Multicasting Protocols for High-Speed, Wormhole-Routing Local Area Networks

Mario Gerla, Prasasth Palnati, Simon Walton
Computer Science Dept., University of California, Los Angeles.

Abstract

Wormhole routing LANs are emerging as an effective solution for high-bandwidth, low-latency interconnects in distributed computing and cluster computing applications. An important example is the 640 Mb/s crossbar-based Myrinet. A key property of conventional LANs, which is valuable for many distributed applications, is transparent, reliable network-level multicast. It is desirable to retain this property also in wormhole LANs. Unfortunately, efficient, reliable multicasting in wormhole LANs is problematic because of the potential for deadlocks. As a consequence, current multicasting implementations typically consist of repeated unicast or assume a priori buffer reservations. These solutions, however, tend to increase latency and do not scale well.

In this paper we address the problem of providing transparent, reliable, efficient network level multicasting in the wormhole LAN. We describe several protocols for achieving deadlock-free, reliable multicasting using restricted routing and fast buffer reservation techniques. Tradeoffs involving complexity and performance of various solutions are discussed, and are illustrated using simulation. A simple multicast implementation for Myrinet has been carried out, and experimental results are presented.

1 Introduction

Wormhole-routing networks [NM93] are finding increasing application as fast, low-latency interconnects, both for interhost communication over small areas, and intrahost communication for multiprocessor machines. The main characteristics of wormhole networks are wormhole routing [KK79], use of backpressure for flow control, and in most cases source routing. The physical layer of these networks is generally reliable. Network sources can normally assume that if they send out a packet, or 'worm', on the network then it will eventually be received at the destination.

Distributed supercomputing and cluster computing impose certain critical demands upon the network. These demands often include low-latency multicast service. Distributed algorithms which use multicasting normally require it to be reliable. Other multicast applications include Distributed Active Simulation [fST94] and the real-time MBone service.

Ideally, the wormhole LAN performs as a very high-speed Ethernet, namely: low latency (the worm can be injected whenever the interface is free); guaranteed delivery (after the worm has been completely output to the network), and; low cost. In addition, the wormhole net offers the following advantages over Ethernet: better scalability because of mesh topology with multiple simultaneous transmissions; lower access delay variance because blocked worms wait their turn instead of being dropped and retransmitted; fairness due to round-robin scheduling of blocked worms. To achieve high speed and low cost a very simple, streamlined switching fabric is used, with very efficient worm processing (e.g. source routing) and minimal network management facilities.

These advantages, however, do not come for free. Blocking and backpressure upon contention (instead of dropping) is prone to deadlocks. This can be avoided in several ways, as later discussed. Another limitation is multicasting. We require multicasting to be as reliable as it is in an Ethernet, say. That is, once the multicast message has been handed to the network, it can be assumed to be delivered to all the active members in the multicast group. Also, multicast latency must be comparable to unicast latency (as it is in Ethernet). Finally, totally ordered multicast is required for some applications (i.e. all members of the multicast group must receive the multicast messages in the same order).

Reliable multicasting is straightforward in shared medium LANs. It can also be efficiently achieved in store-and-forward LANs such as ATM LANs and router based internets [FJM+95]. Unfortunately, multicasting is much more challenging in wormhole networks because the simplicity of the crossbar switches makes it difficult to support multicast trees in the fabric. Furthermore, the lack of internal store-and-forward buffering coupled with the propagation of the message simultaneously on multiple paths increases the risk of deadlocks.

An alternative to providing multicast in the switching fabric is to handle the replication of multicast messages in the host adapters. These are easier to program, and provide buffering. Myrinet has, in fact, adopted this solution in its current host adapter software, by using repeated transmission of copies of the multicast message from the source to all destinations. This solution is very reliable, but does not scale well: the source interface is tied up during the entire multicast session, leading to large latencies. Furthermore, total ordering cannot be enforced.

Another solution, also based on host adapter multicasting is reported in [VLB96]. A buffer credit scheme which is the extension of the Illinois Fast Messages [PLC95] credit scheme is proposed. Multicast is performed on a precomputed binary tree which spans all the multicast members. The source must first acquire from a centralized credit manager (a designated host adapter card) a cumulative buffer credit for all destinations (this is required to avoid buffer deadlocks). Sequenced credits guarantee total ordering. The centralized manager periodically replenishes the credit pool using a credit gathering token mechanism. The advantages of this centralized credit management scheme are total ordering and feedback congestion control (when congestion builds up, new requests are automatically delayed by the lack of credits). On the negative side, the latency is increased by the credit request mechanism. Buffer resources are not efficiently utilized, due to the latency of the credit gathering token and the credit request protocol. In other words, the time taken to reserve the buffer may exceed by far the actual buffer usage time. Finally, the scheme is vulnerable to credit manager failure.

Reliable multicasting can also be provided at the transport level (i.e., above switch and host interfaces) [FJM+95]. For example, consider the case where the members of the multicast group are arranged in a chain, and the multicast source is at one end of the chain. The message is passed from one host to the next via wormhole routing. If the worm becomes deadlocked and is dropped in the middle of the chain, half of the destinations do not receive it. Assuming multicast messages are sent in sequence, the gap in the sequence alerts some hosts of the loss. Using the request/repair scheme proposed in [FJM+95] one of these hosts will time out first and send a retransmission request up the chain. The first host which gets the request and which received the original message will rebroadcast it downstream. While recognizing that transport level multicasting is a cost effective alternative, in this paper we pursue the goal of providing network level reliable multicasting in the wormhole net. Thus, we will focus only on network level reliable multicasting.

