An Idempotent Message Protocol

Jeremy Brown

Abstract

We describe a protocol for reliable (exactly-once) datagram delivery on a wormhole-routed network that discards messages in response to congestion and hardware faults. Our protocol is connectionless – no state needs to be stored for communicating pairs of processes, only for datagrams still being managed. Datagrams are not guaranteed to be delivered in any particular order.

The protocol is simple enough to be implemented directly in hardware.

1 Introduction

[1]

Our protocol is aimed at a massively parallel computer architecture with an integral, unreliable inter-node network. Such a system has several characteristics which have motivated our protocol's design:

Thousands of nodes, millions of wires: By "massively parallel computer" we mean a computer that can scale to, at the very least, several thousand nodes; correspondingly, there will be tens, hundreds, or even thousands of thousands of wires in the inter-node communications network. Guaranteeing the reliability of the processing elements will be a Herculean task, and is the subject of ongoing research both within our research group and elsewhere; guaranteeing the reliability of every network wire will be an impossible task.

An unreliable network with three guarantees: Network protocols designed to guarantee deadlock-free datagram (message) delivery on reliable networks are already quite complex, and thus impose a burden of complexity on the implementation of the routing elements. Combined with the problem of unreliable wires, the overall complexity of guaranteeing reliable datagram delivery on a network for a massively parallel machine – especially with good performance (i.e. low latency) – is a daunting task.

Preferring simplicity, we are instead considering network implementations which do not guarantee the delivery of each message; some mechanism is thus required to provide reliable datagram delivery on the substrate of unreliable messages, and this is the niche our protocol is designed to fill.

Our protocol does depend upon three assumed guarantees of the network:

1. A message, if delivered, is delivered at most once.
2. There exists at least one path with no bad wires between each pair of processors.
3. All messages which are successfully delivered from a node A to a node B are delivered to B in the order in which they were sent from A.

We believe these guarantees are fairly easy to provide in the context of a dedicated network. For example, the first guarantee is provided by a wormhole-routed network which resolves congestion simply by discarding messages randomly. (The implementation of extremely fast router components for such a network is fairly straightforward.) The second and third guarantees are ensured, in this example, by constraining the network topology. In particular, the network provides the second guarantee (to a desired degree of certainty based on the probability of individual wire failures, etc.) by featuring a topology with multiple, randomly-selected paths between each pair of processors, e.g. a FAT tree. The third guarantee arises naturally as long as all paths between a given pair of processors in the network are of the same length, as they would be in a FAT tree.

Since our envisioned target network is, in fact, exactly this network —- a FAT tree featuring wormhole-routing and discard-based congestion resolution — we shall mention one more useful property: because the distance between any pair of processors is fixed (and easily computed), it is relatively simple to decide when to give up on

[1] This memo is incomplete at best; the reader is referred to follow-on memo(s) in 2002. – jhb, Feb. 2002

Project Aries Technical Memo ARIES-TM-014
Artificial Intelligence Laboratory
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA USA

Research performed under
DARPA/AFOSR Contract Number F306029810172

Message Type	Content	Direction
Send(D)	$< UID_d, CONTENT_d >$	$\mathrm{NI}_A \rightarrow \mathrm{NI}_B$
Acknowledge(D)	$< UID_d >$	$\mathrm{NI}_A \leftarrow \mathrm{NI}_B$
Forget(D)	$< UID_d >$	$\mathrm{NI}_A \rightarrow \mathrm{NI}_B$

Table 1: The three messages of the reliable-datagram protocol.

receiving a response to a previously-sent message. This will be useful in setting timeout values.

Unordered datagrams, not streams: Communications between processing nodes will generally be in the form of datagrams (messages), due to either explicit message-passing or references to remote, shared memory. Each processor may, in short order, send datagrams to a great many other processors.

Additionally, multiple datagrams to a single processor may not have an ordering requirement; for instance, a series of memory operations to independent locations need not be ordered under a weak memory model such as release consistency.

Overall, the semantics of streaming communcations – reliable, in-order delivery of a continuous linear sequence of data – are unnecessary for this type of machine, and would add significant complexity to any protocol for reliable inter-node communications.

2 Protocol

We target a conceptual architecture in which each processor P has a network interface NI_p which is responsible for sending and receiving datagrams; it is the responsibility of the NI to implement the reliable-delivery protocol on top of the network's unreliable messages. Obviously the NI might well be implemented as nothing more than a process running on the processor itself, but for conceptual clarity we will refer to it as a separate component in the following discussion.

2.1 Overview

We assign each datagram D an ID so that the datagram is actually a tuple $<$ UID$_D$, CONTENT$_D$ $>$; UID$_D$ is an identifier guaranteed to be unique for all datagrams currently in any stage of being sent, and CONTENT$_D$ is the body of the datagram intended to be delivered to node B. A simple means of generating per-datagram UIDs is for the source node to prepend its node-ID to the value of a counter which is then incremented; the counter need only have enough bits to provide unique IDs for all live datagrams sent from that node.

In order to reliably deliver datagram D from processor A to processor B, a minimum of three messages must be sent between their NIs, and any of these messages may have to be sent more than once in the event that the network discards some due to congestion or bad wires. The messages are tabulated in Table **??**

An informal description of the effects of each message follows:

NI$_A$ must repeatedly send the Send(D) message at intervals until it has received an Acknowledge(D) from NI$_B$, at which time NI$_A$ must send a Forget(D) message to NI$_B$ and may forget all about D. This is safe, as NI$_B$ can not Ack(D) until it has successfully receieved D. If NI$_A$ receives any further acks for D from NI$_B$, they must also be replied to with Forget(D) messages.

On the other side of the network, when NI$_B$ first receives a send from NI$_A$, it must deliver D to processor B, then must commence repeatedly sending Ack(D) messages back to NI$_A$. Because NI$_A$ might send additional Send$D$ messages if it has not yet received an Ack(D) message, NI$_B$ must remember UID$_D$ so that it will not accidentally re-deliver D to processor B.

Because we require of our network that messages are delivered in send-order or dropped – never delivered out-of-order – NI$_B$ can be certain that when it sees a Forget(D) message from NI$_A$, it will never again receieve a Send(D) message from NI$_A$; thus, NI$_B$ must stop sending Ack(D) messages and may forget all about D when it finally receieves a Forget(D) from NI$_A$.