint propagation capabilities enable fast and efficient exploration of different possible causal structures, allowing the system to find those consistent with the constraints. This exploration of alternative causes is a forward-looking depth-first search with backtracking over the set of possible causal orderings. The search is constrained by limiting it to those events with no definite causes and multiple plausible causes. It proceeds by trying all the plausible causes of each event until each has a cause. Any event that does not have a cause can be hypothesized to be caused by an exogenous force (a later step minimizes the number of hypothesized exogenous causes).

The truth maintenance system can identify and record sets of inconsistent assumptions, enabling search to be terminated along other branches of the tree that include the same set of assumptions that led to the contradiction. Although the search is exponential in the worst case, the branching factor is small and for our current problems efficiency has not been an issue. (We believe that there may be more efficient search strategies that take advantage of the fact that causes and effects are generally described together, and refer to physical components that are nearby spatially. This may allow us to partition the search space and reduce the depth of the search.)

Choose the causal structure that is closest to the designer's description Finally, the system must choose from all the consistent models the one that most closely matches the designer's description. Two heuristics are used to select the model: there should be a minimal number of events caused by exogenous forces, and the order of the events in the causal description should be as close as possible to the order in which they were described. This latter criterion is based on our empirical observation that people describe the behavior in the order in which it occurs.

6 Evaluation and future work

Criteria for evaluating a system such as ASSISTANCE include its usability and the range of inferences supported by its representations. We consider these by first comparing existing alternative methods for behavior explanation. We then evaluate the usability and expressiveness of ASSISTANCE and discuss the features that we believe are necessary for the growth of the system.

6.1 Existing alternatives

To date designers have had to choose between descriptions that were formal, constrained, and usable by an automated system, and those that were natural and unconstrained, but not easily automated.

On the formal end of the spectrum are CAD tools, which require the designer to describe the device with mathematical precision, using input media that are very different from the pencil and paper sketches used in the early design stages. Although some CAD systems claim to support sketching, they are still highly modal and force the designer to indicate what they are about to draw, instead of just drawing it. To date CAD systems have also supported the specification of behavior only through the adjustment of parameters, rather than via explicit descriptions of the intended behavior.

On the opposite end of the spectrum are written documents, person to person explanations, and verbatim recordings of explanations. The collection of this information imposes no constraints on the designer but also does not produce a representation usable by an automated reasoning system.

ASSISTANCE aims to combine the strengths of both of these approaches to create a system that gathers information from natural interactions and generate useful representations.

6.2 Usability issues

We have not yet performed a formal evaluation of ASSISTANCE's naturalness but can offer comments from our own experiences.

As we have demonstrated, ASSISTANCE is capable of interpreting the types of input media and language that designers use in the early stages of design. The explanations include many features commonly found in person-to-person explanations such as the use of natural language, sketching, simple deictic references, and the use of behavior-oriented explanations instead of parameter-oriented ones. The representations generated by ASSISTANCE are in a machine readable form suitable for use by other reasoning systems. By generating such representations ASSISTANCE is capitalizing on one of the primary advantages of CAD tools.

One area of the interface we hope to improve is its ability to provide feedback to the user about its current level of understanding. One way to do this would be to involve the computer in a dialog with the designer, in which the system asks for clarifications and asks questions about the roles played by different components. This would both provide the designer feedback about the system's current understanding of the explanation and offer some structure to the explanation which may provide additional constraints on the interpretation of the explanations. This inclusion of the computer as an active participant is an approach that is also advocated in [8] and fits into our overall goal of providing an interface that is as close as possible to person-to-person interactions.

A second improvement would be the extension of the reference disambiguation facilities. As mentioned earlier (Section 5.3) there has been previous work on this topic (e.g. [14, 13]) to provide guidance in expanding the system's current capabilities in this area.

A third improvement would be the extension of the natural language capabilities. The grammar of recognized utterances is currently too small to allow designers who have not previously used the system to describe a device easily. This difficulty is complicated by occasional errors in the speech recognition. Using a mature speech understanding system such as START [10] will alleviate some of these problems, by accounting for common language structures such as passive voice.

6.3 Expand reasoning abilities

Future work also needs to be done to expand the range of devices that the system can understand. In particular the limitation that each body can be involved in only a single state transition precludes many common devices. We hypothesize that this does not pose any conceptual level adjustments to the architecture but it will involve reengineering some of the internal representations of the reasoning system. The ability to manipulate the components of the device directly (in addition to just describing them) could help this problem from a usability perspective by visually displaying the current state of the device. Without this feature the designer must visualize the new positions of components after each event occurs. As the chain of events involving the same component grows this will become a larger issue.

7 Related work

While a variety of work has explored the understanding of descriptions in individual modalities, and some multi-modal systems deal with direct manipulation tasks, relatively little work has attempted to interpret the sorts of multi-modal descriptions handled by ASSISTANCE.

7.1 Related description understanding systems

Borchardt [4], for example, parsed natural language descriptions of device behavior and from this reconstructed the causal relationships described in the text. His insight was to focus on the changes that occurred in the state of the device instead of the states themselves. This closely parallels our goal to focus on descriptions of behavior instead of descriptions of structure. The primary difference in our work derives from having sketched input; having explicit spatial models simplifies many descriptions. This changes the focus of the descriptions and allows them to be conveyed more naturally.

Understanding device behavior was also an element in [16]. In that system the designer specified structure by indicating some elements of the topology of the device and described behavior with a state transition diagram. From these representations, his system was able to understand the operation of the device and suggest alternative designs with the same qualitative behavior as the original. To engineers, a state transition diagram is one form of natural explanation, and as such that work took one step in the direction we have pursued.

Another approach to the behavior understanding problem is to infer the device's behavior by observation, without a separate behavioral description. This was the approach taken in [12] to infer the behavior of a device from static diagrams. Similarly, [5] and [6] interpret the behavior of devices from images and sequences of images of a device in action. These approaches are important because they could provide an initial guess at the behavior which can be augmented and repaired by explanations provided through a system such as ours.

7.2 Related multimodal systems

There has been a great deal of work done on the design and theory of pen and speech based multimodal interfaces (see [13] for an overview). This work has focused on improving recognition accuracy by combining multiple input modalities. It has also identified general properties of multimodal human computer interactions that can guide their design. [14].

Another body of work in the field of multi-modal interfaces has focused on recording human interactions in meetings[3]. The goal of that work is to generate annotated multimedia transcripts of meetings. The transcripts include the text of what was said, who said it, and links between the transcript and video sequences.

ASSISTANCE fits between these two bodies of work in its emphasis on being a silent observer but also understanding the content the user is conveying instead of just recording it in a structured manner. Another system which takes this approach is Rasa described in [11].

8 Contributions

ASSISTANCE demonstrates a new kind of interface for describing mechanism behavior. Rather than proposing better templates, buttons, or menus, ASSISTANCE adopts the interface that designers use everyday to communicate with their colleagues. Armed with knowledge about sketching, natural language, and mechanical devices, ASSISTANCE brings the computer into the designer's world. During conceptual design, designers talk about behaviors and not the parameters that lead to them, hence ASSISTANCE focuses on understanding behavioral explanations rather than providing ways of specifying parameters.

9 Acknowledgments

This work was supported in part by the Ford-MIT Collaboration, and in part by the MIT Oxygen Partnership.

References

1. C. Alvarado and R. Davis. Resolving ambiguities to create a natural computer-based sketching environment. In *IJCAI-01*, 2001.
2. V. Baya and L. Leifer. Understanding information management in conceptual design. In *Analysing Design Activity*, pages 151–168. John Wiley, 1996.
3. Michael Bett, Ralph Gross, Hua Yu, Xiaojin Zhu, Yue Pan, Jie Yang, and Alex Waibel. Multimodal meeting tracker. In *Proceedings of RIAO2000*, April 2000.
4. G. Borchardt. Causal reconstruction. Tech. Report AIM-1403, MIT, February 1993.
5. M. Brand. Physics-based visual understanding. *CVIU*, 65(2):192–205, February 1997.
6. T. Dar, L. Joskowicz, and E. Rivlin. Understanding mechanical motion. *Artificial Intelligence*, 112:147–179, 1999.
7. K. Forbus and J. de Kleer. *Building Problem Solvers*. MIT Press, Cambridge, MA, 1993.
8. Kenneth Forbus, Ronald Ferguson, and Jeffery Usher. Towards a computational model of sketching. In *Proceedings of the 2001 International Conference on Intelligent User Interfaces*, pages 77–83, January 2001.
9. M. Hearst. Sketching intelligent systems. *IEEE Intelligent Systems*, pages 10–18, May/June 1998.
10. B. Katz. From sentence processing to information access on the world wide web. In *AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, 1997.
11. D. McGee, P. Cohen, and L. Wu. Something from nothing: Augmenting a paper-based work practice via multimodal interaction. In *Proceedings of Designing Augmented Reality Environments*, pages 71–80. ACM Press, April 2000.
12. N. Narayanan, M. Suwa, and H. Motoda. *Hypothesizing Behavior from Device Diagrams*, chapter 15, pages 501–534. MIT Press, Cambridge, MA, 1995.
13. S. Oviatt, P. Cohen, L. Wu, J. Vergo, L. Duncan, , B. Suhm, J. Bers, T. Holzman, T. Winograd, J. Landay, J. Larson, and D. Ferro. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions. In *Human Computer Interaction*, volume 15, pages 263–322. Addison-Wesley, 2000.
14. Sharon Oviatt, Antonella DeAngeli, and Karen Kuhn. Integration and synchronization of input modes during multimodal human-computer interaction. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Speech, Haptic, & Multimodal Input*, pages 415–422, 1997.
15. M. Palmer, R. Passonneau, C. Weir, and T. Finin. The KERNEL text understanding system. *Artificial Intelligence*, 63(1-2):17–68, October 1993.

16. T. Stahovich, R. Davis, and H. Shrobe. Generating multiple new designs from a sketch. In *Proc. 13th AAAI*, pages 1022–1030, Menlo Park, August 1996. MIT Press.
17. D. Ullman, S. Wood, and D. Craig. The importance of drawing in the mechanical design process. *Computers and Graphics*, 14(2):263–274, 1990.

Knowledge Access

ACL 39th Annual Meeting and 10th Conference of the European Chapter: Proceedings of the Workshop Human Language Technology and Knowledge Management, Toulouse, France, July 2001.

Gathering Knowledge for a Question Answering System from Heterogeneous Information Sources

Boris Katz, Jimmy Lin and Sue Felshin

MIT Artificial Intelligence Laboratory
{`boris, jimmylin, sfelshin`}`@ai.mit.edu`

Abstract. Although vast amounts of information are available electronically today, no effective information access mechanism exists to provide humans with convenient information access. A general, open-domain question answering system is a solution to this problem. We propose an architecture for a collaborative question answering system that contains four primary components: an annotations system for storing knowledge, a ternary expression representation of language, a transformational rule system for handling some complexities of language, and a collaborative mechanism by which ordinary users can contribute new knowledge by teaching the system new information. We have developed a initial prototype, called Webnotator, with which to test these ideas.

1 Introduction

A tremendous amount of heterogenous information exists in electronic format (the most prominent example being the World Wide Web), but the potential of this large body of knowledge remains unrealized due to the lack of an effective information access method. Because natural language is the most convenient and most intuitive method of accessing this information, people should be able to access information using a system capable of understanding and answering natural language questions—in short, a system that combines human-level understanding with the infallible memory of a computer.

Natural language processing has had its successes and failures over the past decades; while the successes are significant, computers will not soon be able to *fully* process and understand language. In addition to the traditional difficulties associated with syntactic analysis, there remains many other problems to be solved, e.g., semantic interpretation, ambiguity resolution, discourse modeling, inferencing, common sense, etc. Furthermore, not all information on the Web is textual—some is sound, pictures, video, etc. While natural language processing is advanced enough to understand typical interactive *questions* about knowledge (interactive questions are typically fairly simple in structure), it cannot understand the *knowledge* itself. For the time being, therefore, the only way for computers to access their own knowledge is for humans to tell the computers what the knowledge means in a language that the computers can understand—but still in a language that humans can produce. A good way to accomplish this is with the use of *natural language annotations*, sentences which are simple enough for a computer to analyze, yet which are in natural human language. Once knowledge is so annotated, and indexed in a knowledge repository, a question answering system can retrieve it.

The START (SynTactic Analysis using Reversible Transformations) Natural Language System [6, 7] is an example of a question answering system that uses natural language annotations. START is a natural language question answering system that has been available to users on the World Wide Web[1] since December, 1993. During this time, it has engaged in millions of exchanges with hundreds of thousands of people all over the world, supplying users with knowledge regarding geography, weather, movies, corporations, and many many other areas. Despite the success of START in serving real users, its domain of expertise is relatively small and expanding its knowledge base is a time-consuming task that requires trained individuals.

We believe that the popularity of the Web may offer a solution to this knowledge acquisition problem by providing collaborative mechanisms on a scale that has not existed before. We can potentially leverage millions of users on the World Wide Web to construct and annotate a knowledge base for question answering. In fact, we had proposed a distributed mechanism for gathering knowledge from the World Wide Web in 1997 [7], but only recently have we attempted to implement this idea.

An advantage of natural language annotations is that it paves a smooth path of transition as natural language processing technology

---

[1] `http://www.ai.mit.edu/projects/infolab`

improves. As natural language analysis techniques advance, the annotations may become more and more complex. Eventually, a textual information segment could be its own annotation; someday, through other technologies such as speech and image recognition, etc., annotations could even be automatically constructed for non-textual information.

A further advantage is that natural language annotations can be processed via techniques that only partially understand them—via IR engines, or less-than-ideal natural language systems—yet they retain their more complex content and can be reanalyzed at a later date by more sophisticated systems.

2 Overview

We propose a collaborative question answering architecture composed of the four following components:

1. **Natural Language Annotation** is a technique of describing the content of information segments in machine parsable natural language sentences and phrases.
2. **Ternary Expressions** are subject-relation-object triples that are expressive enough to represent natural language, and also amenable to rapid, large-scale indexing.
3. **Transformational Rules** handle the problem of *linguistic variation* (the phenomenon in which sentences with different surface structures share the same semantic content) by explicitly equating representational structures (derived from different surface forms) that have approximately the same meaning.
4. **Collaborative Knowledge Gathering** is a technique by which the World Wide Web may be viewed not only as a knowledge resource, but also a human resource. The knowledge base of a question answering system could be constructed by enlisting the help of millions of ordinary users all over the Web.

3 Annotations

Natural language annotations are machine-parsable sentences or phrases that describe the content of various information segments. They describe the questions that a particular segment of information is capable of answering. For example, the following paragraph about polar bears:

> Most polar bears live along the northern coasts of Canada, Greenland, and Russia, and on islands of the Arctic Ocean...

may be annotated with one or more of the following:

> Polar bears live in the Arctic.
> Where do polar bears live?
> habitat of polar bears

A question answering system would parse these annotations and store the parsed structures with pointers back to the original information segment that they described. To answer a question, the user query would be compared against the annotations stored in the knowledge base. Because this match occurs at the level of ternary expressions, structural relations and transformation (to be discussed in Section 5) can equate queries and annotations even if their surface forms were different. Furthermore, linguistically sophisticated machinery such as synonymy/hyponymy, ontologies, can be brought to bear on the matching process. If a match were found, the segment corresponding to the annotation would be returned to the user as the answer.

The annotation mechanism we have outlined serves as a good basis for constructing a question answering system because annotating information segments with natural language is simple and intuitive. The only requirement is that annotations be machine parsable, and thus the sophistication of annotations depends on the parser itself. As natural language understanding technology improves, we can use more and more sophisticated annotations.

In addition, annotations can be written to describe any type of information, e.g., text, images, sound clips, videos, and even multimedia. This allows integration of heterogenous information sources into a single framework.

Due to the vast size of the World Wide Web, trying to catalog all knowledge on the World Wide Web is a daunting task. Instead, focusing on meta-knowledge is a more promising approach to building a knowledge base that spans more than a tiny fraction of the Web. Consider that reference librarians at large libraries obviously don't know all the knowledge stored in the reference books, but they are nevertheless helpful in finding information, precisely because they have a lot of *knowledge about the knowledge.* Natural language annotations can assist in creating a smart "reference librarian" for the World Wide Web.

4 Representing natural language

A good representational structure for natural language is ternary expressions.[2] They may be intuitively viewed as subject-relation-object

---

[2] See [6, 7] for details about such representation in START.

triples, and can express most types of syntactic relations between various entities within a sentence. We believe that the expressiveness of ternary relations is adequate for capturing the information need of users and the meaning of annotations. For example, "What is the population of Zimbabwe?" would be represented as two ternary expressions:

```
[what is population]
[population of Zimbabwe]
```

Ternary expressions can capture many relationships between entities within a sentence. Such a representational structure is better than a keyword-based scheme which equates a document's keyword statistics with its semantic content. Consider the following sets of sentences/phrases that have similar word content, but (dramatically) different meanings:[3]

(1) The bird ate the young snake.
(1′) The snake ate the young bird.
(2) The meaning of life
(2′) A meaningful life
(3) The bank of the river
(3′) The bank near the river

Ternary expressions abstract away the linear order of words in a sentence into a structure that is closer to meaning, and therefore a relations-based information access system will produce much more precise results.

We have conducted some initial information retrieval experiments comparing a keyword-based approach with one that performs matching based on relations[4]. Using Minipar [12], we parsed the entire contents of the Worldbook Encyclopedia and extracted salient relations from it (e.g., subject-verb-object, possessives, prepositional phrase, etc.) We found that precision for relations-based retrieval was much higher than for keyword-based retrieval. In one test, retrieval based on relations returned the database's three correct entries:

Question: What do frogs eat?
Answer:
(R1) Adult frogs eat mainly insects and other small animals, including earthworms, minnows, and spiders.
(R4) One group of South American frogs feeds mainly on other frogs.

---

[3] Examples taken from [13]

[4] to be published

(R6) Frogs eat many other animals, including spiders, flies, and worms.

compared to 33 results containing the keywords *frog* and *eat* which were returned by the keyword-based system—the additional results all answer a different question ("What eats frogs?") or otherwise coincidentally contain those two terms.

Question: What do frogs eat?
Answer:
...
(R7) Adult frogs eat mainly insects and other small animals, including earthworms, minnows, and spiders.
(R8) Bowfins eat mainly other fish, frogs, and crayfish.
(R9) Most cobras eat many kinds of animals, such as frogs, fishes, birds, and various small mammals.
(R10) One group of South American frogs feeds mainly on other frogs.
(R11) Cranes eat a variety of foods, including frogs, fishes, birds, and various small mammals.
(R12) Frogs eat many other animals, including spiders, flies, and worms.
(R13) ...

Another advantage of ternary expressions is that it becomes easier to write explicit transformational rules that encode specific linguistic variations. These rules are capable of equating structures derived from different sentences with the same meaning (to be discussed in detail later).

In addition to being adequately expressive for our purposes, ternary expressions are also highly amenable to rapid large-scale indexing and retrieval. This is an important quality because a large question answering system could potentially contain answers to millions of questions. Thus, compactness of representation and efficiency of retrieval become an important consideration. Ternary expressions may be indexed and retrieved efficiently because they may be viewed using a relational model of data and manipulated using relational databases.

5 Handling linguistic variation

Linguistic variation is the phenomenon in which the same meaning can be expressed in a variety of different ways. Consider these questions, which ask for exactly the same item of information:

(4) What is the capital of Taiwan?
(5) What's the capital city of Taiwan?
(6) What is Taiwan's capital?

Linguistic variations can occur at all levels of language; the examples above demonstrate lexical, morphological and syntactic variations. Linguistic variations may sometimes be quite complicated, as in the following example, which demonstrates verb argument alternation.[5]

(7) Whose declaration of guilt shocked the country?
(8) Who shocked the country with his declaration of guilt?

Transformational rules provide a mechanism to explicitly equate alternate realizations of the same meaning at the level of ternary expressions.

As an example, Figure 1 shows a sample transformational rule for (7) and (8).[6] Thus, through application of this rule, question (7) can be equated with question (8).

[n_1 `shock` n_2]		[n_3 `shock` n_2]
[`shock with` n_3]	$\leftrightarrow$	
[n_3 `related-to` n_1]		[n_3 `related-to` n_1]
where $n \in$ *Nouns*		where $n \in$ *Nouns*

Fig. 1. Sample Transformational Rule

Transformational rules may be generalized by associating arbitrary conditions with them; e.g., *verb* $\in$ shock, surprise, excite ...

A general observation about English verbs is that they divide into "classes," where verbs in the same class undergo the same alternations. For example, the verbs 'shock', 'surprise', 'excite', etc., participate in the alternation shown in Sentence (7) and (8) not by coincidence, but because they share certain semantic qualities. Although the transformational rule required to handle this alternation is very specific (in that it applies to a very specific pattern of ternary expression structure), the rule can nevertheless be generalized over all verbs in the same class by

---

[5] Beth Levin [9] offers an excellent treatment on English verb classes and verb argument alternations.

[6] This rule is bidirectional in the sense that each side of the rule implies the other side. The rule is actually used in only one direction, so that we canonicalize the representation.

associating with the rule conditions that must be met for the rule to fire, i.e., *verb* $\in$ *emotional-reaction-verbs*; see Figure 2.

$$\begin{array}{|ccc|} \lbrack n_1\ v_1\ n_2 \rbrack & & \lbrack n_3\ v_1\ n_2 \rbrack \\ \lbrack v_1\ \texttt{with}\ n_3 \rbrack & \leftrightarrow & \\ \lbrack n_3\ \texttt{related-to}\ n_1 \rbrack & & \lbrack n_3\ \texttt{related-to}\ n_1 \rbrack \\ \multicolumn{3}{|l|}{\text{where } n \in \textit{Nouns} \text{ and } v \in \textit{emotional-reaction-verbs}} \end{array}$$

Fig. 2. Sample Transformational Rule

Note that transformational rules can also encode semantic knowledge and even elements of common sense. For example, a rule can be written that equates a *selling* action with a *buying* action (with verb arguments in different positions). Or as another example, rules can even encode implicatures, e.g., A murdered B implies that B is dead.

Transformational rules can apply at the syntactic, semantic, or even pragmatic levels, and offer a convenient, powerful, and expressive framework for handling linguistic variations.

In order for a question answering system to be successful and have adequate linguistic coverage, it must have a large number of these rules. A lexicon which classified verbs by argument alternation patterns would be a good start, but this is another resource lacking in the world today. Rules generally may be quite complex, and it would be difficult to gather such knowledge from average Web users with little linguistic background. Requesting that users describe segments with multiple annotations (each representing a different phrasing of the description), might serve as a preliminary solution to the linguistic variation problem. Another possible solution will involve learning transformational rules from a corpus. The difficulty in creating transformational rules is a serious problem and unless and until this problem is solved, an NL-based QA system would have to be restricted to a limited domain where a small number of experts could provide enough transformational rule coverage, or would require a large commitment of resources to attain sufficient coverage.

6 Collaboration on the web

A critical component of a successful natural language question answering system is the knowledge base itself. Although the annotation mechanism simplifies the task of building a knowledge base, the accumulation of knowledge is nevertheless a time consuming and labor intensive task. However, due to the simplicity of natural language annotations (i.e., describing knowledge in everyday English), ordinary users with no technical skills may contribute to a knowledge base. Thus, by providing a general framework in which people on the World Wide Web can enter additional knowledge, we can engage millions of potential users all over the world to collaboratively construct a question answering system. We can distribute the effort of building a knowledge base across many ordinary users by allowing them to teach the system new knowledge.