In this paper, we explore both switch fabric level and host adapter based multicast solutions for a wormhole network. We opt for a fully distributed approach, and reduce latency by acquiring buffer resources as we go, rather than making reservations in advance. Our wormhole network model is inspired by Myrinet, although in several cases we assume features which, albeit feasible, are not currently supported in the Myrinet hardware.

In section 2, we describe the Myrinet architecture, to provide a reference for the wormhole protocols. In section 3, we present several proposals for switch fabric multicasting. In section 4, we describe the basic operation of host adapter based multicasting and then present the implementation for the Hamiltonian circuit and the rooted tree in Sections 5 and 6 respectively. Sections 7 and 8 describe simulation and measurements. Section 9 concludes the paper.

2 Myrinet Protocols

Myrinet is a commercial wormhole-routing, high-speed (640 Mb/s) electronic LAN [BCF+95]. The switches are crossbar interconnects. The links have a maximum length of 25 metres. The host interfaces for this network consist of peripheral bus cards containing their own processor, the LANai. The LANai processor is custom designed for interfacing to the Myrinet, and is a 16-bit processor with 32-bit data paths. It provides the line encoding/decoding for the physical links, and also has a DMA engine for performing transfers over the host's peripheral bus. The processor has 128 KB local SRAM on the interface card which is used for storing its program and for buffering packets. Myrinet uses a few control symbols like STOP, GO (for backpressure), FRES (for resetting the network), BRES (backward reset — not implemented at present) etc.

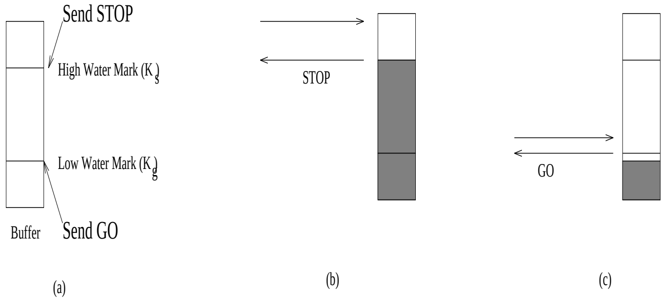

Figure 1: The backpressure flow control protocol. (a) The buffer fills up with flits from the bottom to the top in the figure shown. The two thresholds, K_s and K_g respectively define when the STOP and GO symbols should be sent. (b) When the buffer fills up past the K_s threshold, a STOP symbol is sent to the upstream node. (c) When the buffer empties past the K_g threshold, a GO symbol is sent to the upstream node.

Myrinet employs wormhole routing, backpressure flow control and source routing to achieve low-latency transfer of variable-sized messages — called worms — in the local area network environment. We now briefly describe the Myrinet protocols, to provide a background and reference for the sections to follow.

Wormhole Routing. In wormhole routing [NM93], the unit of information transfer is a worm. A worm can range in length from a few bytes to several thousand bytes. In Myrinet, the maximum worm size is 9 Kbytes (this is a limit imposed by the LANai's control program). Each intermediate switch forwards the worm to the desired output port (if available) as soon as the head of the worm is received, without waiting for the entire worm to be assembled. Thus, the worm can stretch across several nodes and links at any one time. When the desired output port is not available, the worm is blocked and this information is propagated upstream using backpressure. Wormhole routing in Myrinet allows low-latency transfer of information (in the best case, the end-to-end delay is just the propagation delay on the links traversed plus the minimal switching latency).

Backpressure Flow Control Myrinet employs a hop-by-hop flow control scheme called **backpressure flow control** (as described in Figure 1). When the desired output port is unavailable the worm is blocked, i.e., a portion of the worm is stored in a (small) buffer associated with the input port called a slack buffer. Using the STOP/GO mechanism, the rest of the worm is stored in the slack buffers of upstream nodes.

One drawback of backpressure is the possibility of creating a cycle of blocked worms which result in deadlock [DS87, PGL96]. Upon detection of a deadlock, one or more worms could be dropped to break the cycle. However, in this paper, we focus on reliable networks in which loss of worms inside the network must be avoided. Hence, deadlock-free routing must be employed. We next describe the deadlock-free up/down routing approach that was first used by Autonet [SBB+91] and now is employed by Myrinet.

Up/Down Routing. In up/down routing, one of the nodes is chosen arbitrarily as the root of a tree and all links of the topology are designated as 'up' or 'down' links with respect to this root. The 'up'/'down' state of a link is relative to a spanning tree computed in the background by a distributed algorithm. A link is 'up' if it points from a lower to a higher level node in the tree (i.e. to a node at a lesser distance from the root). Otherwise, it is 'down'. For nodes at the same level, node IDs break the tie. The routing from a source to a destination is done in such a fashion that zero or more 'up' links (towards the root) are traversed before zero or more 'down' links are traversed (away from the root) in order to reach the destination. This prevents circular waits and thus the routing is deadlock-free. Myrinet runs a 'mapping' algorithm that computes the current topology in the background.

The advantage of this approach is that node hardware and software are simple. The drawbacks are that the selected paths are generally not shortest paths (i.e. higher average hop distance) and that links near the root may get congested and become bottlenecks leading to lower throughput.

Source Routing. Once the up/down tree has been computed and distributed to all hosts, a source uses it to compute the route (a series of switch output port numbers) to get to a desired destination host. This source route is placed in the header of the worm that is sent out by the source. As the worm comes in on an input port of a switch, the leading byte is read to identify the desired output port. The leading byte is then stripped and the remaining part of the worm is sent to the next switch with a recomputed checksum appended to the end of the worm.

Multicasting over Myrinet. The Myrinet provides no physical support for multicasting or broadcasting. Such physical support is straightforward in bus-type LANs, but difficult to provide in wormhole mesh networks. This is due to the difficulty of performing flow control over a multicast tree, and the increased likelihood of deadlock. The current Myrinet software does provide broadcasting by multicopy unicasting, however. This facility can be used for multicasting by performing filtering at the receiving hosts. Clearly this is inefficient, since it is wasteful of both network link resources in sending to unwanted hosts and of host resources in filtering of unwanted packets.

We therefore examine ways to provide multicasting in a more efficient manner. In the next section we consider how multicasting may be provided within the switching fabric of a wormhole network.

3 Switch-Level Multicasting

The most common way to provide multicasting in a mesh network is within the switches. This is what is currently done by both the Internet and ATM networks. Following this approach in a wormhole network, the replication of a multicast worm into two or more copies must take place in the crossbar switch (see Figure 2). Thus, the simple path created by the unicast worm becomes a tree in the multicast case. As compared with conventional store-and-forward packet- or cell-switching networks, the source routing wormhole network encounters three main difficulties in the implementation of switch-level multicasting, namely: (1) propagating backpressure from the various branches of the tree, (2) extending the definition of source route to map the tree, and (3) avoiding deadlocks.

Propagating backpressure. To perform correct flow control the switch must apply backpressure to an incoming multicast worm if any of its outgoing ports are blocked. This means that source blocking is applied also to the outgoing ports that are not blocked (in other words, these worms will have 'holes' in them, which may be filled by IDLE characters). This is normally allowable in wormhole networks. It does, however, imply wasted resources as links lie idle. In fact, the time for all destinations in the multicast group to receive the entire multicast worm is determined by the slowest path.

Source routes. In the unicast case a source route is simply a list of switch output port numbers on the path to the destination. In the multicast case the route is a tree of port numbers. If we are to preserve the use of source routing in the wormhole network we must transform this tree into a linear representation suitable for use in the worm header. A possible solution is the depth-first traversal of the tree, in which the left subtree is visited first, and then the rest of the subtrees are visited. This scheme is illustrated in Figure 2. At a switch, the multicast source route carried in the incoming worm header consists of a series of port numbers, indicating the output ports to be taken from that particular switch. Immediately after each port number in the header is a pointer (in the form of a byte count from the pointer location to the pointed to location) to the next port number (i.e., the next subtree). Between the first pointer and the next port number is the portion of the source route corresponding to the leftmost subtree. This is stamped in the header of the multicast worm exiting that port. The switch processes the multicast route is as follows: read the port number and pointer value. Copy the bytes indicated by the pointer to that port, followed by an end of route marker. Repeat until the end of route marker is read. Then copy the incoming worm amongst the outgoing ports.

Deadlock avoidance. The multicast routes must follow the same up/down convention as the unicast routes to avoid deadlock. Unfortunately, this alone does not guarantee deadlock freedom, since the backpressure flow-control mechanism introduces dependencies between the diverging branches of the multicast tree for a single multicast worm. A multicast worm may become deadlocked by a unicast (or another multicast) worm, as shown in Figure 3. In this example, the multicast message wants to reach hosts b and c. The unicast wants to reach destination host b.

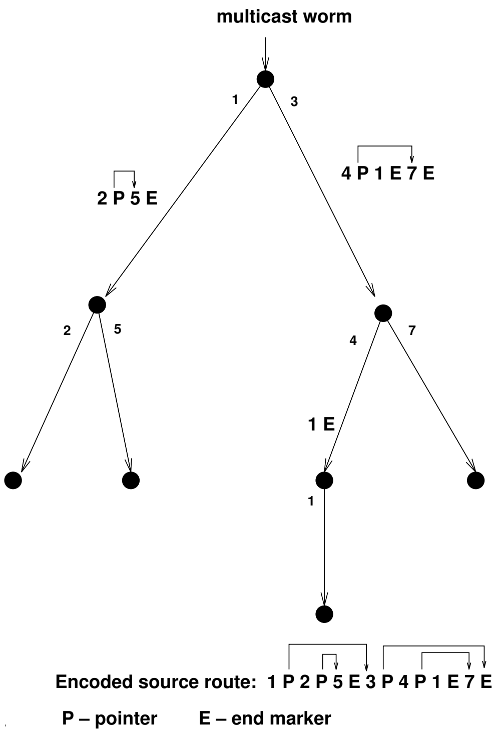

Figure 2: Represention of the multicast source route within the worm header

The multicast worm is backpressured from node C to node A. Thus, node A transmits IDLE characters on the branch (A, B, E, b). The IDLE character stream blocks the unicast worm at node E. We can guarantee freedom from flow control cycles, and hence freedom from deadlock, by restricting all worms to the up/down spanning tree, that is, the tree used for the up/down algorithm. Note that this restriction must be applied to unicast as well as multicast worms. This implies that crosslinks, i.e. links which are not in the up/down tree, (such as link D–E in Figure 3) are not used at all. This restriction may be acceptable if the topology is almost a tree to start with, with a few crosslinks installed as back-ups in case of failure; or, if the traffic is predominantly multicast.

In general, however, it is desirable to utilize crosslinks, at least for unicast traffic. Several variations to the above scheme can achieve this objective. One possible scheme is the following: (a) all multicast transmissions start from the root of the up/down tree (to enforce total ordering of the multicast messages so that one message is entirely received before the next one arrives); (b) when a multicast message is blocked at one point of an intermediate node, the remaining nonblocked ports interrupt transmission altogether (instead of transmitting 'IDLE' characters as before); (c) when blocking ceases, all ports resume transmission (note, the ports which interrupted the worm must prepend the header, which was previously stored at the output port); (d) the destination host reassembles the multicast message from the various fragments (notice that there is no ambiguity of fragments from different multicast messages being interleaved, because of the total ordering enforced).