The idea of using the Internet as a tool for collaboration across geographically distributed regions is not a new idea. The Open Source movement first demonstrated the effectiveness and sustainability of programming computer systems in a distributed manner. Made possible in part by the World Wide Web, the Open Source movement promotes software development by nurturing a community of individual contributors working on freely distributed source code. Under this development model, software reliability and quality is ensured through independent peer review by a large number of programmers. Successful Open Source projects include *Linux*, a popular Unix-like operating system; *Apache*, the most popular Web server in the World; *SendMail*, an utility on virtually every Unix machine; and *dmoz*, the Open Directory Project, whose goal is to produce the most comprehensive directory of the Web by relying on volunteer editors.[7]

Another example of Web-based collaboration is the Open Mind Initiative [17, 18], which is a recent effort to organize ordinary users on the World Wide Web (*netizens*) to assist in developing intelligent software. Based on the observation that many tasks such as speech recognition and character recognition require vast quantities of training data, the initiative attempts to provide a collaborate framework for collecting data from the World Wide Web. The three primary contributors within such a framework are domain experts, who provide fundamental algorithms, tool/infrastructure developers, who develop the framework for capturing data, and non-expert *netizens*, who supply the raw training data.

Open Mind Commonsense[8] is an attempt at constructing a large common sense database by collecting assertions from users all over the Web.[9]

---

[7] `http://www.dmoz.org`

[8] `http://openmind.media.mit.edu`

[9] A non-collaborative approach to building a common sense knowledge base is taken by Lenat whose Cyc project [8] is an attempt to build a common

Other projects have demonstrated the viability of Web-enabled collaborative problem-solving by harnessing the computational power of idle processors connected to the Web.[10] The SETI (Search for Extraterrestrial Intelligence) Institute was founded after NASA canceled its High Resolution Microwave Survey project. The institute organizes thousands of individuals who donate their idle processor cycles to search small segments of radio telescope logs for signs of extraterrestrial intelligence.[11] Other similar projects that organize the usage of idle processor time on personal computers include the Internet Mersenne Prime Search,[12] and the RC5 Challenge.[13]

Recent technical, social, and economic developments have made the abovementioned models of collaboration possible. Furthermore, numerous successful projects have already demonstrated the effectiveness of these collaborative models. Thus, it is time to capitalize on these emerging trends to create the first collaborative question answering system on the World Wide Web.

Even with the components such as those described above, there still remains a major hurdle in jumpstarting the construction of a collaborative question answering system. We are faced with a classic chicken-and-egg problem: in order to attract users to contribute knowledge, the system must serve a real information need (i.e., actually provide users with answers). However, in order to serve user information needs, the system needs knowledge, which must be contributed by users.

In the initial stages of building a question answering system, the knowledge base will be too sparse to be useful. Furthermore, the system may be very brittle, and might not retrieve the correct information segment, even if it did exist within the knowledge base (e.g., due to a missing transformational rule).

It may be possible to address this dilemma with an incremental approach. The system can first be restricted to a very limited domain (e.g., "animals" or "geography"). Users' expectations will be carefully managed so that they realize the system is highly experimental and has a very limited range of knowledge. In effect, the users will be populating a domain-specific knowledge base. Over time, the system will be able to answer more and more questions in that domain, and hence begin to offer interesting answers to real users. After this, a critical mass will

---

sense knowledge base through a small team of dedicated and highly trained specialists.

[10] `http://www.distributed.org`

[11] `http://setiathome.ssl.berkeley.edu`

[12] `http://www.mersenne.org`

[13] `http://www.distributed.org/rc5/`

form so that users are not only teaching the system new knowledge, but also receiving high quality answers to their questions. At that point, a decision can be made to increase the domain coverage of the system.

In order to initialize this process, we can bootstrap off the curiosity and altruism of individual users. As an example, the Openmind Common Sense project has accumulated over 280 thousand items of information by over six thousand users based on a data collection model that does not supply the user with any useful service. The dream of building "smart" systems has always been a fascination in our culture (e.g., HAL from 2001: A Space Odyssey); we believe that this will serve to attract first-time users.

7 Evolving the system

While the collaborative information gathering task proceeds, we are then faced with the problem of maintaining the system and ensuring that it will provide users with useful information. Two immediate issues arise: quality control and linguistic variation.

How can we insure the quality of the contributed material? In general, any system that solicits information from the World Wide Web faces a problem of quality control and moderation. Although most Web users are well-meaning, a small fraction of Web users may have malicious intentions. Therefore, some filtering mechanisms must be implemented to exclude inappropriate content (e.g., pornography or commercial advertisement) from being inserted into the knowledge base. More troublesome is the possibility of well-meant but incorrect information which is probably more common and definitely harder to detect.

How can we handle linguistic variation? There are often different ways of asking the same question; the annotation of a particular segment might not match the user query, and hence the correct answer may not be returned as a result. Transformational rules may be a solution to the problem, but writing and compiling these rules remain a difficult problem.

We propose a variety of solutions for the maintenance of a collaborative question answering system, depending on the level of human intervention and supervision.

At one end of the spectrum, an unsupervised approach to quality control can be implemented through a distributed system of moderation with different *trust levels*. The scheme essentially calls for self-management of the knowledge repository by the users themselves (i.e., the users with high trust levels). Different trust levels will allow users various levels of access to the knowledge base, e.g., the ability to modify or delete information segments and their associated annotations or to modify other users' trust levels. To initiate the process, only a small group of core editors is required.

In such an unsupervised system, the problem of linguistic variation could be addressed by prompting users to give multiple annotations, each describing the information content of a particular segment in a different way. With a sufficiently large user base, wide coverage might still be achieved in the absence of broad-coverage transformational rules.

At the other end of the spectrum, a large organization may commit significant amounts of resources to maintaining a supervised collaborative knowledge base. For example, an organization may be willing to commit resources to preserve its organizational memory in the form of an "intelligent FAQ" supported by natural language annotations. Computers can be effectively utilized to augment the memory of an organization [2], and have been successfully deployed in real-world environments with relative success [1].

If an organization were willing to commit significant resources to a collaborative knowledge repository, then transformational rules can be written by experts with linguistic background. Such experts could constantly review the annotations entered by ordinary users and formulate transformational rules to capture generalizations.

Supervised use of natural language annotation falls short of the grandiose goal of accessing the entire World Wide Web, but is the practical and useful way to apply NL annotation until the transformational rule problem can be solved for unlimited domains.

8 Initial prototype

Webnotator is a prototype test-bed to evaluate the practicality of NL-based annotation and retrieval through Web-based collaboration. It provides efficient facilities for retrieving answers already stored within the knowledge base and a scalable framework for ordinary users to contribute knowledge.

The system analyzes natural language annotations to produce ternary expressions by postprocessing the results of Minipar [10, 11], a fast and robust functional dependency parser that is freely available for non-commercial purposes. The quality of the representational structures depends ultimately on the quality of whatever parser Webnotator is made to access. In the current implementation, ternary expressions are not embedded, elements of ternary expressions are not indexed, and coreference is not detected. Words are stemmed to their root form and morphological information is discarded. The system also implements a version of transformational rules described above as a simple forward-chaining rule-based system.

Using a relational database, Webnotator implements a knowledge base that stores ternary expressions derived from annotations and their associated information segments. Ternary expressions fit neatly into a relational model of data, and thus manipulation of the knowledge (including answering queries and inserting new knowledge) can be formulated as SQL queries. This vastly simplifies development efforts while maintaining robustness and performance.

Webnotator provides an interface through which users may teach the system new knowledge by supplying new information segments and adding new annotations. Essentially, the user enters, in a CGI form, an information segment and annotations that describe the knowledge. Since the segment of information can contain any valid HTML, images, tables, and even multimedia content may be included. Alternatively, the user may simply provide a URL to annotate, and Webnotator will automatically create a link to the URL in its knowledge base.

Currently, Webnotator is a prototype that has been released to a small community of developers and testers within the MIT Artificial Intelligence Laboratory. We plan on releasing the system to the general public in the near future. By collecting knowledge from the general public and by varying the representations and transformations applied by Webnotator, it should be possible to discover which features are most important for a natural-language-based annotation system and whether the state of the art is indeed sufficiently advanced to make such a system practical and effective.

9 Related work

A variety of research has been conducted on better information access methods on the World Wide Web (e.g., the "Semantic Web" [4]). However, most of these approaches have concentrated on methods of annotating existing web pages with metadata such as XML/RDF (Resource Description Framework) [16], extensions to HTML [14, 5, 16], specialized descriptions [19], or even conceptual graphs [15].

The common thread among previous work is the embedding of metadata directly into Web documents, which are then gathered via crawling or spidering. This approach only works if the target community of the system is well-defined; adoption of various metadata techniques are presently limited, and thus it would be pointless to crawl the entire web to search for metadata. A model in which distributed metadata are gathered by a spider will not work with a constantly changing community that is ill-defined. In principle, there is no reason why our natural language annotations cannot be embedded into Web documents also; the issue is strictly a practical concern.

Another common theme in previous work is the organization of knowledge in accordance with some pre-established ontology. This presents several challenges for building a general system for gathering knowledge. Ontologies are often either too specific to be of general use (e.g., RiboWeb's ontology for ribosome data [3]), or too weak to provide much structure (e.g., Yahoo). Since the ontology is static and must be agreed upon prior to any knowledge base development, it may be too constricting and too inconvenient for the expression of new or unanticipated concepts. Although systems do allow for arbitrary extension of the ontology [5, 16], such extensions defeat the purpose of a structure-imposing ontology. Our proposed alternative to a ontological hierarchy is to take advantage of the expressiveness of natural language, and use linguistic devices to relate concepts. The combination of lexical resources (e.g., synonyms and meronyms in WordNet) and transformational rules provide a natural, extensible way to relate and structure different concepts.

A compelling argument for natural language annotations is their expressiveness and compactness. Martin and Eklund [15] argue against an XML-based system of metadata because XML was primarily intended to be machine readable, not human readable. In their paper, they started with an English phrase, and then proceeded to demonstrate the encoding of that sentence in various formalisms. A constraint graph encoding was simpler than a KIF (Knowledge Interchange Format) encoding, which was in turn shorter than a RDF format. Of course, this begs the question: why not just annotate the document with the original English phrase? Current NLP technology can handle a large variety of English sentences and phrases, which may serve as the annotations directly. Such is system is not only simpler, more intuitive, but also more compact.

10 Conclusion

Recent social, technical, and economic developments have made possible a new paradigm of software development and problem solving through loosely-organized collaboration of individuals on the World Wide Web. Many successful precedents have already proven the viability of this approach. By leveraging this trend with existing annotation and natural language technology, we can provide a flexible framework for a question answering system that grows and "evolves" as each user contributes to the knowledge base, with only minimal outside supervision. Testing will reveal whether such a system can help users realize some of the untapped potential of the World Wide Web and other sources of digital information as a vast repository of human knowledge.

References

1. Mark S. Ackerman. Augmenting organizational memory: A field study of answer garden. *ACM Transactions on Information Systems*, 16(3):203–224, July 1998.
2. Thomas Allen. *Managing the Flow of Technology.* MIT Press, 1977.
3. R. Altmann, M. Bada, X. Chai, M. W. Carillo, R. Chen, and N. Abernethy. RiboWeb: An ontology-based system for collaborative molecular biology. *IEEE Intelligent Systems*, 14(5):68–76, 1999.
4. T. Berners-Lee. *Weaving the Web.* Harper, New York, 1999.
5. Jeff Heflin, James Hendler, and Sean Luke. SHOE: A knowledge representation language for internet applications. Technical Report CS-TR-4078, Institute of Advanced Computer Studies, University of Maryland, College Park, 1999.
6. Boris Katz. Using English for indexing and retrieving. In P.H. Winston and S.A. Shellard, editors, *Artificial Intelligence at MIT: Expanding Frontiers*, volume 1. MIT Press, 1990.
7. Boris Katz. Annotating the World Wide Web using natural language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97)*, 1997.
8. Doug Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.
9. Beth Levin. *English Verb Classes and Alternations: A Preliminary Investigation.* University of Chicago Press, 1993.
10. Dekang Lin. Principled-based parsing without overgeneration. In *Proceedings of the 31th Annual Meeting of the Association for Computational Linguistics (ACL'93)*, 1993.
11. Dekang Lin. PRINCIPAR—An efficient, broad-coverage, principle-based parser. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING '94)*, 1994.
12. Dekang Lin. Minipar—a minimalist parser. In *Maryland Linguistics Colloquium*, University of Maryland, College Park, March 12, 1999.
13. Edward Loper. Applying semantic relation extraction to information retrieval. Master's thesis, Massachusetts Institute of Technology, 2000.
14. S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In *Proceedings of the First International Conference on Autonomous Agents*, 1997.
15. Philippe Martin and Peter Eklund. Embedding knowledge in web documents. In *Proceedings of the Eighth International World Wide Web Conference*, 1999.

16. S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. Semantic community web portals. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.
17. David G. Stork. Character and document research in the open mind initiative. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, 1999.
18. David G. Stork. Open data collection for training intelligent software in the open mind initiative. In *Proceedings of the Engineering Intelligent Systems Symposium (EIS '2000)*, 2000.
19. M. Grotschel W. Dalitz and J. Lugger. Information services for mathematics and the internet. In A. Sydow, editor, *Proceedings of the 15th IMACS World Congress on Scientific Computation: Modelling and Applied Mathematics*. Wissenschaft und Technik Verlag, 1997.

Proceedings of the WWW 10th
International Conference, Hong Kong,
May 2001.

Annotea: An Open RDF Infrastructure for Shared Web Annotations

José Kahan,[1] Marja-Riitta Koivunen,[2] Eric Prud'Hommeaux[2]
and Ralph R. Swick[2]

[1] W3C INRIA Rhone-Alpes
[2] W3C MIT Laboratory for Computer Science
{`kahan, marja, eric, swick`}`@w3.org`

Abstract. Annotea is a Web-based shared annotation system based on a general-purpose open RDF infrastructure, where annotations are modeled as a class of *metadata*. Annotations are viewed as statements made by an author about a Web document. Annotations are external to the documents and can be stored in one or more *annotation servers*. One of the goals of this project has been to re-use as much existing W3C technology as possible. We have reached it mostly by combining RDF with XPointer, XLink, and HTTP. We have also implemented an instance of our system using the Amaya editor/browser and a generic RDF database, accessible through an Apache HTTP server. In this implementation, the merging of annotations with documents takes place within the client. The paper presents the overall design of Annotea and describes some of the issues we have faced and how we have solved them.

1 Introduction

One of the basic milestones in the road to a Semantic Web [22] is the association of metadata to content. Metadata allows the Web to describe properties about some given content, even if the medium of this content does not directly provide the necessary means to do so. For example, a metadata schema for digital photos [15] allows the Web to describe, among other properties, the camera model used to take a photo, shutter speed, date, and location. An interesting side effect, is that a same piece of metadata can be used not only for describing content, but also to organize and classify it, thus setting up other properties we had not thought about at first. For example, we can use it to search for photos of a given location taken at a given time.

A first step towards building a Semantic Web is to have the infrastructure needed to handle and associate metadata with content. In order to reach this goal, we have been developing Annotea, a shared Web annotation system. In its simplest form, a Web annotation [25] can be seen as a remark about a document identified by a URI, made by the author of the document or by a third party, with or without author knowledge. In a shared Web annotation system, annotations are stored in specialized servers. Annotations are shared in that everyone having access to an annotation server should be able to consult the annotations associated with a given document and add their own annotations.

From a general viewpoint, annotations can be considered as metadata: they associate remarks to existing documents. We chose to use the annotation scenario to drive our initial metadata infrastructure development as it is a relatively simple metadata application and it would allow us to concentrate on the general details of the infrastructure without getting lost with the more specific details of the application. The most important goal of this project has been to use as much existing W3C specifications as possible. This paper describes how we have reached this goal by combining RDF with XPointer, XLink, and HTTP

The paper concentrates on describing the Annotea RDF infrastructure and its implementation in Amaya. Section 2 gives the overall design of Annotea. Section 3 describes the client implementation. Section 4 briefly discusses differences with related work by others. Section 5 concludes the paper and presents our perspectives for future work on Annotea.

2 Design

In this section we describe the architecture of the Annotea system and the RDF annotation schema. We start with a discussion of the requirements that motivate some of the aspects of our design.

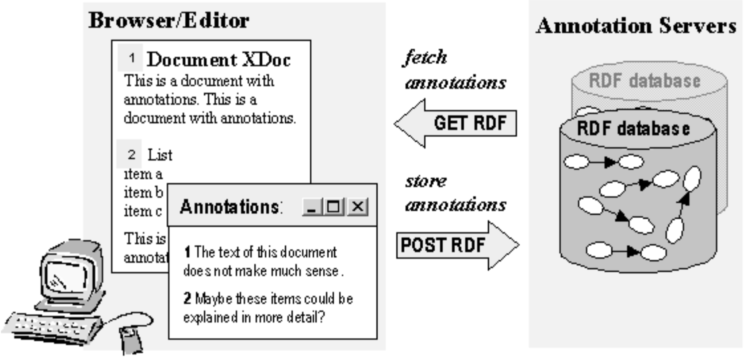
2.1 Requirements

Since the early design of Annotea, we decided to build an infrastructure that was based on generic RDF, with annotations being one possible instantiation of the infrastructure. This choice has allowed us to concentrate more on the infrastructure than on the application itself. We now list the principal requirements that have shaped Annotea (given in no particular order):

- Open technologies. Many of the existing annotation systems are based in proprietary schemes or protocols. This makes it hard to extend them. Annotea is built on top of open standards to simplify the interoperability of Annotea with other annotation systems and to maximize the extensibility of the data this system can carry.
- Annotated documents are well-formed, structured documents. Many Web annotation systems allow users to annotate any kind of resource that has a URI. To simplify our design, we decided to limit annotated resources to those that have a structure, that is, any HTML or XML-based document, including other annotations.
- Annotations are first class Web resources. As any other Web resource, each annotation should be associated with a URI.
- Annotations are typed. At the same time that an annotation can be seen as metadata related to an annotated document, annotations themselves can have distinct properties. The type of an annotation is metadata about the annotation itself. It allows users to classify the annotations as they are creating them (for example, saying this annotation is a comment or an erratum about a given document).
- Annotation types can be defined by users. Different users have different views and needs. Annotea should make it possible for any user group to define their own annotation types.
- Annotation properties must be described with an RDF schema [20, 5].
- Annotations are stored in generic RDF databases. Rather than making a specialized annotation server, we decided to view the servers as generic RDF databases. This is important as it will allow users, in the general Semantic Web picture, to reuse the information stored in such databases without having to change them.
- No assumptions on User Interface. Annotea describes how metadata can be associated with documents and how to query the RDF databases. It does not specify how a user agent must present the metadata to the user. We do predict, though, that some user interface consistency is needed. However, with our approach it is easy to provide additional views of the metadata.
- Local (private) and remote (shared) annotations. Annotations can be stored either locally in the user's host computer or in an annotation server. We assume that local annotations are private and remote ones shared. An annotation server is responsible for controlling access to the annotations that it stores.
- Multiple annotation servers. Having a centralized server may present both scalability [16] and privacy problems. User

groups must be able to easily set up an annotation server and define who can consult it. Thus, in an Annotea instantiation, there may be a number of annotation servers. We do not attempt to solve the general scalability problem at this time.

2.2 Annotea and its operation



This is a document with annotations. This is a document with annotations.

This is annotat

1 The text of this document does not make much sense.

2 Maybe these items could be explained in more detail?

Fig. 1. The basic architecture of Annotea.

In Annotea, annotations are described with a dedicated RDF schema and are stored in annotation servers (Fig. 1). The annotation server stores the annotations in an RDF database. Users can query a server to either retrieve an existing annotation, post a new annotation, modify an annotation, or delete an annotation. All communication between a client and an annotation server uses the standard HTTP methods [21].

The annotations that we handle are collections of various statements about a document. They may be comments, typographical corrections, hypothesis or ratings, but there is always an author that makes a statement about the document or some part of it at a certain time. This is illustrated in Figure 2, where an author makes a statement about a document named XDoc. An annotation is represented as a set of metadata and an annotation body.

The metadata of an annotation is modeled according to an RDF schema and gives information such as the date of creation of the annotation, name of the author, the annotation type (e.g., comment, query, correction...) the URI of the annotated document, and an XPointer that specifies what part of the document was annotated. The metadata also includes a URI to the body of the annotation, which we assume to be an XHTML document. The annotation metadata does not say how the annotations must be presented to the user. This choice is left open to the developer of the client. Section 2.3 describes the annotation schema further in detail.

Annotations are stored in generic RDF databases, which are accessible through an HTTP server. The following scenario explains the interaction between these components when a user creates a new annotation document. For simplicity, we will suppose that annotations are displayed by highlighting the annotated text in the document.

- The user browses a document.
- The user selects some text on the document and tells its browser that he wants to annotate this text.
- The browser pops up a new window, where the user can type the text of his annotation and choose the type of the annotation.
- The user then publishes the annotation to a given annotation server. To do this, the browser generates an RDF description of the annotation that includes the metadata and the body and sends it to the server, using the HTTP POST method. The server assigns a URI to the annotation and to the body and replies with an RDF statement that includes these URIs.
- If the user further modifies the annotation, he will publish it directly to the URI that was assigned.

Note that the first time that a user publishes an annotation, this annotation does not have any URI. It is the server that assigns the URI. When the user requests the URI from the server later, the server will reply with the annotation metadata.

We now describe the scenario where the user browses an annotated document. We suppose this user has previously configured his browser with the list of annotation servers that he wants to query.

- The user browses a document
- The browser queries each of the annotation servers, requesting via an HTTP GET method the annotation metadata that is associated with the document's URI.
- Each server replies with an RDF list of the annotation metadata. If the server is not storing any related annotations, it replies with an HTTP 404 Not Found message.
- For each list of annotations that it receives, the browser parses the metadata of each annotation, resolves the XPointer of the annotation and, if successful, highlights the annotated text.

- If the user clicks on the highlighted text, the browser will use an HTTP GET method to fetch the body of the annotation from the URI specified in the metadata.
- Finally, the browser will open up a window showing the metadata and the body.

In the above scenario, we divided the downloading of annotations into two stages. First, the browser downloads the metadata of an annotation. Next, and only if the user requests it explicitly, the browser downloads the body of the annotation. The motivation for this choice is to reduce the amount of data that is being sent back to the browser. In a heavily annotated document, sending the complete annotations will consume resources and the user may not actually be interested in seeing the body of all the annotations.

Note that once that an annotation is published to a server, it becomes a shared annotation. That is, any user with the correct access rights may retrieve the annotations from the server. For the moment, we expect that the HTTP server will enforce the access control to the annotations, using the standard HTTP authentication mechanisms.

It is also possible to store annotations locally in the host computer of the user, provided that the client simulates the reply to the first query of the server. Our Amaya prototype, that we describe later in Section 3, implements such a feature.

The next section presents the Annotation RDF schema. Appendix A contains a more thorough presentation of the Annotea protocols.

2.3 RDF schema for annotations

The most important feature of an annotation is that it supports the evolving needs of the collaborating groups. For instance, an annotation system for classifying new technologies will need to expand their annotation types to classify specific characteristics of the technologies they are reviewing. Another working group may start with a set of annotation types and then modify this set according to the evolution of their work. Annotea users may wish to define new types of annotations as they use the system more. The group may also add relationships to other objects that are specific to the group's work. RDF provides support for these needs, e.g., by allowing the expression of new relationships, by allowing new annotation types, and by supporting the transformations from one annotation type to another.

RDF provides a simple yet very flexible framework for describing properties of any Web resources. In its most simple level, RDF provides

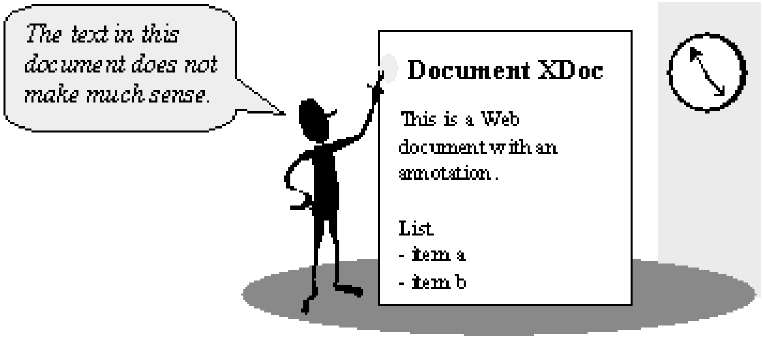


Fig. 2. A basic annotation model with an author making a statement about a document.