One can easily verify that the modified scheme allows unrestricted up/down routing of unicast messages, yet avoiding deadlocks. For example, in Figure 3, the unicast worm can proceed unobstructed at node E since the transmission of IDLE characters previously filling the branch (A,B,E,b) is stopped at node A.

The previous scheme tends to favor unicast traffic, by forcing blocked multicast messages to give up the path (by interrupting the flow of IDLE characters). Now we present an alternative scheme which favors multicast instead. We assume that the crossbar switch monitors the traffic going out of its ports and in particular can detect the case when a port is transmitting IDLE characters. When a port has transmitted IDLE characters for a given interval, it is flagged as 'multicast-IDLE'; that is, it is assumed to carry IDLE fills originating from a multicast which has been blocked on one of its branches. When a unicast message is blocked by a multicast-IDLE port, it is flushed from the network (for example, by using a Backward Reset). The source is thus notified of the drop and retransmits the unicast message after a random time out. Referring to Figure 3, the unicast worm, upon arriving at node E is flushed and thus link (C,D) is freed allowing the multicast message to complete.

Although the above designs operate correctly, they still have a few drawbacks. One is that the available bandwidth is much reduced, in general, by the up/down tree-routing restrictions imposed to prevent deadlocks (especially in the first case). Bandwidth is further reduced by the simultaneous blocking of multiple paths to accommodate the slowest path. A more serious drawback is the increased complexity of the crossbar switch. For example worm header processing is considerably more complex than in the unicast case. This increased complexity may impact cost and speed.

One instance in which the switch-level multicasting becomes attractive is broadcasting. Assume the second scheme is used, with serialization through the root of the up/down tree and interruption of the worm upon blocking. In the broadcast case, the restriction of starting from the root does not cause significant increase in overall path length, because all nodes in the topology must be reached. Furthermore, the routing header is considerably simplified. In fact, the source routing address now consists of a unicast address (to the root) followed by a simple broadcast address, which instructs the switch to route the worm to all down links in the up/down tree. The up/down link information is stored in the switch after the up/down tree is computed.

4 Multicasting by the Host Adapter Interface

In the previous section we have noted that reliable multicasting within the switching fabric requires several features in the crossbar switch that go beyond the basic requirements for unicast worm handling, and which are not supported by Myrinet at present. Another approach to multicasting is to do worm replication and retransmission entirely in the host adapter card (the LANai in the Myrinet case). That is, the host adapter, upon receiving the multicast worm, copies it to its local host and at the same time retransmits it to one or more hosts in the same multicast group. This way, multicast worms appear as normal unicast worms to the crossbar switches. No modification to the switching fabric is required; rather, all the multicasting functions are implemented in the host adapter (which, in Myrinet, is user programmable).

In the host adapter multicasting scheme, the hosts in the multicast group are organized in a predefined struc-

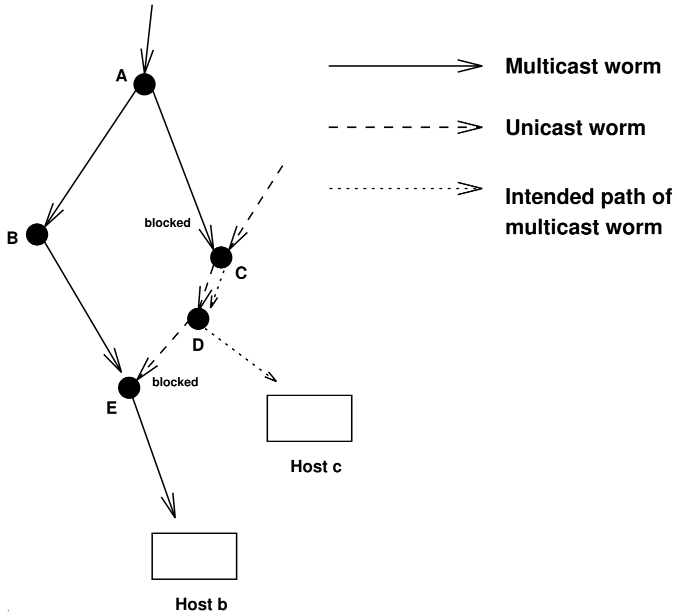

Figure 3: Deadlock involving a multicast worm using up/down routing. The unicast worm has blocked the multicast worm at node C but is in turn blocked by the multicast worm at node E. This leads to a deadlock.

ture (a linear list or a tree, as discussed later). The basic operation carried out by an intermediate host adapter is to (a) recognize the worm as a multicast worm (by the multicast group ID carried in the header); (b) determine, from a multicast group table, the successor host(s) (in the linear list or tree) to which the worm must be forwarded; (c) transmit the worm to each one of the immediate successors (preferably in a cut-through mode) and at the same time copy the worm to the local host.[1]

The host adapter must provide for full buffering of the worm in its memory in case the worm is blocked (backpressured by the network) while being transmitted to the successor. Failure to provide full worm reassembly buffering at each intermediate host adapter can potentially lead to path deadlocks. For example, in Figure 4 the multicast worm X is being forwarded (in a cut-through mode) through host adapters A and B. After exiting B, worm X in turn is blocked by another worm (say, unicast worm Y). This worm in turn is blocked by the segment of worm X in transit between A and B. Host adapter B has insufficient buffering to store worm X and thus backpressures it, causing a deadlock (note that host B cannot drop worm X at this point without compromising reliable delivery). This deadlock situation is resolved by allocating enough buffering in host adapter B to store the entire worm, before the worm is accepted. A possible approach would be to let the originator of the multicast message carry out the reservations before sending the message. Basically, this is the solution implemented in the 'credit' scheme proposed in [VLB96].