(resource, property, value) triples (Fig. 3). A single triple is a statement that indicates that a *resource* has a given *property* with a given *value.* The *resource* can be any Web resource identified by a URI. The *value* may be a literal string or may be the URI of another Web resource. Literal strings may contain XML markup. By design, RDF permits separate communities to develop independent metadata vocabularies and then freely mix statements using those vocabularies in a single database of triples. In RDF, the property names themselves are Web resources, and applications can use the URIs of those properties to make other statements about the properties themselves, such as their meaning and their relationship to other properties.

Fig. 3. RDF triple model.

The type of an annotation is defined by the user or the group by declaring additional annotation classes. These classes are a part of the RDF model and may be described on the Web in an RDF Schema [5]. The general annotation super class is called *Annotation* (more precisely, its name is `http://www.w3.org/2000/10/annotation-ns#Annotation`

– a URI about which an application can expect to ask the Web for more information) and we have defined several sample subclasses based on it (Fig. 4). These subclasses are defined in a separate RDF Schema whose namespace is `http://www.w3.org/2000/10/annotationTypes#`. Likewise, other user groups can easily create new subclasses. We can also easily add new properties to the annotation classes, for instance, we could add a property that defines an annotation set. This property can then be queried with general RDF mechanisms and also presented as text. However, to do more advanced presentations with the basic RDF mechanisms we would need to develop presentation schemas for RDF.

Annotation A super class describing the common features of annotations.

Advice A subclass of *Annotation* representing advice to the reader.

Change A subclass of *Annotation* describing *annotations* that document or propose a change to the source document.

Comment A subclass of *Annotation* describing annotations that are comments.

Example A subclass of *Annotation* representing examples.

Explanation A subclass of *Annotation* representing explanations of content.

Question A subclass of *Annotation* representing questions about the content.

SeeAlso A subclass of *Annotation* representing a reference to another resource.

Fig. 4. Basic annotation classes.

Annotations are user made statements that consist of these main parts: the body of the annotation, which contains the textual or graphical content of the annotation, the link to the annotated document with a location within the document, an identification of the person making the annotation and additional metadata related to the annotation. By using RDF we can take advantage of other work on Web metadata vocabularies wherever possible. Specifically, we use the Dublin Core [9] element set to describe some of the properties of annotations. The annotation properties are illustrated in the RDF model presented in Figure 5 and the corresponding schema definitions for properties are defined in Figure 6.

The RDF schema that defines the annotation properties consists of the property name and the natural language explanation. The type is one of the basic classes in Figure 4 or some other type of annotation defined elsewhere. The *annotates* property stores the link to the anno-

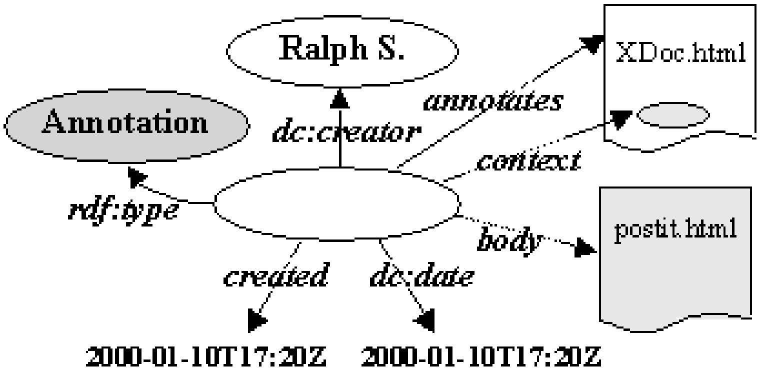


Fig. 5. The RDF model of an annotation.

tated document, *body* is a link to the content of the annotation, and *dc:creator* to the author making the annotation.

The *context* defines where exactly inside the document the annotation is attached. We use XPointer [7] for defining positions within XML documents. This works well for static (unchanging) documents, but with documents that go through revision, such as working group drafts, we may end up with orphan annotations or annotations pointing to wrong places. To prevent unnecessary loss of pointers we can search for the nearest ID to a parent of the object use it as the starting point for the XPointer path. Fortunately, many documents usually have IDs at least at their main levels. Pointing to finer details after the ID can be done by other XPointer means, such as using text matching.

The additional annotation metadata includes date for the creation and last modified time, and *related* for adding relationships to other objects. Other metadata can be added to the annotation when the working group needs that. For instance, the working group will probably add their own properties directly and not specialize the *related* property.

Sample annotations utilizing this schema definition are presented in Appendix A while discussing the protocols.

3 Annotations in Amaya

One of the goals of Annotea is to help us gain experience on building an RDF infrastructure. Since the beginning of the project, we have been implementing both a client and a server prototype. For the client,

rdf:type	An indication of the creator's intention when making an annotation; the value should be of rdf:type *Annotation* or any of its subclasses.
annotates	The relation between an annotation resource and the resource to which the annotation applies.
body	The content of the annotation.
context	Context within the resource named in annotates to which the annotation most directly applies. Eventually this will be an XPointer. It may include a location range too. First locations will points to XML IDs.
dc:creator	The creator of the annotation.
created	The date and time on which the annotation was created.
dc:date	The date and time on which the annotation was last modified.
related	A relation between an annotation and a (collection of) resource(s) that augment the resource that is the *body* of the annotation. This may point to related issues, discussion threads, etc.

Fig. 6. The basic annotation properties.

we have been using Amaya, W3C's testbed editor browser. For the server, we have been using Apache, a MYSQL database running on top of it and some Perl scripts. The rest of this section describes the implementation choices we have made in Amaya 4.0.

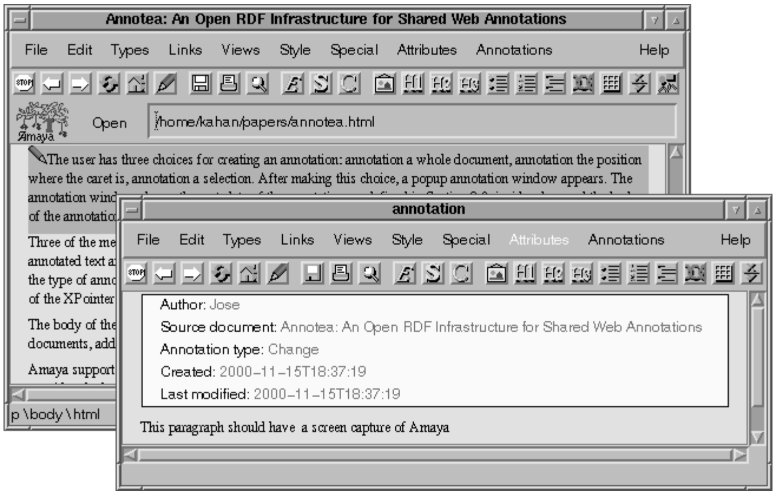
Amaya [1] is a full-featured web browser and editor developed by W3C for experimenting and validating web specifications at an early stage of their development. Amaya supports CSS, MathML, XHTML, HTML, and also provides a basic implementation of XLink and XPointer. Libwww [17] is linked to Amaya and provides HTTP/1.1 support and an RDF parser. Amaya can also show different views of a document. In particular, we have a Formatted view, which shows the interpreted document, and a Links view, which gives a list of all the links in the document.

Our prototype implementation is able to interpret the complete Annotation RDF schema and supports all of the Annotea protocols as described in Appendix A. It is also possible to specify additional annotation types (subclasses) as an RDF schema that can be can be downloaded at runtime. The namespaces for these additional types are specified to Amaya in a local configuration file that is read at startup. Amaya will use the namespace name to try to retrieve an RDF schema from the Web or the schema content can be cached in a local file and specified with the same startup configuration. The prototype is not yet able to recognize the need to download schemas dynamically from the information given in annotations metadata.

We will now describe the most important features of our implementation: creating an annotation, browsing annotations, and filtering annotations.

3.1 Creating an annotation

The user has three choices for creating an annotation: annotate a whole document, annotate the position where the caret is, annotate the current selection. After making this choice, a popup annotation window appears. The annotation window shows the metadata of the annotation, as defined in Section 2.3, inside a box and the body of the annotation. Figure 7 shows a screen capture of Amaya when creating an annotation on a selection.



The user has three choices for creating an annotation: annotation a whole document, annotation the position where the caret is, annotation a selection. After making this choice, a popup annotation window appears. The annotation wind

of the annotatio

Three of the me
annotated text a
the type of anno
of the XPointer

The body of the
documents, add

Amaya support

Author: Jose
Source document: Annotea: An Open RDF Infrastructure for Shared Web Annotations
Annotation type: Change
Created: 2000–11–15T18:37:19
Last modified: 2000–11–15T18:37:19

This paragraph should have a screen capture of Amaya

Fig. 7. Annotating a paragraph with Amaya.

Three of the metadata items are active. If the user clicks on the *Source document* field, Amaya will scroll to the annotated text and highlight it if it is a selection. Clicking on the *Annotation type* field allows the user to change the type of annotation. Finally, each time that the user saves the annotation Amaya updates the value of the *Last modified* field. Note that we do not show the value of the XPointer (context), but rather use it to select the source document highlighting.

The body of the annotation can be edited as any other XHTML document. Users can cut and paste fragments from other documents, add links to other documents, and so on.

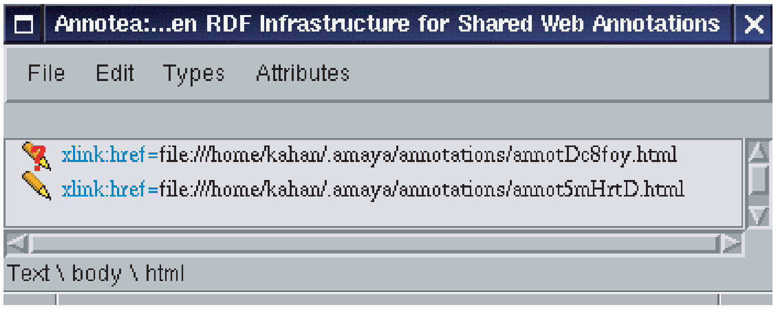
Amaya support both local (private) and remote (shared) annotations. When a user creates an annotation, it is considered a local one and will be stored in the user's Amaya directory. When the user decides to post it to an annotation server, the local annotation will be deleted and subsequent saves will be sent to the server. Both local and remote annotations are represented with the same schema format. The only difference is that for local ones, we emulate the annotation server's query response by using an index file that associates URIs with annotation metadata.

3.2 Browsing annotations

By means of a setup menu, the user can specify the URIs of the annotation servers he wants to query, as well as the local annotation repository. The user can also say if he wants annotations to download automatically or only on-demand. In order to avoid hampering performance, we separated the downloading process in two steps. Once a document is downloaded, the annotations metadata is downloaded asynchronously, just like images, and merged into the document. The body of an annotation is only downloaded when the user opens an annotation. The motivation for this choice is that metadata may be relatively smaller than the body of an annotation. Moreover, if the user does not open all the annotations, we save time by not downloading the body.

For showing annotations, we defined an active XML element, that we will call A-element, that has an XLink pointing to the body of the annotation and a special icon (currently, a pencil). This is similar to the X element that was used in the Annotated XML specification [3], with the difference that in Amaya, it is an active element. When the user clicks once on the A-element, Amaya highlights the target of the annotation. Clicking on it twice will open the annotation window and show both the metadata and the body of the annotation. The A-element is visible in both the Formatted Document and Links views and it is ignored when saving or printing an annotated document. Clicking on the A-element on any view has the same effect.

In the Formatted view, we position the A-element to the location to which the XPointer of the annotation resolves. We made an exception for MathML documents, as it would be disturbing to add it anywhere in the expression. Instead, we place it as the the beginning of the Math expression. Clicking on the A-element will highlight the target of the annotation, even if this target is not close to the A-Element.



```
xlink:href=file:///home/kahan/.amaya/annotations/annotDc8foy.html
xlink:href=file:///home/kahan/.amaya/annotations/annot5mHrtD.html
```

Fig. 8. The Links View showing an orphan annotation and a normal one.

If an annotated document is edited, some of its annotations may become orphan. That is, the XPointer will not resolve anymore to a point in the document. In this case, Amaya will warn the user and make the orphan annotation visible from the Links view. Figure 8 shows this view in a document that has an orphan and a valid annotation. The user may then open the orphan annotation and reposition its XPointer or delete it.

3.3 Filtering annotations

For a heavily annotated document, seeing the A-element icon can make reading the document bothersome. To avoid this problem, we defined a local filter that allows the user to hide the annotations according to one of three criterion: by author name, by annotation type, and by annotation server. It is also possible to hide all the annotations in the Formatted view. Using this menu, the user can hide all but the annotations that interest him. This filter menu does not have any effect on the Links view.

As an alternative to hiding annotations, the user can also temporarily disable some annotation servers using the configuration menu. We also have an experimental customized query feature, where the user can describe his own query, using a language we have named "Algae". The Algae language is derived from Algernon [4]. This customized query interface makes it possible to start filtering the annotations on the server side, for example, by only requesting those done in the past week by a given author and belonging to a given annotation type. Appendix B[1] gives a brief description of Algae.

4 Related work

This section discusses some previous annotation approaches. We concentrate on document-centered approaches where users are browsing documents and examining annotations related to them. There are also discussion-centered approaches to annotations, such as HyperNews [12], where users browse discussion messages and threads and follow a link to a document that these messages annotate.

Web annotations first appeared in version 1.2 of Mosaic [18, 19], almost ten years ago, and many other web annotation aware tools or servers have seen the light since then, such as CritLink [24] and ThirdVoice [23]. [10, 11] list other existing annotation technologies. Due to the lack of existing annotation standards, most of these proposals are proprietary or closed.

The two main categories to Web annotation systems are *proxy-based* approaches and *browser-based* approaches. In a *proxy-based* approach, annotations are stored and merged with a Web document by a proxy server. The browser user agent only sees the result of the merge, typically with some semantic content removed. In a *browser-based* approach the browser is enhanced (either by an external application or by a plugin) to merge the document and the annotation data just prior to presenting the content to the user. The annotation data is stored in the proxy or a separate annotation server. It is also possible to store annotations locally or provide site specific annotations, but these are less interesting to us because of their limitations.

The CritLink [24] annotation tool uses the proxy approach where a Web page and its annotations are served through a different URI address than the original page. This approach works with any existing browser. However, the user must use different addresses for the document depending on which annotation proxy server is used. This is a limitation when a user wants to use more than one annotation server. The proxy approach also inherently restricts the types of content that can be annotated and the presentation styles that can be used for the annotations. Typically, presentation of the annotations is limited to the presentation styles available through HTML. Finally, as the browser does not have any knowledge about annotations, it makes it harder to

---

[1] Available only in the HTML version of this paper.

filter the annotations locally, without having to send a new request to the proxy server.

ThirdVoice [23] uses plugins to enhance web browsers so that they understand annotations. The users can annotate the page or some text on the page with discussions on selected topics. The discussions can be closed to a group of participants or open to anybody. Unfortunately, users cannot host their own servers.

IMarkup [13] is an Internet Explorer annotation tool that has an interesting user interface. Users have a wide variety of palettes for annotation markers and can even circle parts of the text with something akin to a real marker. Annotations can be placed anywhere on the browser's document window, without taking into account the markup of the document itself. All the annotations are local. A menu entry allows to mail annotations to other users and to import them. The format used for describing annotations is proprietary and too related to the browser's API, making their use with other tools practically impossible.

An interesting possibility for presenting the annotations on a Web page is to use internal DOM [14] events without actually changing the mark-up of the page. Yawas [6] is an annotation tool that uses this approach. It codes the annotations into an extended URI format and uses local files similar to bookmark files to store and retrieve the annotations. A modified browser can transform the URI format into DOM events. The local annotation files can be sent to other users only by mail or copied by other means. There is no provision for having active links or filtering options. This kind of approach is limited by the API provided by the browser.

XLink [8], an XML linking technology currently under development in W3C, has some built in features in the mark-up for creating annotations. For instance, it is possible to store XLink arcs in an external document that can be loaded with another document. The content defined by the end locator of an XLink arc may be embedded to the location in a document defined by a starting locator of the arc. Using XLink provides the means to easily present the annotations in predefined ways in any browser implementing XLink. However, the metadata properties that can be expressed with XLink are limited.

5 Conclusions and future plans

Being able to associate metadata with Web resources is an important milestone for building a Semantic Web. Annotea provides a simple infrastructure for associating annotations with Web documents, without having to modify these documents. The principal contributions of Annotea are as follows:

- Annotations are metadata. Annotea is not designed as a specific annotation system, but rather as a general application of a generic RDF infrastructure. This allows a variety of applications to reuse the information that is stored in an Annotea system with other RDF tools that are not necessarily specific to annotations.
- Open infrastructure. Annotea is built on top of W3C specifications. We use an RDF schema for describing the properties of annotations, XPointer for associating annotations to documents, and HTTP for the client/server interactions.
- Use of RDF databases. By storing annotations inside RDF databases, it is possible to make customized queries and limit the amount of data returned by the servers.
- Extensible RDF schema. Our annotation RDF schema defines general properties about annotations. Users can extend it by defining their own annotation types or by adding other annotation properties.
- Client-less. Annotea defines an infrastructure for associating metadata with documents and for storing and retrieving this metadata. In principle, it is possible to build an Annotea client on top of any browser that handles DOM, XPointer, XLink and RDF.

In November 2000, we made the first public release of the Annotea prototypes. The client is included as a built-in feature of the Amaya 4.0 release. We have also set up a public annotation server [2]. All the source code is freely available too. The public server is not intended to be a permanent service, but rather one that will be purged periodically. Its goal is to let people experiment with annotations and motivate them to set up their own servers.

Our wish list for future work on Annotea includes:

- Shared bookmarks. Shared bookmarks are quite similar to annotations. The annotation schema provides a set of fixed annotation types. The user is expected to classify his annotation by selecting one of these types. With shared bookmarks, the user should be able to define his own types on-the-fly, for example, by highlighting keywords on the annotated document. These types can then be used to automatically classify the bookmarks.
- User interface. Currently, a user can see annotations either as pencil icons next to the fragment that was annotated or in a special

Links view. We would like to experiment with other ways for displaying annotations. For example, by embedding the body of the annotations in the annotated document.

- Author metadata. The Annotation schema defines the author as a string. We plan to expand the schema so that the author is defined by another RDF schema and use this property in the Annotation schema. It will then be easy to search for the metadata of the author and, for example, substitute the pencil icon with the photo of the author.
- Robust XPointers. Currently, we are able to detect orphan annotations. However, our XPointer expressions are very simple. If a user edits an annotated document, in some cases, the XPointer of an annotation may point to the wrong place and thus become a misleading annotation. We have made some provisions for this case (use of the ID attribute), but this is not enough. A better XPointer expression would be one that is more tolerant of document changes, but robust enough to prevent misleading annotations.
- Discussion threads. When a user wants to reply to an annotation, s/he can either modify the body of the annotation or make a new annotation. This can become cumbersome as we would need to browse each annotation in order to follow the discussion. We can improve this situation by adding new RDF properties for distinguishing such discussions and by showing all the replies to a given annotation in a specialized view.

6 Acknowledgements

The authors would like to thank Art Barstow, Tim Berners-Lee, Dan Brickley, Daniel LaLiberte, and Charles McCathie-Nevile for their useful feedback and suggestions concerning Annotea. Irène Vatton and Vincent Quint from the Amaya team have given irreplaceable help to us. Eric Miller gave us very enthusiastic encouragement to pursue this application as a testbed for RDF infrastructure.

References

1. Amaya Home Page. `http://www.w3.org/Amaya/`.
2. Annotest (Annotea Test Server) Home Page. `http://annotest.w3.org/`.
3. T. Bray. Using XML to Build the Annotated XML Specification, Sep. 1998. `http://www,xml.com/xml/pub/98/09/exexegesis-0.html`.

4. J. Crawford. *Access-Limited Logic: A Language for Knowledge Representation.* UT Artificial Intelligence TR AI90-141, Department of Computer Sciences, University of Texas at Austin, Austin, Texas., Oct. 1990.
5. D. Brickley and R.V. Guha (eds.). Resource Description Framework (RDF) Schema Specification 1.0. CR, W3C, Mar. 2000. `http://www.w3.org/TR/2000/CR-rdf-schema-20000327`.
6. L. Denoue and L. Vignollet. An annotation tool for Web browsers and its applications to information retrieval. In *Proceedings of RIAO200*, Apr. 2000. `http://www.univ-savoie.fr/labos/syscom/Laurent.Denoue/riao2000.doc`.
7. S. DeRose, R. Daniel Jr., and E. Maler (eds.). XML Pointer Language (XPointer). WD, W3C, Dec. 1999. `http://www.w3.org/TR/1999/WD-xptr-19991206`.
8. S. DeRose, E. Maler, D. Orchard, and B. Trafford (eds.). XML Linking Language (XLink). WD, W3C, Feb. 2000. `http://www.w3.org/TR/2000/WD-xlink-20000221`.
9. Dublin Core Metadata Element Set, Version 1.1: Reference Description. Technical report, Dublin Core Metadata Initiative, Jul. 1999. `http://purl.org/DC/documents/rec-dces-19990702.htm`.
10. J. Garfunkel. Web Annotation Technologies, 1999. `http://ps.pageseeder.com/ps/ps/papers/annot/jongar/jongar.pshtml`.
11. R. M. Heck, S. M. Luebke, and C. H. Obermark. A Survey of Web Annotation Systems, 1999. `http://www.math.grin.edu/\~luebke/Research/Summer1999/survey\_paper.htm%l`.
12. HyperNews Home Page. `http://www.hypernews.org/`.
13. IMarkup Home Page. `http://www.imarkup.com`.
14. L. Wood et al. (eds.). Document Object Model (DOM) Level 2 Specification Ver. 1.0. CR, W3C, Mar. 2000. `http://www.w3.org/TR/2000/CR-DOM-Level-2-20000307`.
15. Y. Lafon and B. Bos. Describing and Retrieving Photos Using RDF and HTTP. Note, W3C, Sep. 2000. `http://www.w3.org/TR/photo-rdf/`.
16. D. LaLiberte and A. Braverman. A Protocol for Scalable Group and Public Annotations, 1996. `http://www.hypernews.org/\~liberte/www/scalable-annotations.html`.
17. libwww Home Page. `http://www.w3.org/Library/`.
18. *NCSA Mosaic Documentation: Group Annotations in NCSA Mosaic*, 1993. `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/group-annotations.htm%l`.
19. *Mosaic Users's Guide:Annotations*, 1998. `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/help-on-annotate-win.%html`.
20. O. Lassila and R. R. Swick (eds.). Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, W3C, Feb. 1999. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222`.
21. R. Fielding et al. (eds.). Hypertext Transfer Protocol – HTTP/1.1. RFC RFC2616, IETF, Jun. 1999. `http://www.ietf.org/rfc/rfc2616.txt`.
22. Semantic Web Development. `http://www.w3.org/2000/01/sw/`.

23. ThirdVoice Home Page. `http://www.thirdvoice.com/`.
24. K.-P. Yee. CritLink: Better Hyperlinks for the WWW. Submitted to Hypertext '98, Apr. 1998. `http://crit.org/http://crit.org/\~ping/ht98.html`.
25. R. Zohar. Web Annotation - an Overview, Feb. 1999. `http://www-ee.technion.ac.il/\~ronz/annotation/`.

A Annotea protocols

We distinguish five types of client-server interactions in Annotea:

- **Posting** a new annotation: the client publishes a new annotation
- **Querying** the annotation server: the client sends a query to the server and gets backs the annotation
- **Downloading** the body of an annotation
- **Updating** an annotation: the client modifies an annotation and publishes these modifications
- **Deleting** an annotation: the client deletes an annotation from the server

For all of these cases, we use the standard HTTP protocol methods. We use HTTP POST for uploading a new annotation to a server, HTTP PUT to update an annotation, HTTP GET to query and download an annotation, and HTTP DELETE to delete an annotation. We will now describe each of these operations in detail.

We use the standard HTTP POST protocol for storing a new annotation to the annotation server and HTTP GET protocol for fetching the annotations and returning the result to the client. POST provides the necessary interface for the server to construct a URI for the new annotation and return that URI to the client. When the client has the URI for a previously created annotation, it can (with the proper permissions) use HTTP PUT to modify the annotation. In all the examples, we use the Apache shorthand CGI convention, using the string `annotations` to refer to the actual CGI script. This makes it easier to refer to an existing annotation.

A.1 Posting a new annotation

To create a new annotation, the client posts some RDF describing the annotation to a selected annotation server. Both the annotation and its body are specified as anonymous RDF resources in the POST message. The server is responsible for allocating the URIs for them. If the body already exists, as will happen if the annotation body is another document that the user wants to use as an annotation, the URI of that existing document can be specified in the RDF when the annotation is posted.

In Figure 9 we illustrate a request to create a simple annotation using an existing document as the body of the annotation. Note that the resource `http://www.example.com/mycomment.html` is presumed to exist independently of this annotation.

```
POST /annotations HTTP/1.1
Host: www.example.org
Content-Type: application/xml
Content-Length: 636

<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:a="http://www.w3.org/2000/10/annotation-ns#"
       xmlns:d="http://purl.org/dc/elements/1.1/">
 <r:Description>
  <r:type resource="http://www.w3.org/2000/10/
annotation-ns#Annotation"/>
  <r:type resource="http://www.w3.org/2000/10/
annotationType#Comment"/>
  <a:annotates r:resource="http://example.com/some/page.html"/>
  <a:context>#xpointer(id("Main")/p[2])</a:context>
  <d:creator>Ralph Swick</d:creator>
  <a:created>1999-10-14T12:10Z</a:created>
  <d:date>1999-10-14T12:10Z</d:date>
  <a:body r:resource="http://www.example.com/mycomment.html"/>
 </r:Description>
</r:RDF>
```

Fig. 9. Creating an annotation with POST, using an existing document as the body.

A design issue we encountered is that we wanted to be able to use XML for describing the body of an annotation, and at the same time we wanted to be able to publish the complete annotation in a single HTTP transaction. In order to use XML in the body, the correct architectural approach is to store the body as a separate resource with its own content type. We therefore designed a simple packaging protocol that permits both the client and server to specify embedded HTTP message bodies. To do this, we declare an RDF namespace for describing certain HTTP headers and we specify those HTTP headers as normal RDF properties, as shown in Figure 10.

In Figure 10, we show the metadata that specifies an annotation of the page whose URI is `http://example.com/some/page.html`. The creator of this annotation is identified as `Ralph Swick`. The text of the annotation body is `This is an important concept`.

As specified by the RDF model, the data we pass to the server in the POST is a set of statements describing properties of the new (and unnamed) annotation resource that we would like the server to create. In response to the POST (Fig. 11), a new annotation is created and the server assigns URIs. Now the server has created the URIs for the anonymous resources and they can be used by the browser. The value of the `a:body` property is a URI of the content of the annotation; in this case the server implementation chose to store the text in a separate location and give it its own URI.

With this little bit of ad hoc packaging we can have a POST method that explicitly creates two resources at the same message and a GET method that returns these same resources in one message. This packaging protocol has the additional advantage that it makes POST and GET of multiple resources an atomic operation; there is no window in which another client might modify the annotation body after the annotation properties have been returned but before the body is returned.

A.2 Querying an annotation server

An annotation server is queried for the URIs of annotations it may hold using the GET method. Since the client will most commonly wish to query for annotations that have an `annotates` property naming a specific page that the user may currently be viewing, a particular query parameter is designated to pass the URI of that page, as shown in Figure 12.

The query parameter `w3c_annotates` may be best thought of as an abbreviation for the longer property name `http://www.w3.org/2000/10/annotation-ns#annotates`; that is, this GET is a short-hand for a query that says "return the names of resources that are the subjects of RDF statements in which the predicate is `http://www.w3.org/2000/10/annotation-ns#annotates` and the object is `http://example.com/some/page.html`". The server responds to this GET request by returning RDF/XML describing the properties of each annotation that has an annotates relationship to the given URI. In the first release of our server implementation, we return all the properties of each annotation including the URI of the body resource. Figure 13 illustrates a typical response; in this case there is only one annotation for the specified page.

A.3 Downloading an annotation

An annotation is downloaded from an annotation server using the GET method and specifying the annotation URI, as returned in a query response (Fig. 14).

The response to this GET will be as in Figure 13.

A.4 Updating an annotation

An existing annotation is updated using the PUT method, specifying the URI of the annotation we wish to update. For example, to update the annotation created in the messages illustrated in Figures 10 and 11 above, we might specify the message in Figure 15.

A.5 Deleting an annotation

An annotation is deleted using the DELETE method, specifying the URI of the annotation we wish to remove. For example, to delete the annotation created in the messages illustrated in Figures 10 and 11 above, we might specify the message in Figure 16.

```
POST /annotations HTTP/1.1
Host: www.example.org
Content-Type: application/xml
Content-Length: 1082

<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:a="http://www.w3.org/2000/10/annotation-ns#"
       xmlns:d="http://purl.org/dc/elements/1.1/"
       xmlns:h="http://www.w3.org/1999/xx/http#">
 <r:Description>
  <r:type resource="http://www.w3.org/2000/10/
annotation-ns#Annotation"/>
  <r:type resource="http://www.w3.org/2000/10/
annotationType#Comment"/>
  <a:annotates r:resource="http://example.com/some/page.html"/>
  <a:context>#xpointer(id("Main")/p[2])</a:context>
  <d:creator>Ralph Swick</d:creator>
  <a:created>1999-10-14T12:10Z</a:created>
  <d:date>1999-10-14T12:10Z</d:date>
   <a:body>
    <r:Description>
     <h:ContentType>text/html</http:ContentType>
     <h:ContentLength>250</http:ContentLength>
     <h:Body r:parseType="Literal">
  <html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ralph's Annotation</title>
  </head>
  <body>
  <p>This is an <em>important</em> concept; see
    <a href="http://example.com/other/page.
html">other page</a>.</p>
  </body>
  </html>
    </h:Body>
   </r:Description>
  </a:body>
 </r:Description>
</r:RDF>
```

Fig. 10. Creating an annotation with POST.

```
HTTP/1.1 201 Created
Location: http://www.example.org/Annotation/3ACF6D754
Content-Type: application/xml
Content-Length: 404

<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:a="http://www.w3.org/2000/10/annotation-ns#"
       xmlns:d="http://purl.org/dc/elements/1.1/">
 <r:Description about="http://www.example.org/Annotation/
3ACF6D754">
  <a:annotates r:resource="http://example.com/some/page.html"/>
  <a:body resource="http://www.example.org/Annotation/
3ACF6D754text"/>
 </r:Description>
</r:RDF>
```

Fig. 11. Sample response when creating a new annotation.

```
GET /annotations?w3c_annotates=http://example.com/some/page.
html HTTP/1.1
Host: www.example.org
Accept: application/xml
```

Fig. 12. A query for annotations related to http://example.com/some/page.html.

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 689

<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:a="http://www.w3.org/2000/10/annotation-ns#"
       xmlns:d="http://purl.org/dc/elements/1.1/">
 <r:Description about="http://www.example.org/Annotation/
3ACF6D754">
  <r:type resource="http://www.w3.org/2000/10/
annotation-ns#Annotation"/>
  <r:type resource="http://www.w3.org/2000/10/
annotationType#Comment"/>
  <a:annotates r:resource="http://example.com/some/page.html"/>
  <a:context>#xpointer(id("Main")/p[2])</a:context>
  <d:creator>Ralph Swick</d:creator>
  <a:created>1999-10-14T12:10Z</a:created>
  <d:date>1999-10-14T12:10Z</d:date>
  <a:body r:resource="http://www.example.com/mycomment.html"/>
 </r:Description>
</r:RDF>
```

Fig. 13. A typical response to the query in Figure 12.

```
GET /annotations/3ACF6D754 HTTP/1.1
Host: www.example.org
Accept: application/xml
```

Fig. 14. Downloading a specific annotation.

```
PUT /annotations/3ACF6D754 HTTP/1.1
Host: www.example.org
Content-Type: application/xml
Content-Length: 657

<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:a="http://www.w3.org/2000/10/annotation-ns#"
       xmlns:d="http://purl.org/dc/elements/1.1/">
 <r:Description about="http://www.example.org/Annotation/
3ACF6D754">
  <r:type resource="http://www.w3.org/2000/10/
annotation-ns#Annotation"/>
  <r:type resource="http://www.w3.org/2000/10/
annotationType#Example"/>
  <a:annotates r:resource="http://example.com/some/page.html"/>
  <a:context>#xpointer(id("Main")/p[2])</a:context>
  <d:creator>Ralph Swick</d:creator>
  <a:created>1999-10-14T12:10Z</a:created>
  <d:date>1999-10-14T13:14Z</d:date>
  <a:body>
    ...
  </a:body>
 </r:Description>
</r:RDF>
```

Fig. 15. Updating an annotation using PUT.