Here, we propose a solution which is more consistent with the wormhole routing philosophy, namely we acquire

[1] Cut-through switching is a variant of wormhole routing in which the tail of a blocked worm is totally reassembled in the switch instead of being backpressured into the network, as in wormhole switching. Note that Myrinet does not have network-interface hardware that supports cut-through; thus in Myrinet the worm is fully reassembled and then retransmitted.

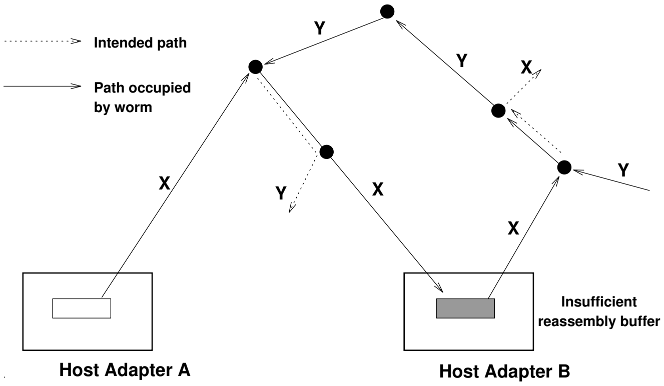

Figure 4: Path deadlock caused by insufficient buffering at host Adaptor B

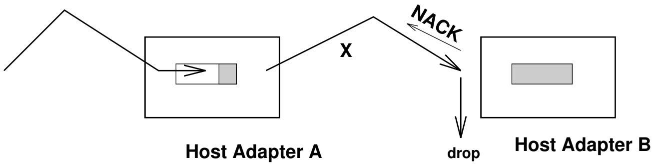

Figure 5: Implicit buffer reservation

resources as we go, and block and wait when the resources are not available. In particular, referring to Figure 5, we assume that host adapter A has secured enough buffering to store the entire worm X. It then forwards worm X (via cut-through) to the successor B. The header of the worm carries the worm size. If B has enough buffers, it accepts the worm and sends an ACK to A. Otherwise, it drops the worm and returns a NACK. In the interim, A has been storing the incoming worm (and also copying it to its local host). If A receives a NACK, it will cease transmission to B, and will resume transmission of the worm (from the beginning) after a time out. Note that eventually the buffer in B will clear (we show later how to prevent buffer deadlocks). In the worst case, the entire worm will be stored in A, waiting to be transmitted. That is, the worm will not be backpressured in the network (this is actually consistent with the Myrinet implementations). Therefore, temporary lack of buffers in host adapter B does not tie up network resources.

Note that in normal operation (i.e. lightly loaded network, small propagation delays) the worm will travel from one host adapter to the next with minimum latency and using only a small portion of the buffers. When network load builds up, the cut-through mode of operation will degrade to a store and forward operation, where the message is fully reassembled at each node, awaiting for a clear path to, or a free buffer at, the successor host.

In some applications, the size of the multicast message may exceed the buffer size on the host adapter (in Myrinet, the host adapter buffer is about 25 Kbytes). This may force the originating host to split the message in smaller fragments and thus increase the overhead. A clever solution, which was reported in [VLB96], is to use the host DMA buffer as an 'extension' of the host adapter buffer. That is, multicast worms in transit through the adapter can temporarily 'overflow' to the host DMA buffer and be retrieved later. Thus, referring to Figure 5, the host adapter B, upon receiving worm X, will check if there is enough buffering

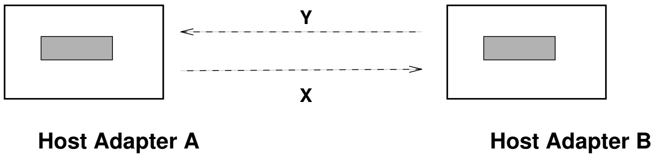

Figure 6: Buffer deadlock caused by competing multicast messages

either in its own queue or in the host DMA buffer. In normal load conditions the worm 'flies' through the host adapter, seldom overflowing to the host DMA buffer.

The step-by-step buffer allocation scheme described above prevents path deadlocks. However, there is still the risk of host adapter buffer deadlocks. Consider the case in Figure 6, where multicast message X has acquired the full buffer in host adapter A, and is requesting a buffer at the successor B. At the same time, another multicast message, Y, has acquired the entire buffer in B and is requesting a buffer in A. This deadlock situation may occur, for example, if two different hosts in the multicast group to which A and B belong have originated different messages; or, if A and B are members of more than one multicast group, and messages X and Y belong to different multicast groups. A simple way of preventing this type of deadlock is to ensure that the multicast propagates from lower ID to higher ID hosts in the multicast group. For example, the hosts can be ordered in a list by increasing ID number, and the multicast propagates from lowest ID to highest ID. In this case, the cycle depicted in Figure 6 will not occur, since the 'wait' arrows can only point in one direction, from the lower to the higher ID.

In general, multicast does not start from the lowest ID host. In fact, it may originate from an arbitrary host in the group. Therefore, at some point, a reversal of the order, from higher ID to lower ID, is required. To handle this situation, we divide the multicast buffers in each host adapter in two classes: the first class is used before the reversal; the second class after the reversal. For example, if multicast propagates along a chain of hosts ordered by increasing ID, the first buffer class is used until the highest ID host is reached. At this point, the worm is forwarded to the lowest ID host, and the second buffer class is used. Likewise, buffers in the second class are used at all subsequent hosts, until the tour is completed. Figure 7 shows a two buffer class allocation example. The multicast originates at host 3 and terminates at host 2.

The two-buffer class scheme does avoid deadlocks if at most one ID reversal occurs during multicast. The proof simply rests on the fact that the buffer requests cannot form a cycle since at each step the request points either to a higher host ID, or to a higher buffer class. The practical implication of this deadlock prevention rule is that each host adapter should have enough memory (possibly including the host DMA extension) to buffer two worms, one before reversal, and another after reversal.