```
DELETE /annotations/3ACF6D754 HTTP/1.1
Host: www.example.org

HTTP/1.1 200 OK
```

Fig. 16. Deleting an annotation using DELETE.

Proceedings of the International
Workshop on Information Presentation
and Natural Multimodal Dialogue
(IPNMD 2001), Verona, Italy, 2001.

FIRE: An Information Retrieval Interface For Intelligent Environments

Krzysztof Gajos and Ajay Kulkrani*

MIT Artificial Intelligence Laboratory
`{kgajos, kulkarni}@ai.mit.edu`

Abstract. Searching for relevant information on the world-wide web is often a difficult and frustrating task. The information one is looking for, is hidden among thousands of documents returned by a search engine. One way of making search for relevant information easier, is to create better interfaces to the search engines; interfaces that facilitate quick and efficient browsing through the multitude of returned documents. In this paper, we present FIRE - a multimodal interface for information retrieval deployed in the Intelligent Room at the MIT AI Lab. FIRE differs from most other interfaces for information retrieval in that it combines a couple of interaction modalities to improve the search process.

1 Introduction

This paper presents the current state of our work on a new multi-modal interface-situated in an Intelligent Environment-for retrieving information from the web. The work brings together progress made in three research areas: multi-modal interfaces, interfaces for information retrieval, and intelligent environments.

The motivation for building FIRE (the Friendly Information Retrieval Engine) was three fold: first, we wanted to build a very natural and effective information retrieval interface. Second, we wanted to demonstrate new capabilities that become possible when building

---

* This work was funded by Acer Inc., Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under the MIT Project Oxygen partnership, and by DARPA through the Office of Naval Research under contract number N66001-99-2-891702.

applications situated in an Intelligent Environment (IE). Finally, we wanted to test the limitations of the technology we have developed for our IE.

FIRE takes advantage of the numerous display devices that many IEs offer and of the ubiquity of speech input and output in such spaces. Our current implementation of FIRE was build within the Intelligent Room Project [2] at the MIT AI Lab.

1.1 Problems with search engines

There are numerous problems with how the current search engines and Web directories organize the information [6]. Many of them stem from the current trend to assign each document to exactly one category. That makes it difficult to look for information that relates to several categories at once. Also, if one wants to browse a number of documents relating to a particular topic, one often needs to traverse a large number of sub trees in order to find all of the relevant information. This has to do with which nodes were chosen as top nodes in the category tree, and which were placed further down. For example, if we were to look for documents on the economy of European countries, it would really matter if the tree was organized like this:

$Economy \rightarrow Regional \rightarrow Europe \rightarrow Poland$

or like this:

$Regional \rightarrow Europe \rightarrow Poland \rightarrow Economy$

In the first case, we can just browse all documents under Europe and all of them will be somewhat relevant to our search. In the second case, if we look at all documents under Europe, we will get information about countries' geography, culture, customs, etc., as well as the economy.

We have designed FIRE in a way that allows browsing of relevant information returned by a search engine, even if the category tree had not been constructed in our favor, or if the topic of interest spans several dinstinct categories.

2 Related work

A number of approaches have been suggested to make searching large centralized corpora for relevant information easier. Three of the main trends are summarized here.

Preprocessing and annotating information. The START natural language query system employs this strategy to return the most relevant information in response to a query [7]. The strength of this approach is that it provides just the right information in response to a query. Its main weakness is that it requires a lot of human effort to set up and maintain.

Processing retrieved documents based on content. Documents returned by keyword-based search engines are analyzed based on their content and grouped according to some measure of similarity. The Scatter/Gather [4] algorithm is a prominent example of this strategy. The strength of this approach is that it allows browsing through collections of uncategorized documents. Unfortunatelly, the entire body of the documents needs to be analyzed thus drastically impacting the speed of the retrieval process.

Advanced visual interfaces, such as Cat-a-Cone [5], organize categorized collections of documents visually in a way that makes browsing and selecting the most relevant information easier. Many of such interfaces make it easy to explore several categories simultaneously and to provide instant access to a large portion of the information base at once, without cluttering the screen. Their shortcoming is that they provide no direct access to the information not presented on the screen. Also, they rely on documents being already categorized.

3 FIRE

The key goal of the work on FIRE is to create an interface that will provide a natural and efficient way of searching and browsing documents on the World Wide Web. FIRE is meant to use one or more of the existing search engines. It provides tools for easily identifying and selecting the most relevant search results from the hundreds or thousands returned by the search engine.

FIRE takes the visual interface approach (described in the previous section) one step further: although it still relies on search engines that categorize their search results, it provides a way to easily reach both visible and invisible search results. It also attempts to make browsing through the returned information easier by incorporating several modalities. Instead of using just a purely visual interface, FIRE combines several modalities: a multi-display graphical component, a pointing device, as well as speech input and output. What is more, FIRE is deployed in an IE, an immersive multimodal environment, where users interact multimodally not only with FIRE but also with the environ-

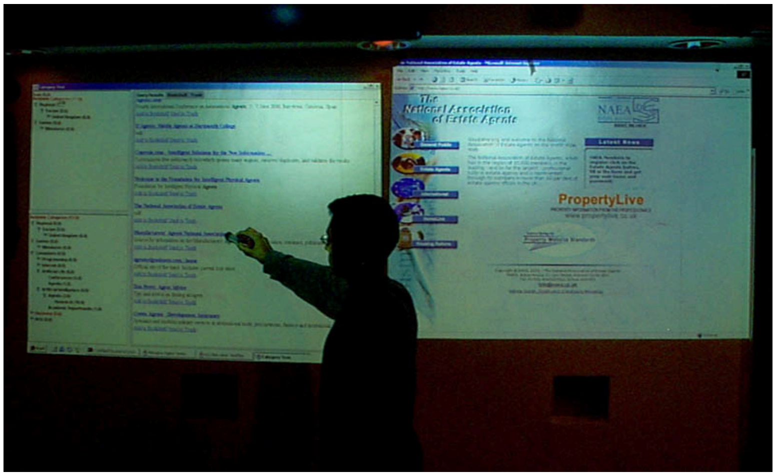


Fig. 1. FIRE in action: the left display shows the FIRE interface; the browser with the most recently selected document is displayed on the right.

ment itself, including devices (such as lights and projectors) and other software (e.g. the browser or the display manager).

FIRE makes search for information particularly effective when speech and gesture are used together to complement one another. It has been observed, however, that users rarely use all of the available modalities simultaneously but tend to pick one mode or switch between modes [8]. For that reason, FIRE is also perfectly usable if used just with a pointing device, or if driven solely by speech with visual feedback. Our initial tests indicate, however, that users familiar with the Intelligent Room find it easy and convenien to use both modalities at once, at least part of the time.

3.1 The interface

FIRE's interface has a number of graphical components interacting with one another. After the user makes a query, and information is retrieved from the Web, a *tree of all potentially relevant categories* is displayed. This tree is constructed based on what categories the documents returned by a search engine belonged to. Simple heuristics are applied to rank the categories in order of most likely relevance. The categories deemed as more relevant are displayed towards the top of the tree and the font size is proportional to the predicted relevance.

Another view shows the tree of *categories selected by the user*. Here the categories are ranked based on user's feedback ("it surely has to do with economy" vs. "it may have something to do with politics"). Our intention was to provide a space where user could see at a single glance all of the categories that he considers worth browsing though.

Another large component shows the *currently analyzed documents*. Whenever the user focuses on a set of categories, relevant documents are presented there. Each document is shown as a title and a short summary.

There is also the *local bookshelf* where the user can place relevant documents for short-term storage. Depending on the availability of resources and on user's preferences, the bookshelf can be placed on a separate display or together with the main part of the interface. The main display always has an icon where the user can place newly found documents to be moved onto the bookshelf.

The trashcan is the last component of the main part of the interface. As the name implies, the trashcan is a container for all discarded elements such as documents and categories. It was added relatively late in the development process. We have realized that sometimes users wanted to undo some of their operation after a relatively long time. Simple undo mechanism was not adequate in such situations. It became clear that we needed to give the users a way of verifying what items have been discarded.

Finally, there is a *browser*, used to show full text of the documents. The browser is almost always placed on its own display unless none is available.

We use two input modalities: speech and gesture. Either of them can be used alone to accomplish the task. However, each of them is better suited for some parts of the process than for the others.

Role of speech Speech in FIRE is used for four main tasks:

Taking shortcuts and probing invisible parts of the category tree and document base. For example, if a user asks about "agents", the system will display main categories such as Travel, Business, or Computers. The user, can quickly probe the system by asking "Do you have anything related to Artificial Intelligence?" If Artificial Intelligence is among the categories associated with any of the returned documents, the system will show the subset of documents about agents that are in that category and all of its subcategories.

Accessing multiple parts of the tree at once. As described in the example in Section 1.1, to view documents about the economy of European countries, the user may need to visit many separate branches in the category tree. Using FIRE, the user may very conveniently use speech to say “My query has to do only with Economy,” and *all* of the branches about economy will be presented and all other branches will be discarded.

Speech is also very useful for *command and control* part of the interaction. It can be used to undo actions or to manipulate the interface itself.

Finally, speech can often be used to make the *initial query.* In cases where the query includes uncommon terms, the user can easily fall back on a keyboard.

Role of gesture FIRE uses standard gestures such as selection or drag-and-drop. Its strength comes from incorporating novel input devices such as a laser pointer (whose position is tracked in real time with a camera), or an on-wall display with a specially instrumented electronic marker. Gestures are used to interact with the interface in a traditional GUI style, and to set context for spoken commands.

In particular, by using hand gestures the user can browse through the available categories and documents, and select documents for viewing and moving onto the bookshelf.

We are currently in the process of adding two new gestures: strike-through to delete (i.e. move to trash), and circling to select one or multiple objects.

4 Modalities: recognition and integration

FIRE uses relatively unsophisticated—yet effective—recognition methods for speech and gesture recognition. For gesture recognition we use primarily a pen-like pointing device, which can be used to interact accurately even with small objects on the screen. The two gestures we currently recognize (pointing and drag-and-drop) are unambiguious and easy to recognize. The two other that we are in the process of adding (strike-through and circling) are not as trivial but still easy to recognize correctly.

Our speech recognition system [3] is entirely grammar-driven. This ensures very good recognition rate and makes the processing of the spoken utterances straight-forward. Our choice of tools has made the implementation process easy at the expense of the “naturalness” of our interface. Hand and finger gestures would be preferred to pen strokes for pointing, and unrestricted speech recognition would eliminate the problem of user occasionally trying to use a phrase that is not in the grammars.

The benefits of our approach are very low recognition error rate and ease of development. Grammar-driven speech recognition engines make it easy to recognize and parse complex utterances. Thanks to this, our users can make statements like “My query has to do with Economy, Politics and Government but not with Culture or Travel” or “It has nothing to do with Artificial Intelligence but it might be relevant to Programming.” There are very few such hybrid constructs that we observed people using during their interactions with FIRE, and they are easy to describe within a grammar.

Because of the good recognition accuracy of our speech and gesture recognition systems, the integration of modalities is done at the post-recognition stage in FIRE, and the modalities do not cross-influence one another. In practice, therefore, multi modal integration in FIRE is restricted to the resolution of diactic references in utterances like “this category is not relevant,” “move this to the bookshelf,” or “put this there.” In the case of the last utterance, we need to resolve two references.

The context for resolving these references may be set by either speech (e.g. “What do you have under HCI” sets context to the HCI category) or gesture.

When we do the integration, we use temporal co-occurance and semantic compatibility to verify that the current context is relevant to the spoken command. If we cannot resolve what the user is referring to, we request clarification. For example, in case of the “put this there” command, if we cannot detect a valid destination for an object, we will ask the user “Where do you want me to put it?”

It is not to say, however, that the integration task has been made trivial. There are still cases that require some semantic analysis of recent events in order to establish how to resolve references best. For example, let us assume that the user drags a document to a trashcan. If the then says “put this there as well” while pointing at another document, “this” will be resolved to mean the new document and “there” to mean the trashcan. If instead she were to say “I actually meant to put it there” while pointing at the bookshelf, this time “this would refer to the document that was previously placed in the trashcan and “there” would mean the bookshelf.

5 Sample interaction

User says "I need information about agents." FIRE contacts a search engine and retrieves the results. It then displays a tree of all potential categories and a list of a few documents that appear most relevant. The top categories are Computers, Business and Travel. The user drags Computers onto the area with chosen categories. This sub-tree is expanded as deep as possible given available screen space (giving preference to those branches that are predicted to be more relevant). Again, most relevant documents are shown, this time only from the branch relevant to Computers. The user now asks "Do you have anything under HCI?" HCI is not visible on the screen but, indeed, under $Computers \rightarrow ArtificialIntelligence$ there is HCI. FIRE expands the right part of the tree and shows documents under HCI. The user selects some of them with a pointing device and they appear in the browser. Those that are particularily interesting, the user moves onto the bookshelf icon.

The user can now ask "Is there anything under Programming Languages?" FIRE replies that there is nothing but then the user notices that there is a branch called Programming under Computers. Selecting this branch with a pointer, reveals a number of documents about current agent programming tools. The user moves some of them onto the bookshelf. Saying "I am done" clears the main interface and brings up the bookshelf with all the documents placed there during the search. Now the user can evaluate the quality of the collected material and, potentially, save it for future reference.

6 Evaluation

Our initial informal experiments have shown that the itnerface is comfortable to use after a short initial training. Users were given a short explanation of the individual elements of the interface, and the extent of things they could use speech for. Our test users were members of the Intelligent Room project, already familiar with other multi-modal applications running in the Room. The users were particularily happy with the bookshelf, and with the ability to quickly browse the search results by category. Users have also commented favourably on having separate windows for browsing the returned results and for viewing the full documents.

On the negative side, users commented on our Spartan interface, and they found having two separate category trees unnatural, though were not able to suggest a different method of keeping track of selected categories while having access to all the rest of them. They also found the speech interface somewhat brittle. This particular concer we will address in the next setcion.

7 Futher work

We are currently working on a number of improvements to the system. Most significantly, we are in the process of incorporating new speech recognition software based on the Galaxy [9]. This engine is speaker independant and while it also works in a grammar-driven mode, it is much more flexible in that the grammars specify only the keywords and the general structure of the allowed utterances. In Galaxy, the grammar descriptions can contain wild-cards and thus allow for wider linguistic flexibility.

We are also in the process of integrating Haystack [1] with FIRE. Haystack is a personal information management tool. After integrating with FIRE, it will be able to answer questions like "When I was looking for information on agents yesterday, did I see anything about 007?"

Finally, we are developing an algorithm that will allow us to rebuild the category tree returned to us by the search engine in a way that best suits a particular search.

8 Contributions

FIRE demonstrates how the new potentials for human-computer interaction—that become available with the emergence of Intelligent Environments—can be used to build an effective and natural interface for information retrieval. IEs usually have more resources than a single desktop computer. If those resources become available, FIRE makes effective use of them. It uses up to three displays to separate navigation through information space from previewing retrieved documents. FIRE also benefits from the ubiquity of speech input and output in a smart space: while in such a space, the user does not have to make any special effort to start interacting with FIRE by means of speech, because all other compontents of the space use speech already. In comparison to purely visual interfaces, through the use of speech FIRE allows easy access to multiple parts of the category tree at once and makes it possible to take shortcuts to invisible parts of the information space.

References

1. E. Adar, D. Karger, and L. Stein. Haystack: Per-user information environments. In *Proceedings of the 1999 Conference on Information and Knowledge Management, CIKM*, 1999.
2. Michael Coen. Design principles for intelligent environments. In *Fifteenth National Conference on Artificial Intelligence (AAAI98)*, Madison, WI, 1998.
3. Michael Coen, Luke Weisman, Kavita Thomas, and Marion Groh. A context sensitive natural language modality for the Intelligent Room. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.
4. D. R. Cutting, J. O. Pedersen, D. Karger, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proc. of the 15th Int. ACM/SIGIR Conference*, 1992.
5. M. Hearst and C. Karadi. Cat-a-Cone: An interactive interface for specifying searches and viewing retrieval results using a large category hierarchy. In *Proc. of the 20th Int. ACM/SIGIR Conference*, Philadelphia, PA, 1997.
6. Marti A. Hearst. Interfaces for searching the web. *Scientific American*, March 1997.
7. Boris Katz. From sentence processing to information access on the world wide web. In *Proceedings of the Dans AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, 1997.
8. Sharon L. Oviatt. Ten myths of multimodal interaction. *Communications of the ACM*, 42(11):74–81, November 1999.
9. S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, , and V. Zue. Galaxy-II: A reference architecture for conversational system development. In *Proceedings of ICSLP 98*, Sydney, Australia, November 1998.

Proceedings of the 5th International
Conference on Information
Visualisation, London, 2001.

Query by Attention: Visually Searchable Information Maps

Mark A. Foltz and Randall Davis

MIT Artificial Intelligence Laboratory
{`mfoltz, davis`}`@ai.mit.edu`

Abstract. This paper explores how the design of information spaces might be grounded in knowledge of human visual processing, notably what kinds of visual selection are most efficient. Information maps spatially array graphical symbols representing items of information and their attributes. Ideally, their users should be able to do *query by attention:* answer questions about the information quickly by controlling visual attention (i.e., through spatial selection and visual search), instead of manipulating an interface. We propose a preliminary method for designing visually searchable maps based on experimental results about what kinds of visual search are easy. The hope is that the resulting maps will better employ the perceptual capabilities of their viewers when they search. An example information map of recent movies illustrates the approach.

1 Information maps and visual search

Reading a map like that in Figure 1[1] to navigate the Boston subway requires at least two episodes of visual search: find the originating station, then find the destination. Could we improve the design of the map by using our knowledge of what kinds of visual search are easiest?

A small visual stimulus that appears to be at a different depth relative to its surroundings "pops out" – it can be found almost instantaneously [27, p. 39]. We could add such a stimulus to a map posted inside a station to indicate that station (Figure 1, right). Now a map

---

[1] Color versions of the figures are available at `http://www.infoarch.ai.mit.edu/publications/`.

reader (with an appropriate legend) can locate the station by efficient visual search, instead of scanning the text labels.[2]

Designers of information spaces would like to make the same kinds of improvements to more complicated displays. This paper explores the relationship between information space design and visual search, in the hope that spaces can be made that facilitate rapid visual queries. The preliminary proposal is that they should be designed to make maximal use of spatial selection through attentional control, and visual properties that pop out, i.e. require almost no effort to find even in a display with many items.

In this paper, an information map is a display that spatially arrays graphical symbols representing information items and their attributes in two dimensions. By *information item* we mean an object with attributes meaningful to users, e.g. a movie, document, or historical event. The task is to obtain the subset of items whose attributes match the query criteria. For example, in the Beethoven symphony map in Figure 2, we can find which pieces were composed before 1808 by attending to the upper third of the map. Or, we can find the symphonies in minor keys by attending to the textured symbols. Once attention has been directed to the subset of the items, further inferences can be drawn about them (e.g., no symphony in C was composed after 1808). (This map is similar in layout to GlassEye, a visualization of Phillip Glass' works [14], but was designed independently.)

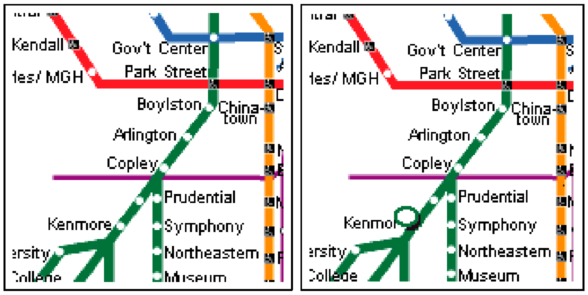


Fig. 1. Left, part of the Boston subway map. Right, an improvement using a depth cue to highlight the nearest station.

---

[2] Using Wolfe's classification of search task efficiency [27]. A classic example of efficient search is locating a single `X` among `O` distractors. Search time increases with display size with a very shallow slope (< 5 ms/item).

1.1 Why information maps?

Information designs that use the position of graphical primitives have long been used to concisely convey data [3]. An advantage of these designs is that the user can *query by attention* – answer questions by controlling visual attention (and receiving immediate feedback), rather than manipulating a database query interface and waiting for a server to return results. The user can more rapidly explore the information space because he needs to alter only his visual attention (and not the interface) to adjust the parameters of the query.

Information maps, which give meaning to a item's location, also engage their viewers' capabilities for spatial imagery and spatial memory. If the user remembers an item's location, he can return to the map and find it there, instead of repeating a textual query or scanning a list. With the right design, information maps are a less cumbersome tool for querying and drawing inferences from the ever-growing amount of information we are faced with daily.

Information maps have limits: there are a limited number of independent visual properties that support rapid visual search. And, using too many properties has its drawbacks. A map that uses its symbols' position, color, shape, depth, *and* orientation to encode information risks overloading its viewer's ability to attend only to the properties relevant to his information need, by the presence of multiple distracting properties. Another risk is that the mapping of properties to meanings must be retained in the viewer's limited working memory.

To get around the problem of a limited number of visual attributes, maps can be interactive (displayed on a computer). The user can obtain multiple views of the same information by selecting relevant attributes to display. If there are too many items to display without crowding, he can prefilter the information in the map [1] or browse it with a hierarchy. These methods help to bridge the gap between query by attention and query by manipulation.

2 Designing for query by attention

Consider the common scenario in which a user wants to explore a small database by posing a group of related queries (a task that would ordinarily require several transactions with a textual interface). Assume that the database is small enough that a map can display all the items without overcrowding.

The information is treated as a bag of items; adding or removing an item would not substantially change the overall meaning of the map.

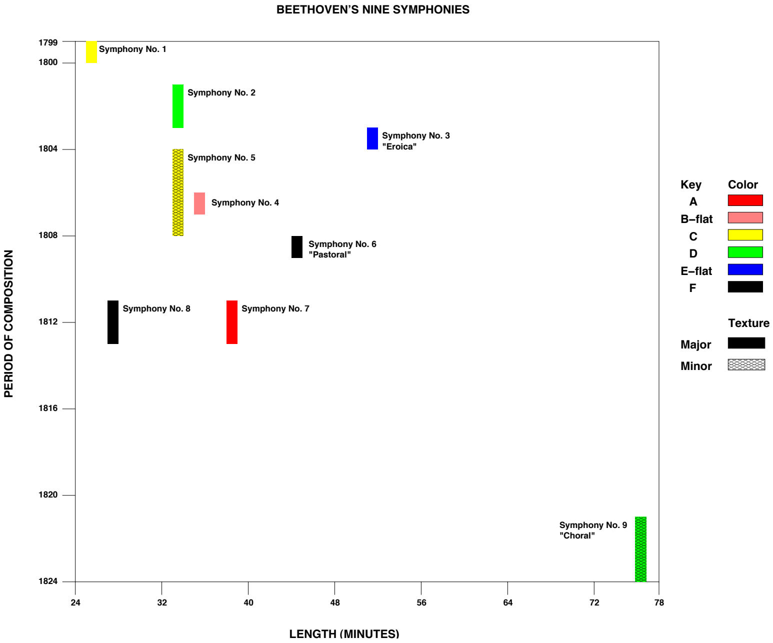


Fig. 2. An information map of Beethoven's nine symphonies. Color and texture are used to indicate the key of the composition.

Each item has attributes, which are variables that can be binary, enumerated (chosen from an unordered set), integer, real, or textual.

A *query* is a set of restrictions on the values of an item's attributes that must all be satisfied for an item to match. An example of a query in Figure 2 is `Key = F AND Year < 1810`. The user explores the information space bottom-up – adding and removing restrictions to a working query and seeing which items match.

These maps are a simple kind of information space that set the context for this initial sketch of the constraints and affordances that relate visual search to information space design.

2.1 A design method

We propose a four-step method for designing a visually searchable information map:

1. List the queries viewers are likely to make and the information attributes that are available to answer those queries.

2. Propose a mapping of information attributes to visual properties, using the design flowchart (discussed below) to suggest which mappings can be searched efficiently.
3. Produce an initial map and test it to see if users can quickly and easily make the kinds of queries listed in Step 1.
4. If some of the queries are slow, adjust the mapping of information attributes to visual properties and repeat Step 3.

Of course the real difficulty in this method lies in Steps 2 and 4 – finding the initial mapping and adjusting it so that the map is usable in practice. The design flowchart discussed below constrains the possible mappings in terms of attribute types, but does not specify a unique mapping. Instead, this approach is iterative: using what we know about the cognitive affordances for search, try an initial design with these guidelines, then iteratively improve it with feedback from users. A good design solution will match the user's patterns of semantic attention across the information with patterns of visual attention across the map.

Another difficulty is that the number of relevant attributes to display often exceeds the number of available visual dimensions. In this case, the designer must consider which attributes best support the user's query goals. (Dimension reduction can also help, as discussed below.) Finding the best attributes often requires user input, iterative prototyping, and careful task analysis [12], which are beyond the scope of this paper. Here we assume we are given a data model and consider how to display it in a visually searchable form.

3 Visual properties

This section considers in more detail the visual properties available to represent a symbol in the map. We begin with single visual properties that can define a symbol's appearance, then consider combinations of those properties that can support efficient conjunctive search. This summary is based largely on Wolfe's review of visual search experiments and results [27], and Feature Integration Theory [25].

3.1 Single properties

Research has shown that many visual properties can support efficient visual search. Below we consider visual properties that have been used to convey information in print information design [3, 26], and are also discussed in Wolfe's review.

Position. A user can control visual attention to search within a spatially delimited region, making it appropriate to map to the X- or Y-axis attributes that are queried by range (e.g. *Find all films in the 1980s*). The user can quickly shift attention to different scales or regions, easily adjusting the query.

An axis can also be subdivided to allow one spatial dimension to encode more than one attribute. (This approach has been independently applied to graph multivariate functions [17].) A primary, enumerated attribute can partition an axis into subaxes, and a secondary, continuous attribute can be mapped onto each subaxis. For example, we could add Beethoven's contemporaries to Figure 2 by partitioning the X axis by composer, then plotting symphony length on each subaxis. Although subdivision allows more attributes to be represented at once, searches that require the secondary attribute but not the first require the serial examination of several disjoint regions of the map.

Color. Color is widely used to distinguish symbols in maps. The colors at the extrema of opponent processing [19, p. 113] (red/green, blue/yellow, and black/white) are considered good candidates for efficient search. D'Zmura proposes that a color will pop out if it is linearly separable in CIE color space from distractors [9], while other work has shown that search for a color target among as many as nine distractor colors is efficient if the colors are spread far apart in color space [22].

This suggests choosing colors that are symmetrical in a ring of high saturation in color space, maximizing distance and separability (Figure 3). If white is reserved for the background, then at least five colors are left for map symbols. Color is best for enumerated attributes that can take two or more values. [3]

Shape. Because there are many parameters that determine shape, coming up with a fixed set of rules for finding shapes that can be searched for efficiently is difficult. Some determining features include line termination (presence/absence), closure, holes, and possibly intersection [27, pp. 31-4]. Most experiments described by Wolfe report efficient search with homogeneous distractors, so a conservative approach would map shapes like `X` and `O` to a binary attribute. Orientation also supports efficient search, for example using bars oriented at 0 or 90 degrees.

Motion. Motion exhibits search asymmetry [21] – a moving target among stationary distractors is easy to find, while a stationary target among moving distractors is not. Thus, a map that uses motion to code a binary attribute is biased against searches for the stationary value.

---

[3] It is important to note that 8 percent of men and 1 percent of women are colorblind [19, p.104], so visualizations that use colored symbols should provide an alternate presentation of that information (e.g. with shape).

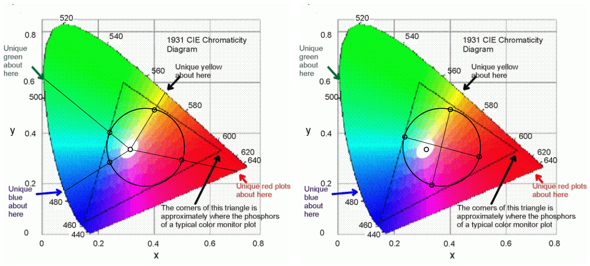


Fig. 3. Left, a set of symbol colors (open circles) chosen to approximate the extrema of color opponent processing (indicated by arrows). Right, colors chosen to maximize separability. The triangle encloses the colors displayable on a CRT monitor.

However, the user may choose to render a subset of the symbols as salient in an interactive map, and search among them. This prospect is appealing, because search for motion in conjunction with other properties can be efficient (as discussed below).

Depth. Pictoral depth cues such as shading, occlusion, or shadows can support efficient search [27, p. 39]. This cue is best used for binary features, as search in more than two depth planes is less likely to be efficient.

Other cues. Wolfe lists other basic features that can be searched for efficiently: vernier offset, curvature, gloss, size, etc. While these are also worth investigating as useful for information maps, the cues discussed above – position, color, shape, motion, and depth – have been investigated thoroughly in the visual search literature. This initial proposal explores the possibilities they afford for information map design.

3.2 Combinations of properties

The original version of Feature Integration Theory [25] held that visual properties are preattentively and simultaneously processed into feature maps across the visual field. Searching for a target defined by a single property is fast because only one feature map is consulted to locate it. To search a conjunction of features, however, information from multiple feature maps must be bound together at item locations, and each location serially checked to see if it possesses the target conjunction.

Later work has shown that for certain combinations of distinct visual properties, subjects can perform conjunctive searches efficiently.

In particular, Egeth et al. as well as others have shown that subjects can efficiently search a colored subset of items for a differently shaped target [10]. Efficiency improves when the targets are far apart in color space.

Other work has shown that visual search is efficient for a moving stimulus among stationary distractors that also differs in depth, orientation, or shape [7, 8, 15, 16] (but as mentioned, motion search is asymmetric).

Stereoscopic depth can also be used with color or orientation for efficient conjunctive search [18]. Computer-generated stereo displays usually require special viewing equipment, however.

These results are important because they suggest a user can query by at least four attributes, first by spatial selection (using two real-valued attributes), and then by searching for a conjunction (e.g., color and shape).

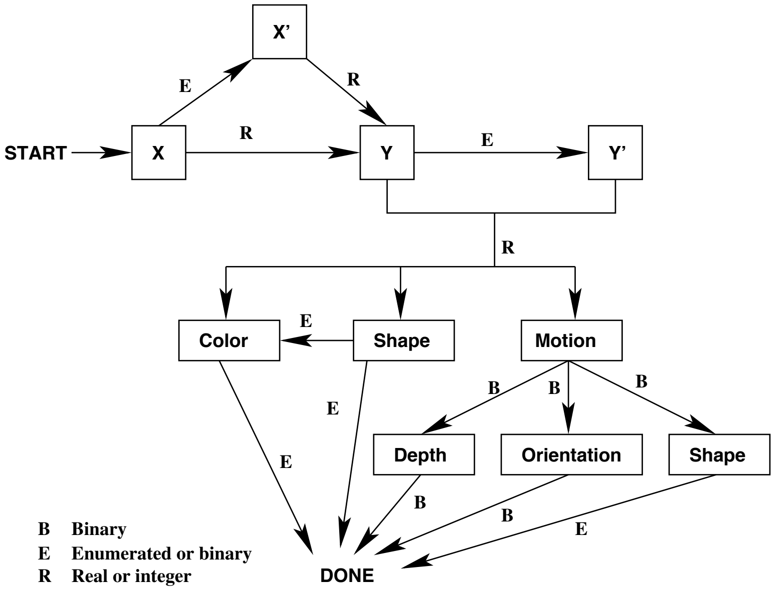


Fig. 4. A design flowchart for visually searchable information maps. A map design traces a path from START, assigning visual properties to information attributes along outgoing edges. X' and Y' represent the subaxes created when the X and Y axes are partitioned by an enumerated attribute. The path may terminate before reaching DONE.

3.3 Summary

The flowchart in Figure 4 summarizes the suggested design combinations that afford efficient search. A mapping is created by tracing a path from START in the graph, assigning visual properties to information attributes of the indicated types. The edge chosen to leave a visual property node assigns its information attribute (e.g., placing `SymphonyLength` on the **R** edge leaving **X** means that the map will have symphony length on its X-axis). Names or other labels may be ordered alphabetically and treated like integers in the flowchart, or included as text adjacent to map symbols. (For simplicity, the latter case is not included in the flowchart.)

4 Example information maps

To illustrate the method, we first consider the problem of designing an information map to visualize a database of about 25 movies. Suppose the database contains the attributes shown in Table 1.

Attribute	**Type**	**Example**
Title	textual	`Titanic`
Year	integer, 1900-2001	`1999`
Genre	enumerated	`Drama`
Oscar	binary	`Y`
Review	real, 0.0-10.0	`6.8`

Table 1. Movie database attributes and example values. The review scores are from Internet Movie Database votes.

The first step in the design procedure is to list the questions users might ask of the data. The questions in this list depend on the motivation of the prototypical user (e.g., a video rental browser versus a film researcher). Table 2 shows some sample questions that might be asked by someone browsing for good, recent movies.

In these sample questions, both `Year` and `Rating` are ordered attributes queried by range, so they are good candidates to map to axes. `Oscar` is binary and can be mapped to two shapes, while `Genre` is enumerated over five values and can be mapped to five colors.

Combining these observations, a possible design path is shown in 5, with the corresponding information in Figure 6. We can now predict

Question	**Query**
Was “Aliens” any good?	`Title = 'Aliens'`
Which recent comedies are well-reviewed?	`Year > 1995 AND Genre = Comedy AND Review > 7.5`
When have the Oscars made an unpopular choice?	`Review < 7.5 AND Oscar = Y`
What movies received Oscars in the eighties?	`1980 <= Year < 1990 AND Oscar = Y`
Has any horror film ever received an Oscar?	`Genre = Horror AND Oscar = Y`

Table 2. Sample questions a user might use a Movie Map to answer.

whether users would be able to answer the sample questions efficiently when using a map with these assignments. The first question requires searching the symbol labels serially, because it uses the `Title` attribute, which is not mapped to a searchable property in this design. The remaining questions involve subsets of attributes on the design path, and are predicted to require quick visual search.

4.1 Mapping mutual funds

We illustrate the use of the flowchart with another example (Figure 7), in this case to design information maps of mutual funds. Suppose the available attributes are `Name`, `InvestmentObjective`, `YTDReturns`, `5YearReturns`, `Beta`, and `NetAssets`.

The first design (Figure 7, top) allows the user to weigh the tradeoff between volatility and performance for funds of a given size and investment objective. It uses a partition of the X axis to indicate the fund's investment objective, and a partition of the Y axis to indicate its size. Within each partition, the horizontal and vertical position of the symbol indicate its year to date and five year returns, respectively. The shape indicates the fund's volatility. In this way, five aspects of the fund can be concisely conveyed and searched.

A second design (Figure 7, bottom) uses the performance measures as the dominant selection criteria. The user can optionally animate a subset of funds by investment objective. The use of the flowchart thus facilitates the exploration of the map design space, suggesting multiple views that can accommodate different sets of query goals.

5 Related work

Pirolli et al. have performed a study that tracks the eye movements of users as they perform textual navigation in a hyperbolic tree [20]. Their analysis finds that the size of the user's attentional spotlight is proportional to the local relevance of the text, and inversely proportional to the text's density. These results are relevant to creating information maps with large data sets, where views that let the user focus on a subset of information while retaining its context will be needed to prevent unusable information densities. Also, their theory focuses on how semantic (textual) cues serially guide the attentional spotlight, while this work focuses on perceptual, preattentive visual properties that the viewer can process in parallel. Clearly these processes are complementary and research to integrate such models is needed.

Card and MacKinlay present a formal way of describing the mapping of information types to visual properties [5]. Their goal is to analyze the structure of the larger information visualization design space, which includes trees, graphs, and hyperbolic views. Their goal is analysis, not design, but their representation would be useful if our work were extended to describe the perceptual constraints in the design of other types of information spaces.

Finally, the SAGE and VQE systems are part of a project to create a presentation design expert system [13, 6]. SAGE creates a media-independent design plan that fulfills the presentation's communicative goals. The design plan is given to a media allocator that can generate textual or graphic realizations of the plan. These design suggestions could be integrated into SAGE to assist its generation of information graphics. VQE adds *threads* that connect multiple views of the same data. Our proposal can be extended to accommodate multiple views by ensuring consistent visual semantics (e.g., never change the meaning of symbol colors across views).

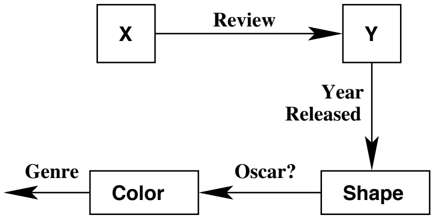


Fig. 5. A design for the Movie Map, which is an example path in the flowchart in Figure 4.

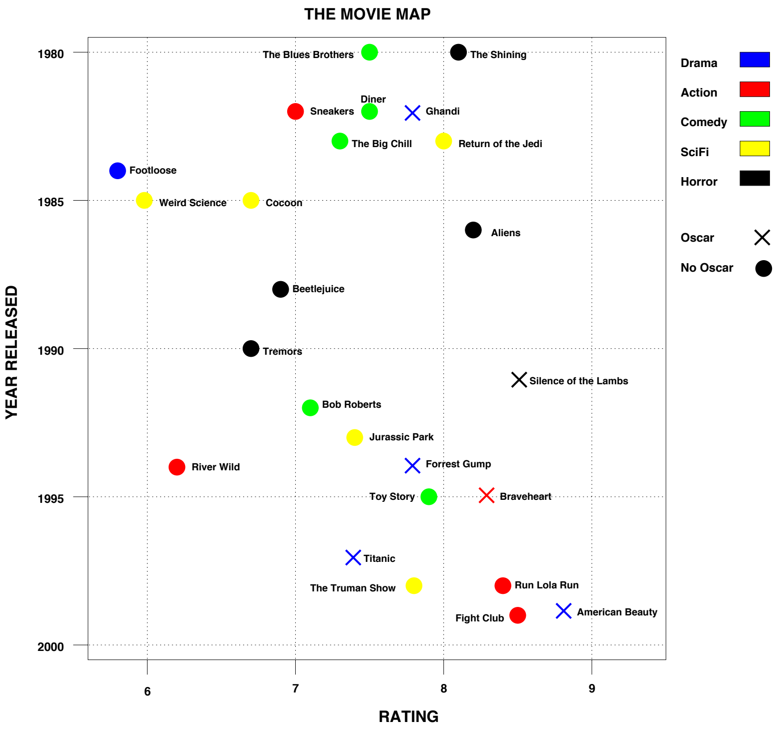


Fig. 6. The Movie Map produced from the design path in Figure 5.

6 Conclusion

This report explores the relationships among spatial selection, visual search, and visualization design. It proposes a method for designing information maps that makes use of experimental results regarding which kinds of visual search are efficient for viewers. These results were obtained from Wolfe's review of the experimental literature on visual search. The goal is to enable the map's users to do *query by attention* – answer questions by controlling visual attention (and receiving immediate feedback), rather than by manipulating the interface. Visual search for information is preferred because it is better suited for people, while database search remains better suited for computers.

While the application of experimental vision science to information design can result in oversimplification of both fields, we believe that useful design principles can be gleaned. Traditional textbooks on in-

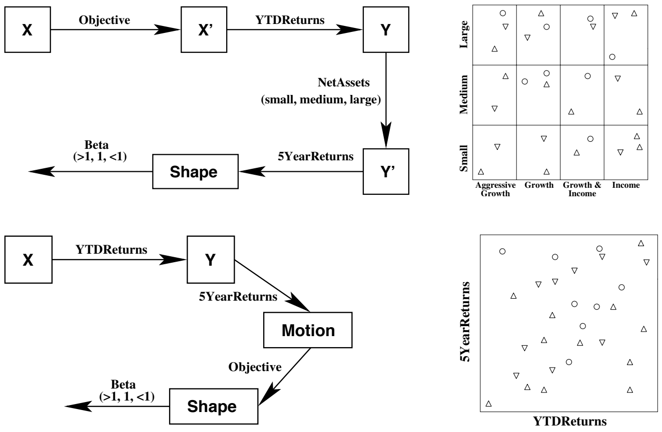


Fig. 7. Two designs of maps for mutual funds. In cases where a real attribute is used on an edge marked E in the flowchart, the value ranges are shown in parenthesis. The figures at right suggest the appearance of the maps.

formation design proceed by examining and critiquing design examples [26], and do not explicitly refer to results from vision science.

It remains to be seen whether this method can scale to real-world design problems, as the number of items in the map (100? 1,000?) and the number of attributes displayed increases. Techniques that give the user the ability to dynamically filter the data by manipulating sliders and toggles have proven effective for exploring large data sets [11, 2, 24]. And, dimension reduction techniques such as principal components analysis can simplify design (but only if the reduced dimensions are informative to users).

Further investigation of our visual capabilities can lead to new design insights. It remains to be seen whether people can search efficiently using arbitrary subsets of visual properties, so that more properties (such as texture, color, and shape) can be used at once. Also, most visual search experiments present a single type of target and distractor; understanding the role of heterogeneous distractors will help determine which visual properties can take more than two values. Using the heuristics discussed here in more design examples would further refine them, along with usability studies that can point out their shortcomings in practice.

It would also be worthwhile to see how well an empirically-based model of search performance could predict the time needed to complete map usage tasks. However, visual search experiments are conducted in carefully controlled conditions that may differ greatly from those in the day-to-day use of an information visualization. Users also tend to reformulate their information needs continuously. These factors would need to be taken into account in such an evaluation.

A future plan is to implement this proposal with an interactive tool that allows a user to rapidly prototype an information map given a small database or spreadsheet. The tool would let the user adjust the mapping of attributes to visual properties, while applying the kinds of perceptual constraints on efficient search presented here. (The tool's function would be similar to VQE [6].)

Efficient visual search in information maps is not possible in all cases. But maps that better employ our visual perception potentially have a great advantage over textual interfaces for understanding and managing information [4, 23]. We believe that visual information tools can benefit from progress in vision science, by offering design guidance related to our perceptual capabilities.

6.1 Acknowledgments

We thank Michael Oltmans, Tracy Hammond, Christine Alvarado, Glenn Evans, and Howie Shrobe (all at the MIT AI Lab) for helpful comments on an earlier draft. Thanks to Bart Anderson at the MIT Brain and Cognitive Sciences Department for his Visual Perception seminar that inspired this work. We also thank the anonymous reviewers for helpful feedback. This research is sponsored by the The MIT Ford Motor Company Collaboration and MIT Project Oxygen.

References

1. Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of CHI '94: Human Factors in Computing Systems*, pages 313–317, Boston, Massachusetts, 1994.
2. Christopher Alhberg and Erik Wistrand. IVEE: An information visualization & exploration environment. In N. Gershon and S. G. Eick, editors, *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '95)*, pages 66–73, Atlanta, Georgia, October 1995.
3. Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps [orig. Semiologie Graphique]*. University of Wisconsin Press, Madison, WI, 1983. Translated by William J. Berg.

4. Stuard K. Card, Jock D. MacKinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, San Francisco, 1999.
5. Stuart K. Card and Jock MacKinlay. The structure of the information visualization design space. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '97)*, pages 92–99, October 1997.
6. Mark Derthick, John Kolojejchick, and Steven F. Roth. An interactive visual query environment for exploring data. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '97)*, pages 189–198, 1997.
7. J. Driver. Motion coherence and conjunction search: Implications for guided search theory. *Perception and Psychophysics*, 51(1):79–85, 1992.
8. J. Driver. Reversing visual search asymmetries with conjunctions of movement and orientation. *Journal of Exp. Psychology: Human Perception and Performance*, 18(1):22–33, 1992.
9. M. D'Zmura. Color in visual search. *Vision Research*, 30(6):951–966, 1990.
10. H. E. Egeth, R. A. Vizri, and H. Garbart. Searching for conjunctively defined targets. *Journal of Exp. Psychology: Human Perception and Performance*, 10:32–39, 1984.
11. Stephen Eick. Graphically displaying text. *Journal of Computational and Graphical Statistics*, 3(2):127–142, June 1994.
12. Jason B. Ellis, Anne Rose, and Catherine Plaisant. Putting visualization to work: ProgramFinder for youth placement. In *Proceedings of CHI '97: Human Factors in Computing Systems*, pages 502–509, Atlanta, Georgia, 1997.
13. Nancy Green, Stefan Kerpedjiev, and Steven F Roth. Generating visual arguments: a media-independent approach. In *AAAI98 Workshop on Representations for Multi-modal Human-Computer Interaction*, July 1998.
14. Harry Hocheiser. Browsers with changing parts: A catalog explorer for Philip Glass' Website. In *Proceedings of ACM Designing Interactive Systems Conference*, pages 105–115, Brooklyn, New York, 2000.
15. P. McLeod, J. Driver, and J. Crisp. Visual search for conjunctions of movement and form is parallel. *Nature*, 332:154–155, 1988.
16. P. McLeod, J. Driver, Z. Dienes, and J. Crisp. Filtering by movement in visual search. *Journal of Exp. Psychology: Human Perception and Performance*, 17(1):55–64, 1991.
17. Ted Mihalisin, John Timlin, and John Schwegler. Visualizing multivariate functions, data, and distributions. *IEEE Computer Graphics and Applications*, 11(3):28–35, 1991.
18. K. Nakayama and G. H. Silverman. Serial and parallel processing of visual feature conjunctions. *Nature*, 320:264–265, 1986.
19. Stephen E. Palmer. *Vision Science: Photons to Phenomenology.* The MIT Press, Cambridge, MA, 1999.

20. Peter Pirolli, Stuart K. Card, and Mija M. Van Der Wege. Visual information foraging in a Focus+Context visualization. In *Proceedings of CHI 2001*, Seattle, Washington, 2001.
21. Ruth Rosenholtz. A simple saliency model predicts a number of motion popout phenomena. *Vision Research*, 39:3157–3163, 1999.
22. H. S. Smallman and R. M. Boynton. Segregation of basic color in information displays. *Journal of the Optical Society of America A*, 7(10):1985–1994, 1990.
23. Robert Spence. *Information Visualization.* ACM Press/Addison Wesley, 2000.
24. Spotfire. `http://www.spotfire.com/`.
25. Anne Treisman and Garry Gelade. A feature-integration theory of attention. *Cognitive Psychology*, 12:97–136, 1980.
26. Edward R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, 1990.
27. Jeremy M. Wolfe. Visual search. In H. Pashler, editor, *Attention.* University College London Press, London, 1996.

Collaboration

Proceedings of the 2nd International
Workshop on Self-Adaptive Software,
Budapest, Hungary, 2001, to appear.

Design Principles for Resource Management Systems for Intelligent Spaces

Krzysztof Gajos, Luke Weisman and Howard Shrobe*

MIT Artificial Intelligence Laboratory
{`kgajos, luke, hes`}`@ai.mit.edu`

Abstract. The idea of ubiquitous computing and smart environments is no longer a dream and has long become a serious area of research and soon this technology will start entering our every day lives. There are two major obstacles that prevent this technology from spreading. First, different smart spaces are equipped with very different kinds of devices (e.g. a projector vs. a computer monitor, vs. a TV set). Second, multiple applications running in a space at the same time inevitably contend for those devices and other scarce resources. The underlying software in a smart space needs to provide tools for self-adaptivity in that it shields the rest of the software from the physical constraints of the space, and that it dynamically adjusts the allocation of scarce resources as the number and priorities of active tasks change.
We argue that a resource manager can provide the necessary functionality. This paper presents a set of guiding principles for building high-level resource management tools for smart spaces. We present conclusions we arrived at after two years of exploring the topic in the Intelligent Room Project at the MIT AI Lab. The paper is based on a number of implemented and tested tools.

1 Introduction

For several years, our research group in the MIT AI Lab has been developing an "Intelligent Room" [8, 9, 5], a space that interacts with

---

* This work was supported by Acer Inc., Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under the MIT Project Oxygen partnership, and by DARPA through the Office of Naval Research under contract number N66001-99-2-891702.

its users through sensory technologies such as machine vision, speech recognition and natural language understanding. Our room also is equipped with a rich array of multi-media technologies. These technologies are intended to provide a natural, human-centered interface to its users.

The Intelligent Room is designed to be a utility that must always be available and it must provide reasonable services to its users even though their needs are not easily predicted. It must continue to provide these services even if there are equipment failures or if there is contention for the use of resources among the users or applications. It is also desirable that it be able to provide improved and additional services if higher quality equipment is added.

Finally, and most crucially to be truly human-centered it must be able to do all these things seemlessly while running, without intervention by programmers and systems wizards. In other words, the Intelligent Room must be a self-adaptive system in the spirit of [17, 16]. It must monitor the environment as well as its own state, have a variety of techniques for accomplishing its goals, and make intelligent choices about which technique to use in the current context.

This paper describes our experience with building such a system. The key insights are:

1. People should interact with the Intelligent Room not in terms of resources, but rather in terms of abstract services (e.g. "show me this information" rather than "print this on that printer")
2. The Intelligent Room should be capable of mapping a service request to a variety of solutions ("project the information," "display it on a PDA," "print it on a printer")
3. The Intelligent Room should choose a solution based both on how well the solution meets the users' needs and how well it minimizes the use of costly or rare resources and
4. It should make this decision at runtime so that it can respond to a changing set of requests and a changing environment.

1.1 What is a resource manager for a smart space

What we mean by a resource manager is a system capable of performing two fundamental tasks: *resource mapping* and *arbitration* .

By resource mapping (a.k.a. match-making) we mean the process of finding out what actual resources can be taken into consideration given a specific request.

By arbitration we mean a process of making sure that, at a minimum, resources are not being used beyond their capacities. At best, arbitration ensures–via appropriate allocation of resources to requests–optimal, or nearly optimal, use of scarce resources.

This paper is concerned with management of higher-level resources. While OS level management (memory, CPU time, etc.) is of course important, and load-balancing of computationally intensive agents over multiple machines is also, we limit our focus to higher-level resources such as physical devices and large software components (see [28] and [22] for an example of a system that deals with resources in a smart spaces at the OS level). Our concerns lie with, for example, projectors, multiplexors, wires, displays, modems, user attention, software programs, screen real estate, sound input and output devices, CD players, drapes, and lamps.

1.2 Some definitions

For clarity, we define here some potentially ambiguous terms.

Metaglue Metaglue [10, 21, 26] is the multi-agent system forming the software base for all work at the Intelligent Room Project. Metaglue manages agent-to-agent communication via Java's RMI system. Agents can start and obtain references to other agents via a `reliesOn` method. All agents have unique IDs; part of an ID is the "occupation" which is the top-level interface the agent implements. Agents are also collected in societies so multiple users and spaces can have distinct name-spaces. Metaglue makes it easy to coordinate the startup and running of agents on any number of machines with differing operating systems.

Agent Agents are distinct object instances capable of providing services and making requests of the resource manager. This means agents themselves are considered to be a type of resource (see below) because they provide services.

Device A physical or logical device is something akin to a projector, screen, or user-attention; devices are often, but not necessarily represented by agents. Devices provide services and so are resources.

Service Services are provided by agents and devices; a single agent or device can provide more than one service and any kind of service can be provided by a number of agents or devices. This is explained in more detail in Section 4.1.

Resource A resource is a provider of a service. Both agents and physical devices are resources. For example, a physical LED sign is a resource (providing the LED sign hardware service) obtained and used by the `LEDSignText` Agent which is in turn a resource (providing `TextOuput` service and `LEDSign` service) that can be obtained and used by any other agent needing such a service.

2 Our work to date

This paper is based on our work on the Intelligent Room Project [8, 9, 5] at the MIT AI Lab, including otherwise unpublished research on resource management. Below we summarize our results relevant to this paper in order to give the reader a better idea of how we arrived at our observations.

Over the past two years we have developed several resource management tools for the Intelligent Room. The tools differed in approach and level of sophistication. At the two extremes we have Namer and Rascal [15]. Namer only does context-based name resolution (i.e. some service mapping but no arbitration). Rascal, on the other hand, is a very complex system that uses a rule-based language (JESS, [13]) for representing knowledge about agents' services and needs, as well as for service mapping, and uses a constraint satisfaction engine (JSolver, [7]) for arbitration. All of our resource managers implement a common interface which allows us to interchange them without changing any of the other code in the system. The reason for having several different resource management schemes was motivated by more than just the need to find the right solution: it is our assumption that different resource mangers will be used in people's various mobile personal spaces (with one or two devices and where computation is scarce) and large and well-equipped static spaces.

As mentioned before, Rascal is our most complex resource manager, conforming to most of the design principles laid out in this paper. Currently Rascal does not deal with issues of privacy and access control and we have only just began work on cooperation mechanisms.

Rascal relies on agents having external descriptions of themselves. Such descriptions include a list of startup needs, a list of provided services (each with a list of its own needs) and descriptions of all possible requests for resources the agent may make in its life-cycle.

Knowing startup needs and needs for providing particular services allows Rascal to ensure that before it assigns a particular agent to provide a service in response to a request, all of the needs of that new agent (and its underlings) can be satisfied. For example, if some agent requests a TextOutput device and the possible candidates are SpeechTextOutput and GuiTextOutput, Rascal will ensure that either speech generation is available for SpeechTextOutput or a computer screen is available for GuiTextOutput, before assigning either candidate to the requester. Additionally, knowing agent's startup needs also allows us to dynamically choose what particular machine an agent should be started on.

So far our software has been installed in six spaces of four different kinds: three small offices, one small living room, one large office (also used for small meetings), and a twelve-seat conference room. The instrumentation of these spaces varies widely; we have a single dilapidated projector and a couple of lights in one of the small offices on one hand, and six projectors and a large number of A/V devices in the conference room on the other. Our living room has two projectors, a TV, several cameras, and A/V equipment.

3 On-demand agent startup - reasoning about absent agents

An agent system in a smart space should have a way of autonomously starting agents on-demand and consequently the resource manager should be able to reason about agents that are not alive right now but could be brought to life if needed.

Because on-demand agent startup is one of the basic features of Metaglue, we have taken it for granted but many other agent systems do not support it. Hence we will now briefly argue why on-demand agent startup is a desirable feature of an agent system in charge of a smart space and then discuss the consequences for resource management.

3.1 Why support on-demand agent startup in smart spaces

Most agent systems deal with very dynamic, spontaneously created and often unstable collections of agents. Therefore, creators of such systems have to refrain from making assumptions about what is available in the system at any given time and usually have to resort to dynamic discovery, direct negotiation or other such techniques when an agent looks for a service or resource (e.g. [18, 23, 12]). This general attitude has been assumed by creators of agent systems controlling smart spaces. Standard Jini [2] implementation and Hive [20] are good examples of such systems.

Smart spaces, by the virtue of being based on stable physical environments, impose a special set of constraints on the underlying software infrastructure. It is true that a lot of adaptivity is still needed – new components can appear and disappear, people come and go, devices are brought in and removed–but at the same time we benefit from assuming certain level of persistence.

It is a feature of a physical space that most of its components are static in the sense that they are usually there. If one day a space contains lights, A/V equipment, projectors, and telephones, it is reasonable to expect that those devices would be present the next day as well. They will be there whether we are using them or not. This level of predictability can (and should) be reflected by the underlying software infrastructure. This is not to say, of course, that the software should not be capable of dynamically accepting new components.

On-demand agent startup is highly useful in any flexible space in a variety of ways. For example, it allows us to make multiple instances of an agent when we want to perform several versions of the same task. Furthermore, it allows us to have very complex interrelationships between agents and very large numbers of agents. Without on-demand startup one needs to craft elaborate startup scripts or hand-start all the agents in the system; both of these are infeasible when talking about collections of forty agents or more, especially when considering that the particular agents change depending on who is starting the system, the various tasks the system is to accomplish, and the room the system is being started in.

With on-demand startup, starting a single high level agent is sufficient to obtain a service provided by that agent. This agent will then request and, cause to be started, all other agents it needs in order to do its job well.

Even with the convenience argument set aside, the following example illustrates some additional benefits of being able to start agents dynamically.

Example 1. Let us assume that the phone service is provided by the phone agent. The agent needs a computer with a voice modem hooked up to a phone line in order to provide its services. Imagine a system consisting of several machines with voice modems hooked up to a single phone line (e.g. in a shared graduate student office). If we did not allow for on-demand agent startup, we would have to do one of the following:

1. Start the phone agent on a prespecified machine, running a risk that if that machine goes down the service is no longer available.
2. Start an instance of the phone agent on every machine with a voice modem and a connection – a rather misleading solution because each of the agents would be advertising phone service but only one of them would be able to provide it at a time because all of the machines share a single phone line.

Another immediate use for automatic agent startup has to do with robustness and recovery: if an agent providing a computational service goes down because of computer failure, it can be automatically restarted at a new location.

We understand that this point has much potential for debate, and so we will not dwell it as other aspects for and against it lie outside the realm of resource management.

3.2 Impact on resource management

If we assume that on-demand agent startup is supported by the underlying software architecture, then it stands to reason that the resource manager for such a system has to be able to reason about absent agents.

To the best of our knowledge, it is uncommon in agent architectures, even those in charge of smart spaces, to have non-alive agents be taken into account during any coordination efforts. It is our belief that taking potentially available agents into account allows a resource management system to make intelligent decisions about resource allocation as in Example 1 in the previous section. (See also Example 2 in Section 4.3.)

An important consequence of embracing on-demand agent startup is that we cannot rely on agents themselves to provide descriptions of their needs and services as they might not be running. The resource manager has to have access to such descriptions without having to instantiate any of the agents. Rascal requires agent programmers to create separate description files but other solutions could easily be created (e.g. descriptions could be cached by the resource manager).

Implicit in Example 1 in the previous section is the assumption that the system, and in particular the resource manager, has a way of starting agents on a specific computer or virtual machine. Metaglue provides such capability as one of its two main primitives. It is unclear to us at the moment to what extent other systems support it.

4 Representation

In this section we concentrate on what knowledge should be contained in the resource manager but not on how that knowledge should be encoded. In particular we argue that when building a resource manager for a smart space, the following key points should be observed:

- Represent resources in terms of the services they provide (e.g. text output) as well as their type (e.g. scrolling LED sign).