In the following sections, we apply the basic host adapter multicast scheme to two different predefined structures, namely, the Hamiltonian circuit and the rooted tree.

5 Multicasting on a Hamiltonian Circuit

In Hamiltonian-circuit multicasting we form a directed circuit amongst the hosts that are in the multicast group, one

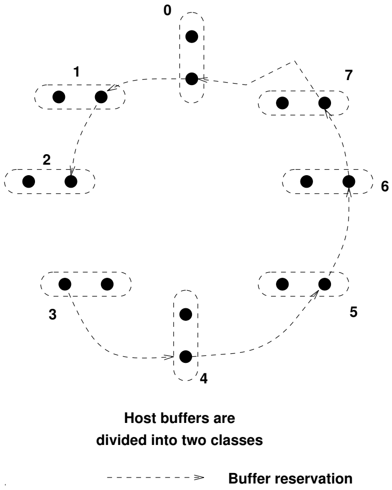

Figure 7: Buffer deadlock prevention by using two buffer classes

circuit for each multicast group on the network. Hosts are ordered by increasing ID number. For this discussion, it is convenient to consider the graph of host connectivity induced on the network topology. Such a graph of host connectivity will be complete, since it is assumed that all pairs of hosts are mutually reachable. The weight of each edge will be some metric which represents the cost of the resources used in traversing the unicast path between the hosts — we simply use the hop count of the path. Each edge of the host connectivity graph therefore represents a simple path in the network graph. The transformation from the network graph to the multicast circuit is illustrated for a sample network topology in Figure 8. A Hamiltonian circuit that could be used traverses the hosts in the order A, B, C, D. The hop length for this circuit is 4.

In order to implement the Hamiltonian circuit multicast scheme the worm header requires, in addition to the information for unicasting, a multicast group identifier and a hop count. At each hop the hop count is decremented by one and retransmission halted when zero is reached. The information required at each host is a table of multicast groups in which each entry consists of the group identifier, the address of the next host and the hop length of the circuit. All hosts must map the multicast group identifier to the next hop; the originator must also initialize the hop count. There are two possible approaches to transmitting around the circuit: the multicast worm may be retransmitted until it returns to its originator, or the last hop may be omitted. The first method has the advantage of providing confirmation of successful multicast, at the slight expense of additional bandwidth use. When deadlock prevention is not strictly enforced, this facility could provide (combined with timeout and retransmission) the guarantee of reliable delivery.

The typical multicast message will originate from an arbitrary host and will travel along the Hamiltonian circuit in ascending order. In some applications, it may be desirable that multicast messages within a group be delivered in sequence (i.e. total ordering). If two arbitrary hosts inject independent messages in the Hamiltonian circuit at the same time, the sequencing is violated, since the destinations will get the messages in different order, depending on their position in the circuit. The sequencing of mes-

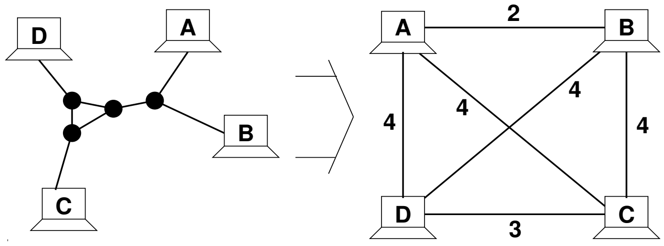

Figure 8: Transformation from network topology graph to multicast graph

sages is easily enforced by requiring that the originator send the message to the lowest ID host, which then starts the multicast. This way, the lowest ID host serializes all the transmissions.

6 Multicasting on a Rooted Tree

The second regular structure over which multicasting has been investigated is a rooted tree. The tree is formed over the host connectivity graph, just as the Hamiltonian circuit was. There is a separate rooted tree for each multicast group. Each node of the tree therefore represents a host adapter, and an incoming multicast worm is retransmitted by the host adapter to its neighbors in the tree.

Multicasting on the rooted tree is identical to multicasting on the Hamiltonian circuit (in fact, the Hamiltonian can be viewed as a degenerate tree), except for two aspects, namely, host ID ordering and cut-through operation. In the rooted tree, hosts must be ordered by increasing ID from root down. That is, the 'children' must have higher ID than the 'parent' (see Figure 9). Furthermore, the multicast must start from the root (i.e. the multicast originator must send the message to the root first.) These two rules prevent buffer deadlocks. They also guarantee total ordering. An alternate approach, also deadlock free, consists of having the originator simply broadcast on the tree. That is, each intermediate node forwards the message to the outgoing links, except to the link on which the message arrived (see host 7 in Figure 9). The two-buffer class rule must now be used: the worm reserves one class while climbing, and another while descending. Note that a worm inverts its direction at most once in its journey. This scheme provides lower latency than the former, but does not guarantee total ordering.

As for the cut-through processing, the host adapter, upon receiving the multicast worm, will identify the list of successors in the tree and will proceed to transmit the worm in cut-through mode to the first successor. In the meantime, the worm is fully received in the buffer. After the worm is completely transmitted to the first successor, transmission to the second successor commences, and so on. Clearly, this procedure reduces to that of the Hamiltonian circuit when the tree reduces to a single chain. One may note that, if load is light (i.e. no contention for bandwidth or buffers), and propagation delays and cut-through latencies are negligible compared with worm transmission time, the tree will lead to higher latency because of full reception at hosts with two or more children. On the other hand, if load is heavy, practically requiring full worm reception at almost every host, the parallelism of the tree will help reduce latency. The tree will also help reduce latency when cut-through operation in the host adapter is not available, as is the case in Myrinet, and when propagation delays are non-negligible.