- Ensure that representations are rich enough to allow the requesters to get *the best* tools for the job. In particular we caution those using Java against using only interface names for describing resources.
- Ensure that the representation is capable of describing resources that are not represented within the agent systems by agents or other special proxy objects. Examples of such resources would be hardware that is not directly controlled by the agent system but yet is crucial for system's performance (e.g. wires, low level computer components such as modems, third party software modules, etc.)

4.1 Services not devices

To be truly useful, smart spaces have to be affordable, which implies that it should be possible to build them out of mass produced, interconnected components. This includes both the hardware and the software. Hence we can imagine that in the future we will be getting packaged software for our rooms and offices just as today we get it for our desktop computers. Creating such programs, however, may prove very difficult.

It is already difficult to keep desktop computers similar enough to make it possible for the same software to run on all of them. It will certainly be even more difficult when it comes to smart spaces. People take great pride in how they arrange their work and living environments and so creators of software for smart spaces cannot impose how those spaces should be arranged or equipped. While software creators for desktop computers can require that a computer should be equipped with a display, a CD-ROM and a sound card, they certainly cannot require the same level of uniformity among smart spaces. Thus we have to make it possible for applications to run in a variety of spaces with diverse devices and configurations.

The differences among desktop computers have been minimized by the use of software drivers for various devices installed in those computers. Hence, it does not matter what kind of a video card or a monitor one has - the drivers are going to make all cards and monitors "speak the same language" and provide the same services to all applications.

In intelligent spaces the situation will be even more difficult: not only will spaces have different kinds of displays, ranging from little TVs to large plasma displays, but some spaces may not have displays at all. Thus we have to express the abilities of various devices in smart environments in more abstract terms. As well as providing uniform interfaces to devices, as is done on desktop computers, we propose providing uniform interfaces to the services provided by those devices. This distinction is more profound than it may at first appear. It comes from the fact that each service can, in principle, be provided by a number of conceptually different devices and each device can provide a number of distinct services. For example, on one hand, the "short-text-output" service may be rendered by a computer display device, a speech output device or by a one line scrolling-LED display. On the other hand, the LED sign, as well as providing short text display, can provide simple graphics and animation.

We are not unique in suggesting that devices represented by device drivers are insufficient for a smart space; a somewhat different approach was suggested by Winograd [27]. Schubiger-Banz et al. [25] argue for "addressing by concept" in all ubiquitous computing environments (both spaces and/or collections of mobile devices). INS [1] uses "intentional names" for all networked resources. EasyLiving [6] also seems to represent resources in terms of services they provide.

4.2 Rich representation - rich requests

We now examine how the services should be represented by the resource manager. The details are, of course, dependent on the particular implementation.

Open Agent Architecture OAA [19], which relies on a facilitator agent for all inter-agent communication and task brokerage, uses a PROLOG-based ICL (Interagent Communication Language) for describing agents' needs and capabilities. The language allows service providers to describe the agents in terms of tasks they can perform and not really in terms of resources they represent.

Decker [11] uses KQML for communicating needs and abilities of agents.

A common tendency among Java-based systems (e.g. Jini [2], Hive [20], Rascal [15, 14]), is to use the name of the interface (or interfaces) that the resource implements, and a list of attribute-value pairs for describing agents' capabilities.

In this last case, an agent's interface provides information on how the agent's capabilities should be invoked. It also often provides most of the information on what the agent does. One of the advantages of using interface names for describing agents is that interface "ontologies", i.e. APIs, are easily understood by programmers and some of them get adopted by large communities. But it has to be stressed again, that the interfaces should provide access to agent's services. Thus an agent can advertise a number of interfaces, one for each services it provides.

We agree with designers of Hive and Jini in that the types of interfaces implemented by an agent provide a lot of valuable information about agent's capabilities and expected behavior. We also agree with them in that the interface names are not sufficient for describing any agent fully.

Just having interfaces and nothing else be an agent's description is not enough. Often a number of parameters, some of them continuous, contribute to a service's full description. Display services need to be described in terms of resolution, size, color depth, brightness, etc. Having detailed descriptions of services allows for more precise requests: an agent that needs to show a map with a lot of detail, will request a high-resolution color display, not just a display. At the same time, a mail alert agent could deal with a very low resolution display as long as it is visible and so would ask for the display without additional parameters.

4.3 Abstract resources

One important feature that distinguishes multi agent systems in charge of smart spaces from other multi agent systems is that they reside on the frontier between the physical and computational worlds. To function well, those systems have to not only accept but also embrace the physical world around them (we refer to this point again in Section 8).

As a consequence of this, it becomes necessary for the system to explicitly describe not only the services provided by its agents but also those provided by physical hardware and non-agent software present on available computers.

A common approach to this problem is to add agents to represent all needed physical and computational capabilities of the host environment. Hive, for example, uses "shadows" to represent physical devices accessible on or from particular computers. Metaglue has agents that represent individual devices. But how do we know where to start those shadows or agents? An unsatisfactory way is when startup has to be done by a human or by a script leaving the system with no way of reasoning about it or taking action on its own. In case of Metaglue, the device-controlling agents upon startup retrieve the name of a computer they should tie themselves to. In our view, the agents that directly interact with hardware or other software should be able to start dynamically (see Section 3) and dynamically find the computers with all necessary equipment and software.

Example 2. Currently in our system, the main way of providing the speech-input service is with personal wireless microphones connected to computers running third party speech recognition software. In our conference room we have several computers with the right software, several microphones, and an audio mixer that allows us to route microphone signal to any of the computers.

When any of our agents requests speech-input service, Rascal, our resource manager, checks the description of our speech input agent for all of the services that it will need to provide the service. Those will include a computer with a speech recognition engine, a microphone, and a connection between the two. Neither the speech recognition engine nor the microphones have software proxies in our agent systems yet the resource manager is able to reason about them. Rascal ensures that the speech-input agent starts on a computer with the right speech recognition engine and will award a microphone that is not being used for other tasks (e.g. teleconferencing) to the agent, and will ensure that there exists a connection between the two (see Section 8 for discussion of connections).

5 Arbitration

At the heart of resource management is arbitration. By our definition of a resource manager, when two or more agents vie for the same limited resource, the resource manager has to evaluate which gets what.

In this section we argue that arbitration is essential in any larger system embedded in a smart space because it allows individual agents and applications to be written without having to take other agents' and applications' needs into considerations. It also provides for the most basic (but not the simplest) apparently smart behavior of a space. Some arbitration schemes applicable in open agent systems, such as marked-based resource allocation, will prove less effective. Cost-benefit based on self-reported needs and preferences has proven a good solution especially when combined with access control (which limits requests by untrusted and potentially malicious or non-conforming agents).

In addition, in cases where a resource needs to be taken away from a requester to satisfy a new, more urgent, request, every effort should be made to find a replacement for the withdrawn resource.

5.1 Why arbitrate

Arbitration allows for easier implementation of individual agents: the agent writer can view the world more selfishly than if there was no arbitration mechanism. With arbitration in place, the agent programmer can be sure that if any other agent needs resources more, the system will take care of necessary re-allocations (just like in properly multitasking operating systems programmers do not need to worry about yielding to other processes).

Another (obvious) benefit of arbitration is achieving apparent "intelligent" behavior of a space. Just like animals are expected to use their body parts intentionally and in a coordinated fashion, we also expect computer-steered spaces to be "aware" of the interface devices.

5.2 How to arbitrate

The simplest way of resolving ties among requests is to award a resource to the most recent request. For many reasons this may prove to be insufficient. For example it would not be desirable for a new email notification to take over a screen during a video conference with one's boss. Hence there exists need for some analysis before allocating resources. Rascal, for example, uses a simple cost-benefit analysis (details in [15, 14]) to decide who should be awarded a particular service. This scheme relies on agents accurately and honestly reporting how urgently they need a resource. This approach is potentially problematic in that it allows for malicious or inaccuarate representation of one's needs.

A more natural and simple approach to arbitration in potentially open systems in smart spaces seems to be one in which some access control mechanism is used in conjunction with some priority-based scheme. In such a situation the access control mechanism would weed out requests from untrusted and unauthorized agents and then a priority mechanism would decide which of the trusted and authorized requesters should get what resources. In a model where agents can act on behalf of spaces or people, the role-based access control model [24] seems a viable option. We discuss the need for access control further in Section 10.1.

Other approaches had been developed with open systems in mind, notably some based on market mechanisms [3, 4]. Those approaches require existence of a central "bank" and some sort of currency. Such approaches, in their natural form, are not well suited for smart environments. It should not be possible, for example, for someone thousands of miles away to buy control of the room with their extra virtual currency.

Because resource managers can take resources away from requesters, it is reasonable for a requester to keep a resource even after finishing a task if it expects it may need the resource again in near future. For example, a email notification agent may want to keep its output channel as it is desirable for the sake of consistency in space's behavior for those notifications to come through the same channel unless there is a good reason to change.

5.3 Arbitration should allow for clever re-allocations

Consider the following scenario: a space is equipped with a TV set, LCD projector, VCR, video mux, and some computers. The VCR (that also acts as a TV receiver) can be connected to either the projector or to the TV set through the video mux. The computer, however, can only be connected to the projector.

The user is watching the news on the projector, this being the best resource to satisfy a request for a large display. Then the user hears some really important news and decides to share it with a friend while watching the rest of the newscast and so she requests her email agent. With our simpler resource management schemes in place, the projector would be taken away from the news and allocated to the email agent. The more desirable behavior, in this situation, would be for the newscast to be moved over to the TV set and the email to be then displayed on the projector.

The point here is that in many cases the only way to accommodate a new request is to take a resource away from one of the currently active requests. The disturbance can often be minimized, however, by reallocating the old request to a different service. The insight here is that the sets of services that can satisfy various requests overlap only partially and the relationships are often more complex than just proper inclusion (see Figure 1). The reason for it is two-fold: first, different kinds of devices can provide different sets of services; second, physical connections for different kinds of signals are routed differently (so, for example, in one of our spaces the video signal goes through a multiplexer and thus can be connected to either of the projectors or to a TV set, while VGA connections are hard wired).

Allowing for re-allocations makes arbitration among requests much more complex: whenever a resource manager receives a request for a resource, it has to look for a solution that satisfies not only the new request but all of the old ones as well (as far as possible). In other words, a simple task of selecting the best resource for a request turns into a global constraint satisfaction problem.

One point to keep in mind, of course, is that re-allocations are costly. If we move the newscast from a projector to a TV,the user is bound to find it distracting. In some cases disturbance will be minimal, for example a mail notification agent will not mind a re-allocation if it

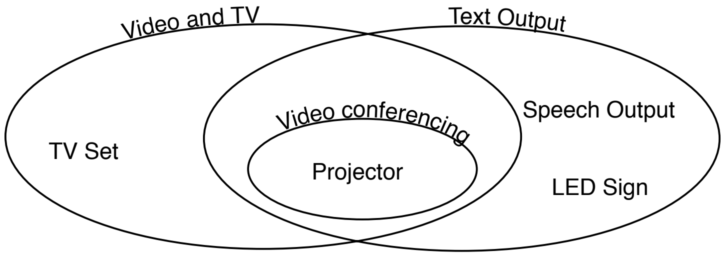


Fig. 1. Different kinds of tasks can be performed with different but overlapping possible sets of devices: both the TV set and the projector can be used for watching videos or tv while only the projector can be used for teleconferencing. At the same time, the projector, the LED sign or the speech output can be used for text output.

happens between notifications—the next time it will simply use a different output device. It's more serious in case of agents that may have to rebuild a lot of their state after re-allocation. Some examples of this would be a camera which was carefully focused on a face or area of the room, or an Internet browser with all of its browser history.

Reasoning about the cost of a re-allocation has to be a part of the overall arbitration process. In Rascal, requesters can specify how costly a re-allocation would be to them and the cost can vary between zero and the cost of taking the service away altogether. The cost of a re-allocation in Rascal has two components: the fixed cost specified by the requester and the difference in utility between the new request and the old request (with the stipulation that it cannot be smaller than zero).

6 Ownership of resources over time (resources vs. tasks)

In this section we argue that many of the services provided by agents in a smart space (e.g. display service provided by a projector) are not tasks and therefore they should be managed differently from tasks. In particular requesters should be given ownership of resources over periods of time. Agents need to own their resources as they are often engaged in long-term jobs that can be changed or modified. We discuss all of this by comparing what we mean by resource management with task management performed by the faciliator agent in the Open Agent Architecture (OAA) [19].

Open Agent Architecture (OAA) is a good example of an agent system that could control a smart space and that also has a complex inter-agent facilitation scheme. What needs stressing, however, is that the OAA “facilitator agent” actually performs *task* management and not *resource* management. That is, the facilitator agent will break down a task into simpler sub tasks and allocate those to individual agents who can fulfill them best. It will not, however, ensure that all of the resources needed for the tasks are available and not in use by other agents. Hence OAA is well suited for a task like sending the current Boston weather report to all of requester’s friends. The task will be broken into components, appropriate information obtained and message sent. OAA is not well suited for tasks that cannot be thought of as point-like in time. Implicit in the OAA model is the assumption that agents can never conflict over the use of scarce resources. Task management is, of course, very important but in a system that controls a physical space with a large number of scarce resources task management should work hand in hand with a resource manager.

It is more natural to think of many agents as having a life cycle, and going independently about their own long-term jobs. For example, an agent listening to and recording conversations in the room in order to be able to bring back audio snippets via keyword searching, needs resources over an extended period of time to complete its job. Showing a movie can be thought of as a task but it can be interrupted, modified, or abandoned in the middle. It can also prevent other agents from using a display for their jobs. In that sense, showing a movie is different from the OAA view of a task.

7 Third party resource request annotation

Once the core resource management system is in place, it should be easy to write modules that do specific types of reasoning and then use that reasoning to annotate requests to limit or reassess possible matches. Often knowledge of the world has direct impact on the appropriate nature of a specific resource to a specific request–this knowledge being outside either the resource manager’s realm of expertise or the requesting agent’s knowledge–and it is imparative that it be easy to have a third party entity contribute such knowledge.

For example, say a user is in a particular room and desires to play a song via his SongPlayer agent. The user would start his agent that would then ask for and locate the bits of the given file, and then attempt to gain access to another agent which would play the actual file. This, of course, would be a resource request for an agent with the ability to play sound.

However, there is also another criterion to the desired agent: location. If the user has been wandering from room to room, it is important that the sound playing agent used be in the room the user is in. This is knowledge that needs to be appended to the request, but neither the resource manager nor the song requesting agent would appropriately have this knowledge.

It would be a violation of normal notions of modularity if the SongPlayer agent had to check the user's location and annotate its resource request. It also seems unwieldy for the resource manager to be responsible for finding and maintaining this knowledge; certainly if this knowledge were in the resource manager's domain, then much other knowledge would be as well. Furthermore, the nature of a flexible agent system is knowledge itself is unlikely to be codified in a universal standard, and so the resource manager would be responsible for translating the output of various other agents into proper resource request annotations. Solving this problem is definitely an active area of research, but in this case it make for a massively large and unwieldy project in the writing of the resource manager.

The best solution we found is to have third party agents that extend the functionality of the resource manager. Authors write agents or functions which pattern match on resource requests and add then additional criterion to those requests as appropriate. In the example above, a distinct other agent which tracks the user eavesdrops on all resource requests and annotates any relevant ones to only consider physically local possibilities.

Request annotations should be able to happen in two ways. The first is modifying the request before a list of possible matches is generated. The second method is filtering the possible matches at the tail-end of the process, after the list of possible resources has been generated. Regardless of method, third party annotators allow for a real componotization of the agents; without them either one or the other agent on any given transaction needs to know too much about the significance of the job at hand. The idea is to have dumb objects wired together smartly to get thinking results, not to have heavyweight objects that are hard to write or maintain.

A further advantage of third party annotators is being able to provide the room with a way of dynamically adapting to equipment failures by writing modules that extended reasoning about certain particular resource allocation problems. For example, if we had an agent that could tell if a projector was broken by looking at the screen with a stearable camera, we could easily have an agent update the resource manager so all resource requests for projectors automatically remove that projector from consideration.

Furthermore, having the ability to have third party annotators should, we hope, serve nicely in the future when contemplating adding large features to the system such as access control (see Section 10.1). Once the model of requesting resources and receiving them is established, pretty much anything can be thought of as modifying or changing the appropriateness of a given resource to a given request–namely annotating a preexisting request.

8 Connections

One style of resource that deserves special attention are connections. Connections are a vital piece of the background of a smart space, and a system with a resource manager that fails to manage them is bound to end up in serious trouble.

The way our room is wired, we have several muxes and switches allowing for information to flow from source devices (cameras, VCRs, microphones) to output devices (projectors, TV sets, modems). Computers are also integrated into this web as either sources or sinks. We also have some trunk wires connecting muxes to muxes, for example, which can only carry one signal at a time. This, of course, is a limited resource. We are a long way from the time when the optimal carrier of all information signals (audio, video, etc.) is the same Ethernet, and until then we need to take into account the specialized wires in an intelligent space. This often means we do not have a fully connected graph of signal sources and sinks, and so the physical connections themselves are a limited resource that needs management.

Due to this, we enter all our connections into the manager as "connection resources". When an agent requests, say, a VCR and projector combination, they also request the collection of resources consisting of the path of connections leading from the VCR to the projector.

We keep the connection aspects of the system very much behind the scenes as an extension to the resource manager. Just because they are a crucial piece does not mean that they need to be in the forefront of a high-level agent programmer's attention. Agents can just ask for resources with the caveat that they are connected, and do not look at the resources involved in the connecting itself at all. The connection extension to the resource manager forges the actual path.

9 Special requests

We have discovered a need for a few "special requests" for resources that seem to lie a bit outside the parameters discussed above. Happily, these are extensions of above, and can be added layers on top of the existing system. We will briefly discuss them in the following sub-sections.

9.1 "Screen saver"

Many agents may want to use resources for a low-level background effect if the resources are not being used for something else. For example, the news ticker or weather forecast agent may want to use the LED sign if there is no better use for it.

The "Screen Saver" type of request gets automatically re-filled after the resource is taken away, used, and then released by some other agent. It is a way of the agent saying, in effect, "I want these resources whenever they are free. If you take them away, then give them back when they become free again."

The advantage of this approach is that it prevents a busy wait on the agent's side. Without "Screen Saver" requests, an agent would have to poll the resource manager from when it has lost its resource until it obtains it again.

An alternative solution would be to have blocking requests, which would also work. We have not closely examined this option, however.

9.2 Auto upgrade

When a resource being used by an application is released, it is worth checking to see if other agents would be better served by getting that resource now that it is available. Agents can specially request that they do not mind being switched to a better resource at any time.

9.3 High-urgency short-term loans

Some requests are for more task-oriented reasons. In these cases, a resource may be needed only for a brief moment. For example, an alert agent might briefly need the speakers of the room to inform a room occupant that there is a call waiting. If the occupant was watching a movie, it would be much more smooth if the alert agent could just borrow the audio for a moment and then give it back. Without borrowing, the original agent would have to re-request the lost resources, and again we would have the polling situation described in the previous sub-section 9.1.

Loans, of course, make cost analysis in the resource manager even more difficult and we have found no easy answers as of yet.

10 Future

In this section we talk about two issues in resource management in smart spaces that we have identified as important but have not yet researched in depth.

10.1 Access control

The real world is full of access control mechanisms. In particular, there are many ways in which access to spaces and enclosed equipment is restricted to certain people. The same is true of information. It stands to reason that agents acting on behalf of people should be subject to similar constraints their owners. If Alice does not have a key to Bob's office, then she is probably not supposed to be able to use his VCR either. We can take this parallel a step further and introduce some more interesting problems.

All members of our lab have a right to enter our conference room. They also have the right to control all of the a/v equipment, the lights, etc. To what extent should this right be extended to their electronic proxies? Should people be granted access to the devices when they are not physically present in a space, e.g. while on a trip to a faraway country? Should the access to the devices only be granted to authorized people on the condition that they are physically present in the space?

If we assume that physical presence is required of most people, let us take another scenario into consideration. Our research group has a meeting and one of the members is in a different city and needs to teleconference with us. During the meeting she needs to show us some of her results. Should she then be allowed to control our projectors and our slide show software? Should telepresence be treated equally with physical presence? Should perhaps one of the people physically present at the meeting grant her the permission? If so, who should have the rights to grant permissions to others?

As we said before, we are not clear yet how access control should be performed in a smart space but we are quite certain that the resource manager would have to be a part of the process. After all, it is the resource manager that grants agents access to particular resources. Thus the resource manager needs to be able to find out what resources the requester has rights to.

10.2 Cooperation

As research on smart spaces progresses, it becomes more and more likely that several spaces will be controlled by the same software. A number of people will be moving from one smart space to another and will expect to be able to make various requests in those spaces. They will also expect some of their agents to “follow” them. Building a single resource manager that would manage resources of all the spaces and all the people is clearly impractical. hence, there will have to be a number of resource managers, each representing a particular collection of resources and requesters. Given that spaces may border with each other or be enclosed by one another, and also given that agents acting on behalf of people will need to use resources provided by spaces, it is necessary for resource managers to communicate with one another to perform optimal resource allocation.

11 Contributions

We have outlined a number of issues that we found to be important in the design of high-level resource management systems for smart spaces. Smart spaces are a relatively new research area and few projects have reached a point where resource management would become critical. We believe, however, that all projects will eventually face these problems once their basic infrastructure is in place and multiple, independently developed, applications are being ran in a space at the same time.

12 Acknowledgments

Many people have contributed to the research presented here. In particular Stephen Peters has provided us with a lot of help in implementing and testing some of the experimental resource management software. Stephen is also working on representing multiple users and multiple spaces in our system. Rattapoom Tuchinda is currently researching the issues of privacy and access control in smart spaces.

References

1. W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM*

Symposium on Operating Systems Principles (SOSP), Kiawah Island, SC, December 1999.

2. Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, and Ann Wollrath. *The Jini Specification*. Addison-Wesley, Reading, MA, 1999.
3. Jonathan Bredin, David Kotz, Daniela Rus Rajiv T. Maheswaran, Çagri Imer, and Tamer Basar. A market-based model for resource allocation in agent systems. In Franco Zambonelli, editor, *Coordination of Internet Agents*. Springer-Verlag, 2000.
4. Jonathan Bredin, Rajiv T. Maheswaran, agri Imer, Tamer Basar, David Kotz, and Daniela Rus. A game-theoretic formulation of multi-agent resource allocation. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 349–356, June 2000.
5. Rodney Brooks. The intelligent room project. In *Proceedings of the Second International Cognitive Technology Conference (CT'97)*, Aizu, Japan, August 1997.
6. B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: Technologies for intelligent environments. In *Handheld and Ubiquitous Computing*, September 2000.
7. Hon Wai Chun. Constraint programming in Java with JSolver. In *First International Conference and Exhibition on The Practical Application of Constraint Technologies and Logic Programming*, London, April 1999.
8. Michael Coen. Building brains for rooms: Designing distributed software agents. In *Ninth Conference on Innovative Applications of Artificial Intelligence (IAAA97)*, Providence, RI, 1997.
9. Michael Coen. Design principles for intelligent environments. In *Fifteenth National Conference on Artificial Intelligence (AAAI98)*, Madison, WI, 1998.
10. Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The Metaglue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.
11. Keith Decker, Mike Williamson, and Katia Sycara. Matchmaking and brokering. In *The Second International Conference on Multi-Agent Systems (ICMAS-96)*, 1996.
12. Jacques Ferber. *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Reading, MA, 1999.
13. Ernest J. Friedman-Hill. Jess, the Java Expert System Shell. Technical Report SAND98-8206, Sandia National Laboratories, 1997.
14. Krzysztof Gajos. A knowledge-based resource management system for the intelligent room. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.
15. Krzysztof Gajos. Rascal - a resource manager for multi agent systems in smart spaces. In *Proceedings of CEEMAS 2001*. Springer, 2001. To appear.
16. Robert Laddaga. Active software. In Paul Robertson, Robert Laddaga, and Howard Shrobe, editors, *Self-Adaptive Software*, number 1936 in Lecture Notes in Computer Science. Springer, 2001.

17. Robert Laddaga, Paul Robertson, and Howard Shrobe. Results of the first international workshop on self adaptive. In Paul Robertson, Robert Laddaga, and Howard Shrobe, editors, *Self-Adaptive Software*, number 1936 in Lecture Notes in Computer Science. Springer, 2001.
18. Victor Lesser. Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1:89–111, 1998.
19. David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The Open Agent Architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128, January-March 1999.
20. Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian, and Pattie Maes. Hive: Distributed agents for networking things. In *Proceedings of ASA/MA'99, the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents*, August 1999.
21. Brenton Phillips. Metaglue: A programming language for multi agent systems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.
22. Manuel Roman and Roy H. Campbell. Gaia: Enabling active spaces. In *Proceedings of 9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 2000.
23. Tuomas W. Sandholm. Distributed rational decision making. In Gerhard Weiss, editor, *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999.
24. Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. *IEEE Computer*, 29(2), February 1996.
25. Simon Schubiger, Sergio Maffioletti, Amine Tafat-Bouzid, and Bat Hirsbrunner. Providing service in a changing ubiquitous computing environment. In *Proceedings of the Workshop on Infrastructure for Smart Devices - How to Make Ubiquity an Actuality, HUC 2000*, September 2000.
26. Nimrod Warshawsky. Extending the Metaglue multi agent system. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.
27. Terry Winograd. Interaction spaces for 21st century computing. In John Carroll, editor, *Human-Computer Interaction in the New Millenium*. Addison-Wesley, 2001.
28. Tonomori Yamane. The design and implementation of the 2k resource management system. Master's thesis, University of Illinois, Urbana, Illinois, 2000.

Advanced Information Systems
Engineering, 13th International
Conference (CAiSE2001), Interlaken,
Switzerland, June 2001, in Lecture
Notes in Computer Science, Vol. 2068,
Springer-Verlag, 2001.

Project Oxygen: Pervasive, Human-Centric Computing – An Initial Experience

Larry Rudolph

MIT Laboratory for Computer Science
`rudolph@lcs.mit.edu`

1 Introduction

For the past six months, I have been integrating several experimental, cutting-edge technologies developed by my colleagues at MIT as part of the MIT LCS/AIL Oxygen project. This paper gives a snapshot of this work-in-progress.

Project Oxygen is a collaborative effort involving many research activities throughout the Laboratory for Computer Science (LCS) and the Artificial Intellegence Laboratory (AIL) at the Massachusetts Institute of Technology (MIT). The Oxygen vision is to bring an abundance of computation and communication within easy reach of humans through natural perceptual interfaces of speech and vision so computation blends into peoples' lives enabling them to easily do tasks they want to do – collaborate, access knowledge, automate routine tasks and their environment. In other words, *pervasive, human-centric computing.*

At first blush, this catch-phrase appears vacuous. Today, computers are certainly pervasive; it is likely, at this moment, you are within 100 meters of a computer. Computers are certainly human-centric; what else can they be? On the other hand, computers are not yet as pervasive as is electricity or water. Although computers perform jobs required by humans, they do not feel human-centric – humans must conform to an unnatural way of communicating and interacting with computers. Finally, the tasks described have little to do with computation; computer-mediated functions is a more accurate term but sounds awkward.

The vision and goals of the Oxygen project are described in detail elsewhere [11, 2, 1], the purpose here is to show how many maturing

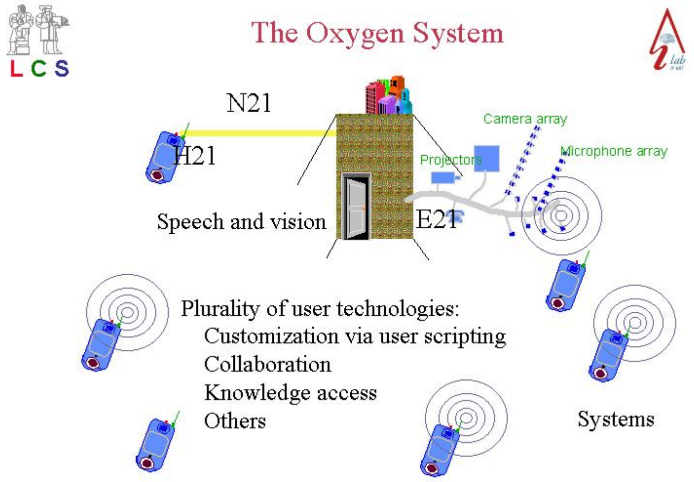


Fig. 1. An overview of the Oxygen Infrastructure, showing the division into three parts: H21, a handheld digital device, N21, the network infrastructure, and E21, the environment infrastructure.

technologies can be integrated as a first step towards achieving the Oxygen vision. There are research efforts at other universities and research institutions that roughly share the same vision, however, each institution focuses on integrating their own maturing technologies. Oxygen has a three-pronged approach by dividing the space into three broad categories: the H21, a hand-held device, the N21, an advanced network, and the E21, a sensor-rich environment (see Figure 1).

In what follows, an Oxygen application is described in terms of its human-centric features as well as the required technologies. It is important to keep in mind that this is just one of many applications and that it is merely a vehicle to explain how many technologies can be integrated and how to create the infrastructure necessary to enable the introduction of context into applications making them more “natural” to use.

The sample application is that of a seminar presentation support system. The next section gives an overview of the application. Section 3, reviews many of the technologies that will go into this application. Section 4, shows how they integrate to form the application. A preliminary programming language and middleware support is described in Section 5.

2 A computer-mediated, seminar presentation system

This section describes a computer-mediated seminar presentation system. As you read through the description, compare it to how presentations are given today. Although a laptop with programs like Powerpoint or Freelance attached to an LCD projector is a vast improvement over the old days of foils or 35mm slides, the human has given up a degree of control, freedom, and naturalness. The system described below provides for a more natural human interface.

Alice is to give a seminar about her $O_{2.5}$ project. As she walks into the seminar room, she allows herself to be be identified as the speaker. She does not need to carry a laptop with her slides on it – all of her files are globally accessible. Alice tells the system how to find her talk by simply supplying enough keywords to uniquely identify the file she wants. Her files are well indexed and so she merely describes the file in human terms and not with some bizarre syntax. The system knows where she is and marshals all the physical components that may be needed for her to control the display.

Alice wants to control the display so that it matches her current desires. A seminar is a live event and the dynamics depend on the audience and speaker. Although it is crucial that she control the presentation, this control should be of minimal distraction. The same is true for the audience – they should be able to see the visual content, hear her commentary, and take notes at the same time. Moreover, unexpected events should be handled in a natural way.

Even today, Oxygen technologies can make a computer-mediated presentation a more natural experience. In particular, three natural ways to control the slides are provided, as opposed to the traditionally way where Alice either clicks the mouse or hits the enter key. It is computer-centric to force the speaker to always walk over to the laptop in order to advance the slide. A wireless mouse is only a partial solution as it requires that something be held in a hand. For Bob this might be fine, but Alice likes to use a laser pointer to highlight objects on the screen and she finds holding two objects to be very awkward. An integrated pointer/mouse is no better since it now requires attention to find the correct button.

Alice can use her laser pointer to highlight words, draw lines and sketches, as well as to switch between slides. Holding the pointer in the bottom right corner means to advance to the next slide. A camera looking at the screen interprets Alice's laser pointer gestures. But not all humans like to use laser pointers. Some people, especially when they are continuously engaged in speaking, like to use verbal commands to control the presentation. This is done with a microphone and software that continuously tries to understand commands. All three modes of control will always be active, allowing the speaker to use whatever is convenient.

There is more to a presentation than just advancing slides. Alice may want to see her notes, see the next slide before the audience, skip a slide or present the slides in a different order. A laptop, handheld, or any other personal communication device can be used by the speaker. To skip to a different slide without anyone knowing it, is a task that is easily performed by simply clicking on a different slide image on her personal display. The personal display must remain consistent with the public display. So, whether Alice says "Next Slide," chooses a slide from her private computer (handheld or laptop), or uses the laser pointer, both displays are updated.

The audience should also have a choice of ways to observe the presentation. They can look at the large projection screen in the front of the room, as is usually the case, or they may choose to view the presentation on their own personal digital device. The output is simultaneously broadcast to these devices. Some people in the audience might like to take notes and have them correlate with the presentation itself. We propose broadcasting a URL or some other identification symbol for the current contents. This can be either used to display the slide on the laptop, or be inserted into their notes. Later on in the privacy of their own room, these notes can be merged with an archived version of the talk. The archived version will match the presentation rather than the original file. Alice may have many "emergency" slides prepared that will be shown only in response to a question.

To summarize, there are several output modalities: the LCD projection, a broadcast of the current content, an archival copy that can be accessed afterwards, and the ability to correlate the public presentation with her own personal view of the presentation.

Lastly, Alice also has "meta" operation control - e.g. switching to a different presentation package, such as a browser or Mathematica, or even to the contents of another presentation. She should also be able to control whether or not content is broadcast or archived.

3 Technology overview

Research into many technologies that support the above scenario being pursued as part of Project Oxygen. Once again, we wish to emphasize that there are many competing technologies being developed elsewhere.

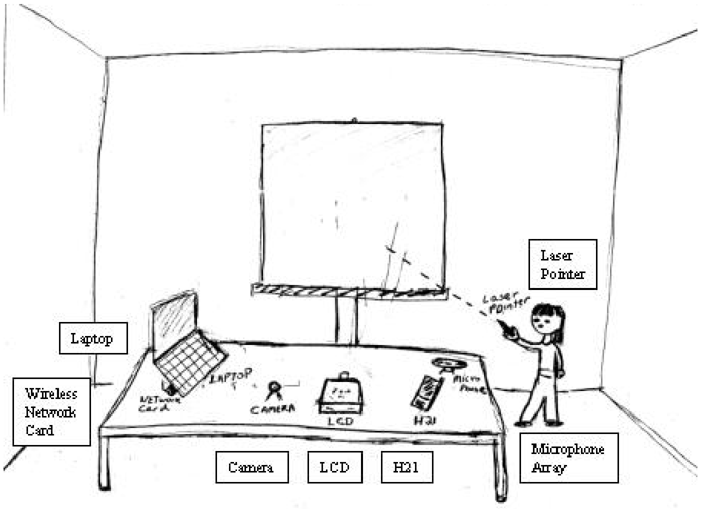


Fig. 2. The seminar room can be assembled from off-the-shelf components. The laptop controls the LCD, camera, microphone, and the networking parts of connecting to the file system, broadcasting, and archiving. The H21 is used by the speaker for personal notes and skipping slides. This application makes use of many of emerging technologies being pursued at the Lab. for CS and AI Lab at MIT.

We deliberately ignore them for several reasons[1]. First, to do justice to them all would make this article too large. Second, close physical proximity is usually required when making use of experimental, research systems. While it is possible to do this remotely for one component, it is nearly impossible do this for a number of research projects. We wish to provide feedback to these other research projects before they are ready for prime-time and we deliberately try to use them in some unintended way. While there are similar efforts in many of the intelligent or instrumented rooms, our example is simply geared towards exposing how components interact even with commodity hardware. As fun as it is, the particular demo of an oxygenated presentation is not the goal.

3.1 The Handy 21 (H21)

Although the commercial sector has been cranking out all kinds of hand-held devices, there is still much research to be done. The H21 should replace the plethora of communication gadgets with a single portable device. In particular, it should combine at least the functions of a cellular phone, wireless Internet connection, pager, radio, as well as a music and video player/recorder. Packing all this functionality into a single device appears to make it too heavy to be portable. So, industry strives to find the right set of combinations and to then sell add-ons to fill-in the missing pieces. The Oxygen approach is different: all that is needed is a minimal set of components built into the hardware with software and reconfigurable hardware used to provide whatever functionality is needed.

The SpectrumWare project [13] is developing a multipurpose communications system that can be programmed to receive and transmit many different types of signals in order to form a "communications chameleon." One can program the H21 to be a radio, cell phone, or television receiver. To fit in a small space it will need configurable hardware.

The RAW project [3] is developing a chip that will deliver unprecedented performance, energy efficiency and cost-effectiveness because of its flexible design, by exposing its wiring to the software system, the chip itself can be customized to suit the needs of whatever application is running on it. The Raw chip could be incorporated into a single de-

---

[1] The author wishes to apologize to all those who do not agree with these reasons. In a future, expanded version of this paper, many competing technologies will be cited. The author would be happy to learn about an relevant research.

vice that could perform a wide variety of applications: encryption or speech recognition, games or communications.

The commercial sector also understands the need for low-power devices, especially handheld ones. However, to make substantial improvements, it is important to re-examine computer architecture from basic principles. The SCALE project [6] is aimed at doing just that.

Rather than waiting for this research to come to fruition, the Oxygen project will make use of commercial handheld computers. In fact, it is doubtful that we will ever build our own device. More likely, we will continue to modify and adapt commercial products that at a minimum, support Linux, audio and visual I/O, and multiple communication channels [4]. Although in an ideal world one will have the right devices for the job, in reality that is usually not the case. It is thus important to be able to make use of what is available. Users want to get the job done and so we expect to support a wide range of devices.

3.2 Networking, naming and location management

One's personal data should be easily and universally accessible. Having a multitude of digital devices, each with some possibly inconsistent set of data, is neither natural nor geared towards the needs of the human. Having to remember a set of arbitrary names just to use a physical device sitting in plain sight is also demeaning to the human user.

The self-certifying file system, SFS [14], is a universal, secure filesystem that has a single global namespace but no centralized control. Other similar filesystems require the users to use a particular key management system to provide security and authentication. SFS separates key management from file system security, thereby allowing the world to share the filesystem no matter how individuals chose to manage their keys.

Within a building it is useful to know where things, including one's self, are physically located. The traditional approach is to have all things periodically broadcast their identity and to have sensors spread throughout the building that detect these things. To provide a degree of privacy, among other reasons, the Cricket [7] location-support system takes the opposite approach. Spread throughout the building are a set of beacons. The beacons are a combination of RF and infrared signals that broadcast physical and logical location information. Things in the environment sense these beacons. Thus, a handheld knows where it is located rather than the system knowing it. A person has the freedom to reveal his or her location – usually when some service or resource is required. All sorts of devices need to be integrated into the system having network connectivity and location awareness [12].

Knowing the location of things enables one to name digital devices by their intended use rather than by some specific URL or IP address. The Intensional Naming System [8] does just that by maintaining a database of currently active devices along with attributes describing their intended use. Devices periodically check-in to avoid having their entries time-out. INS supports early binding for efficiency and late binding for mobility. With INS, it is possible to route packets to the LCD projector that is located in the same room as the speaker or to route messages to whatever display device is near the intended recipient.

3.3 Security and correctness

As evident by the central place of this subsection, the Oxygen Project considers security and privacy as a central component of a human-centric system. We are developing a personal identification device that has two interesting features. It has a very simple interface, perhaps only a single button to distinguish between identification and authorization [12, 19]. The simpler the interface the easier it is to make the device secure. The second feature is that identification mechanisms provide privacy. A guiding philosophy is that privacy is the right to reveal information about oneself. When one chooses to make use of public system resources one is choosing to reveal information about one's self. Various schemes for secure, private group collaboration are being developed [19] as well.

As computers continue their infestation of human activities, their reliability becomes more important. Specifying the behaviors of interacting systems is particularly challenging. Research efforts, I/O Automaton (IOA) [16] and Term Rewriting Systems (TRS) [5], aimed at proving the appropriateness of collective behaviors, have focused on precise and concise specifications.

3.4 Human interfaces

There is no question that verbal and visual interfaces to computers are rapidly maturing and are already being successfully deployed. However, speech and natural language systems need to extend beyond simple dialog systems. The approach is to gather information from the speaker in a number of ways, to fuse this with information from other sources and to carry out tasks in an off-line fashion as well, so as to optimize the users time [22]. This effort is also trying to make it easy to develop domain-specific speech interfaces; moving the creation of a interface from an art to a science.

The focus on visual input system that recognize a range of gestures tries to leverage multiple input devices. Just like stereo vision makes it easier to differentiate objects, collaboration between multiple monitoring devices simplifies many recognition tasks. For example, a microphone array along with a array of cameras can be used to do speech processing of all the conversations going on in a meeting, even when several people talk at once [9]

On the output side, there is research aimed at building very large displays. The challenge is to overcome the bandwidth problem of getting all the pixels out of a computer. The approach is to embed processing, connectivity, and display all in one package so that the pixel generator is close to the pixel display, thereby creating a sufficiently rich environment to mimic the Holodeck of Star Trek fame [18]. A related effort is to develop an economical method for displaying 3D using an auto-stereoscopic display [17].

3.5 Collaboration

There is much to be done in the way of supporting computer-mediated human collaboration. Teleconferencing has made strives in allowing collaboration between people who are widely spatially disjoint, but it is still difficult to collaborate when people are temporally disjoint [21]. Much of this work is going on in the Artificial Intellegence Laboratory at MIT and unfortunately, I only know a little bit about it. The seminar presentation scenario described in this paper is just the beginning.

4 Implementing the seminar presentation system

We can now relate the technologies described in the previous section with the needs of our seminar presentation system. The explanation roughly follows the description in Section 2. We ignore traditional issues like authorization and allocation of resources and application code written in traditional ways.

When Alice enters the seminar room, she must be identified and the presentation manager must be initiated. A simple tag broadcasts her public key to her H21 or, if she does not have one, then to the room computing E21 infrastructure. The H21, with Alice's permission, will initiate the seminar presentation manager application as an extension of Alice's computing environment. Rather than having applications run on machines with only a loose connection to the owner, they are all under direct control of the initiator, who has the ability to interact with them from any Oxygen supported I/O device..

Access to Alice's presentation files is via the secure, global filesystem, SFS. Advanced indexing systems, such as Haystack [15], will be used to find files or slides within a file. The Cricket location management system is being used to know where the presentation is occurring. It is possible for the speaker and the audience to seamlessly move to a larger seminar room without losing any content. The intentional naming system, INS, is used to route packets between the components of the system and provide for fault tolerance and mobility. If one component crashes, INS will help in finding an alternate or reconnect when the component comes back online.

For the input modes, speech and vision processing is used. The speech project, Galaxy, has been developed mostly for dialogs and is being adapted to an active monitoring mode. The vision system [9] is used for the initial laser pointer and later for human gesture recognition. A microphone array combined with a vision system that precisely located the position of the speakers mouth is being developed to allow the speaker more mobility. Recognition of drawing gestures, makes use of technology underlying the Rational Capture project [10].

The presentation itself will be controlled in a conventional manner. For powerpoint presentations running under windows, we use visual basic to connect to the rest of the application middleware as well as to control the presentation itself. For the speaker's note view, a stripped down web-browser is used with the application code written in Java.

The output side, at the moment, is the least sophisticated. We hope to make use of the Auto-stereoscopic Display work that will enable 3D image rendering and the Holodeck research that will support very large active displays. In addition, capturing the experience for later review will make use of the research in collaboration [21].

Finally, the presentation manager application is written in a special "communication oriented" language and middleware, described in the next section. Such communication oriented languages, along with IOA and TRS research will lead to the development of correct distributed systems that work first time out, and allow one to focus on performance as the system scales.

5 The language overview

This section highlights the core of a communication oriented language used to program some Oxygen applications. The work described in this section is preliminary and so the description is deliberately sketchy[2]. Although, Java could be used, especially since most of the underlying technologies provide a Java interface, our language lets the application writer to focus on what is relevant, permits very aggressive optimizations and is more concise and precise.

At a high level of abstraction, there are only a few components that need to be manipulated: nodes, edges, messages, and actions. Nodes are just about anything that can be named and communicated via sockets. An edge is a directed connection between nodes. A message is an entity that flows along an edge. An *event* is the creation or destruction of one of these components; thus there are only six different types of events. Rules (or actions) make up the final component of the language. An *action* consists of a trigger and a consequence. A trigger is an event, such as the creation of a node or the destruction of an edge. A consequence is also an event. For example, the existence of a message on an edge can trigger a set of edges to be disconnected.

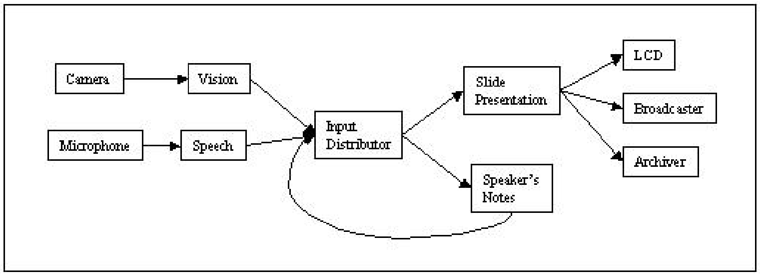


Fig. 3. A graphical view of the components and their connections

Nodes are named in a way that is compatible with the intentional naming system [8] and consists of a collection of key/value pairs. When a node is named, these pairs are matched against a database of existing, functioning devices or services. For simplicity, any node can be created or destroyed. In actuality, it is only the connection that is created or destroyed when the node is a physical device or an enduring software service. Connections are IP/Port specifiers; it is assumed that all devices and third-party services have some kind of wrapper that converts input and output to the appropriate formats. Rather than have a special case to handle the case when a named node does not exist, it is

---

[2] This work is so preliminary that the language has yet to be named.

assumed that such nodes are created and then immediately destroyed. The code to handle a non-existent node is exactly the same as the code to handle a node that was connected but becomes disconnected or destroyed.

Messages are self-describing and self-typed. They can be named, as with nodes, by a set of key/value pairs but must include a location, either a node or an edge. Very large messages or streams, such as audio, video, or screen images are conceptionally the same as text messages, but the implementation treats them different to ensure sufficient performance. As each message moves through the system, it is assumed to be created when it arrives at a location and destroyed when it leaves that location. This permits actions to treat message events and creation/destruction of node and edge events in the same uniform way.

All the action is, of course, with the actions. Actions are simple yet powerful rules. Actions can be created or destroyed, just like all other objects in the language, and are thus events. So, some event trigger can create new actions or remove current ones. Actions are needed to control what happens in a dynamic, sensor rich environment. When one enters a room from the hallway, two events happen: the link to the hallway is destroyed and the link to the room is created. Either of these events can serve as triggers for a whole slew of actions.

Figure 4 shows the specification of the presentation manager. The nodes are named using key/value pairs. The variable *owner* is a parameter of the system and is passed-in when the application begins. The location specifier could be done in the same way, but in the code in the Figure it is hardwired. Nodes, messages, and edges are all named to make it easier to read the code. Two sets of actions are specified. One disconnects the I/O devices. This, presumably is useful for situations in which the speaker wishes to temporarily pause the current presentation and to switch to a different one[3]. The first action is invoked whenever there is a “pause” message on the dialog-in port. The consequence of this action is to destroy the four edges that connect to the camera, microphone, LCD, and broadcast process. These will be used by the other application. The archiver is dedicated to this application and so can remain connected. A second set of actions show another example of disconnecting only the edges to the broadcaster and archiver nodes.

There is a middleware system that executes the language [20]. Initially it executes on a single machine, but soon will be made fault tolerant and decentralized. Nearly all actions performed by the mid-

---

[3] Hopefully, the speaker is not checking her mail during the presentation itself!

Application Name:
Seminar Presentation Manager (owner)

Nodes:
microphone : [“Device”, “microphone”, “Location”,“NE43-518”]
camera : [“Device”, “camera”, “Location”,“NE43-518”]
input : [“Process”, “input-collector”, “OS”,“Unix”,“Owner”, owner]
ppt : [“Process”, “powerpoint-displayer”, “OS”,“Windows”,“Owner”, owner]
ppt’ : [“Process”, “speaker-notes”, “Platform”,“H21”,“Owner”, owner]
lcd : [“Device”, “lcd”, “Location”,“NE43-518”]
broadcaster : [“Process”, “broadcast-slides”,“OS”,“Unix”,“Owner”, owner]
archiver : [“Process”, “archive-slides”, “OS”,“Unix”,“Owner”, owner]

Edges:
m_{pause} : [“Message”, “pause”, “Location”, dialog-in]
m_{resume} : [“Message”, “resume”, “Location”, dialog-in]
$\mathrm{m}_{confidential}$: [“Message”, “confidential”, “Location”, dialog-in]
m_{public} : [“Message”, “public”, “Location”, dialog-in]

Messages:
e_{ms}: (microphone , speech) , e_{si}: (speech , input)
e_{cv}: (camera , vision) , e_{vi}: (vision , input)

e_{ip}: (input , ppt) , $e_{ip'}$: (input , ppt’)
e_{pl}: (ppt , lcd) , e_{pb}: (ppt , broadcaster)
e_{pa}: (ppt , archiver) ,

$e_{p'i}$: (ppt’ , input)

Actions:
(m_{pause} , ($!e_{cv}$, $!e_{ms}$,$!e_{pl}$, $!e_{pb}$))
(m_{resume} , (e_{cv} , e_{ms} ,e_{pl} , e_{pb}))
($\mathrm{m}_{confidential}$, ($!e_{pb}$, $!e_{pa}$))
(m_{public} , (e_{pb} , e_{pa}))

Fig. 4. Part of the communications program that expresses the connections. There are always two implicit nodes: dialog-in and dialog-out. The actions disconnect and reconnect the I/O devices on a pause or resume command. Presumably this is used to switch to another presentation. Similarly, the speaker may want to go “off-the-record” and show slides that are not archived nor broadcast. Going “public” reestablishes these links.

dleware corresponds to an event and events can be triggers for other events.

Although, we expect that the language will be compiled for optimum performance and reliability, it is also possible to interpret commands during run time. A user can modify an application during run-time to adapt to changing needs. The simple structure makes this easy to do provided there is a way for a user to easily name nodes, edges, and messages.

6 Conclusion

Scientific endeavors have always alternated between periods of deep and narrowly focused research activities and periods of synthesis across many fields. I believe we are in the midst of a new computer revolution. The relentless doubling of performance every 18 months, the even faster exponential growth of the web and its communication infrastructure, and the maturing of many human-computer interface technologies means that things will not stay the same. While industry is doing some of this work, the emphasis is on producing products that are good at one thing. Oxygen is not producing products, rather it is exploring what is possible when one synthesizes the fruits of research across many fields.

This paper describes a work in progress. Not only is this system still under development, but many of the technologies that it exploits are also under development. The presentation manager is simply a data point. It gives insight into the tools that will be needed in the future, gives feedback to those researchers developing the components, and is just one of several parallel efforts. These efforts will create the infrastructure necessary for the next decade. There is much to be done but we must keep the goal in sight – computers must become easier and more natural to use.

7 Acknowledgements

The author would like to thank John Ankorn for his help with all aspects of the integration effort as well as with this text. Thanks are also due to Hilla Rogel for her drawings and to Anant Agrawal for involving me in the Oxygen Project. This work is supported in part by our industrial partners,: Acer, Delta, Hewlett Packard, NTT, Nokia, and Philips, as well as by DARPA through the Office of Naval Research contract number N66001-99-2-891702.

References

1. Mit. http://www.oxygen.lcs.mit.edu.
2. Scientific american. August Issue, 1999.
3. A. Agarwal and S. Amarasinghe. RAW. http://www.cag.lcs.mit.edu/raw.
4. J. Ankorn. EMS. http://www.oxygen.lcs.mit.edu/ems.
5. Arvind and L. Rudolph. TRS. http://www.csg.lcs.mit.edu/synthesis.
6. K. Asanovic. SCALE. http://www.cag.lcs.mit.edu/scale.
7. H. Balakrishnan. Cricket. http://nms.lcs.mit.edu/projects/cricket.
8. H. Balakrishnan. INS. http://nms.lcs.mit.edu/projects/ins.
9. T. Darrell. Visual interface. http://www.ai.mit.edu/ trevor/vision-interface-projects.html.
10. R. Davis. Rational capture. http://www.ai.mit.edu/people/davis/davis.html.
11. M. Dertouzos. *The Unfinished Revolution*. Harper-Collins Publishers, New York, 2001.
12. S. Devadas. Aries. http://caa.lcs.mit.edu/caa/projects.html.
13. S. Garland and J. Guttag. Spectrumware. http://nms.lcs.mit.edu/projects/spectrumware.
14. F. Kaashoek. SFS. http://www.fs.net.
15. D. Karger. Haystack. http://haystack.lcs.mit.edu.
16. N. Lynch. Input output automata. http://theory.lcs.mit.edu/tds/i-o-automata.html.
17. L. McMillan. Auto-steroscopic display. http://www.graphics.lcs.mit.edu/ mcmillan.
18. L. McMillan. Image based rendering. http://www.graphics.lcs.mit.edu/ mcmillan/IBRwork/.
19. R. Rivest. CIS. http://theory.lcs.mit.edu/ cis/cis-projects.html.
20. L. Rudolph. Project oxygen. http://www.org.lcs.mit.edu.
21. H. Shrobe. Knowledge-based collaboration. http://kbcw.ai.mit.edu.
22. V. Zue. GALAXY. http://www.sls.lcs.mit.edu/sls.

Author Index

Index