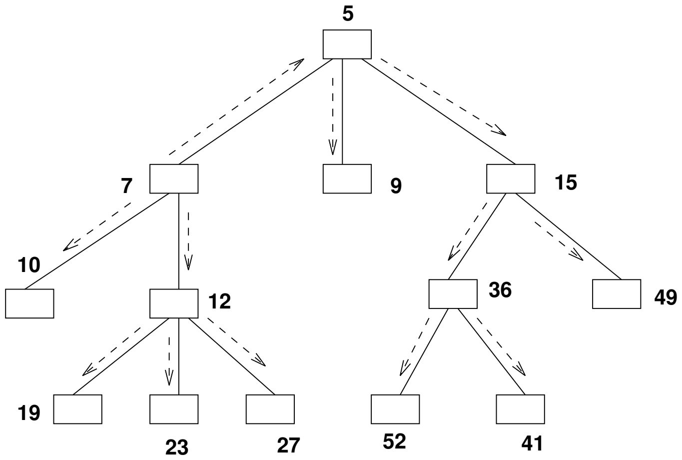

Figure 9: Rooted multicast tree

7 Simulation Experiments

A simulator that emulates the network at the byte level for accurate modelling of wormhole network protocols has been written [BGK+96] in Maisie [BtL94], a C-based, parallel, discrete-event simulation language.

In order to evaluate multicasting strategies for both the Myrinet and the optical backbone the two schemes previously described (Hamiltonian circuit and rooted tree) have been incorporated into the simulator. The simulator (as with the Myrinet) does not propagate backpressure from the host adapter to the network. Arbitrary multicast groups can be created; they are specified in the same configuration file that specifies the network topology. The multicast traffic level is controlled by a parameter that specifies the proportion of generated worms that will be multicast worms. The proportion of multicast worms in the network will, in general, be higher because each multicast worm will be copied and forwarded several times. At the originating host, each multicast worm chooses a multicast group uniformly from among the groups that it is a member of.

7.1 Simulation Results

The Hamiltonian and rooted tree schemes have been compared for two topologies: the 8×8 torus and the 24-node bidirectional shufflenet [PLG95]. The average multicast latency was used as a measure of network loading and performance. The average worm length was 400 bytes. Worm generation was by a Poisson process and the lengths were geometrically distributed.

In the case of the 8×8 torus the proportion of generated multicast worms was 10%. That is, each new worm generated by the host (for hosts that were members of a multicast group) had a 0.1 probability of being a multicast worm. The actual proportion of transmitted multicast worms was higher (approximately 46%) because each worm was reassembled and transmitted several times. Ten multicast groups, each with ten members, were simulated, with the members chosen at random. For the unicast traffic destinations were uniformly chosen. The resulting average

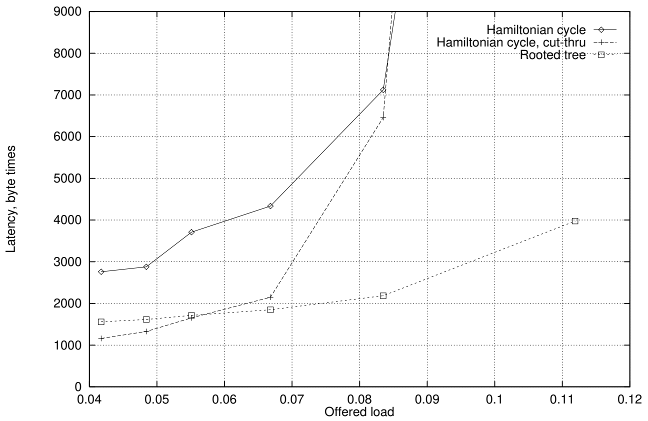

Figure 10: Average multicast latency against network load on 8 $\times$ 8 torus

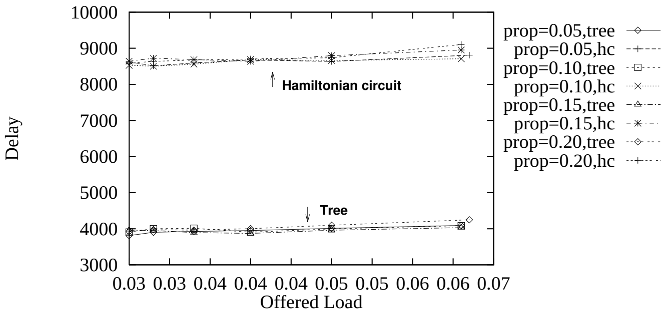

Figure 11: Average delay for varying proportions of multicast traffic on a 24-node bidirectional shufflenet

latencies are shown in Figure 10. The 'offered load' is the output link utilization per host. Three schemes were compared, namely, the Hamiltonian circuit with store-and-forward at each node, the Hamiltonian circuit with immediate cut-through (if the output port is available), and the tree with store-and-forward. The results indicate that the tree scheme produces lower multicast latencies and less loading on the network than the Hamiltonian scheme with full reassembly, as expected. Also, the Hamiltonian with cut-through has lower latency than the tree for light loads, and higher latency for heavier loads, as predicted. As network load increases, the cut-through mechanism provides less of an advantage since the output port is often unavailable for immediate cut-through, and the worm must be buffered. Both Hamiltonian-circuit schemes achieve the same throughput at heavy load. This saturation point is the same as for nonmulticast simulations; the tree multicasting scheme actually achieves higher total throughput because the average hop length for each link of the tree is less than the average hop length for all pairs. The relatively low saturation load is due to the use of up/down routing, which typically causes congestion around the root node. In the simulation a fixed choice of one path per source-destination pair among all possible equal length paths was used. This may also have contributed to lower saturation loads by increasing congestion on some links.

For the 24-node bidirectional shufflenet, the results are reported in Figure 11. Four multicast groups with six nodes per group were considered. The propagation delay on the links was 1000 byte times. As shown in Figure 11, the Hamiltonian circuit approach has, in general, higher latencies for the multicast worms than the tree-based approach. When we compare the actual throughputs the two schemes generate the same amount of traffic. As the proportion of multicast worms increases, the actual throughput also increases.

8 Implementation

A Hamiltonian-circuit multicast scheme has been implemented in the Myrinet. In order to reduce the load on the host kernel and peripheral bus the retransmission of multicast worms is done entirely within the network interface card by the software that runs on the card processor. A control process running on the host as a normal application informs the network interface (via its device driver) of the pertinent multicast information, namely the triple of multicast group, next hop address and hop count. The remainder of the operation is performed by the device driver (for originating hosts only) and the network interface software. The control process, the multicast group manager, is currently a stub process but it is expected to develop into a more complex program that will interact with multicast group managers on other hosts and with the IP group management protocol.

Under this implementation, multicast worms are not returned to their originating host but stop at the previous node in the circuit. Also, worms are stored and forwarded at each host when retransmitted, rather than being cut-through, because of the Myrinet hardware limitations previously described.

8.1 Interoperation with Multicast IP

Multicast IP uses class D addresses to identify multicast groups. This allows a 28-bit address space for multicast groups. Because of the nature of multicasting, class D addresses are only ever used as destination addresses, not source addresses. The actual multicasting is performed in a network dependent manner, and so the mapping from multicast IP address to physical multicast address also varies between networks. For the Myrinet implementation of multicasting, eight-bit multicast group identifiers are used (a separate eight-bit space is used by Myrinet for physical addresses). Multicast group 255 is used for the broadcast address, leaving 255 addresses for Myrinet multicast groups.

This fits naturally into the IP multicast scheme — the low eight bits of the multicast IP address are taken as the Myrinet multicast group. Note that this does not preclude the use of multicast IP addresses that are nonunique in the lower eight bits, since multicast packets are filtered by the receiving IP layer. It is necessary, however, to ensure that Myrinet multicast groups are created which are the union of all IP multicast groups that have common lower eight bits.

Such an address mapping scheme has been incorporated into the Myrinet device driver and this has enabled the multicast implementation to be demonstrated with existing multicast IP applications such as 'wb' [Jac94] (a distributed whiteboard) and 'nv' (a video conferencing system).

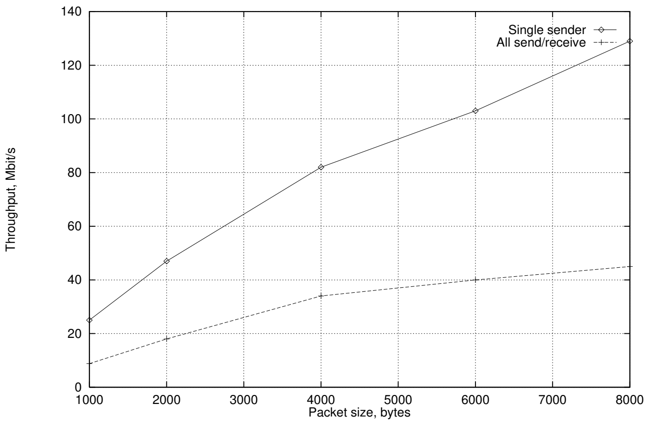

Figure 12: Measured throughput (per host) for a Hamiltonian circuit of eight hosts. *Solid line*, single transmitting host; *Dashed line*, all eight hosts transmitting.

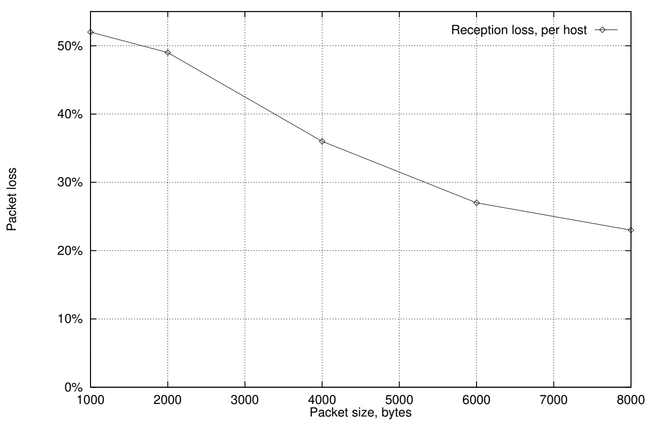

Figure 13: Packet loss rate per host

8.2 Measurements

Only limited experimentation has been possible, because of difficulties in stressing the network with IP traffic given our current hardware configuration (Sun SPARCstation 5s running at 70 MHz). The kernels of these hosts have low IP throughput compared to the network speed. However, it is possible to use tools which communicate with the Myrinet interface via an application space interface. This bypasses the kernel and passes packets directly to and from the network interface. Using this tool, modified to generate multicast worms, it was possible to obtain some measurements of actual network throughput for multicasting. The measurements were performed over a multicast group of eight members using the Hamiltonian-circuit scheme on a four-switch Myrinet configuration. The application simply sent as many packets as possible out to the network. Throughput values for varying packet size are shown in Figure 12. The upper curve shows the throughput for a single host multicasting to the other members of the group. The lower curve shows the received data rate at each host for the case in which all hosts are multicasting to all other hosts. In the single source case no loss of packets due to input buffer overflow was observed. This can be understood from the fact that nodes can receive and forward worms faster than they can originate them.

In the multiple sources case, however, considerable packet loss was measured at the input buffer (the only place that loss can occur in this scheme). The loss rate for incoming worms at each host is shown in Figure 13; lost packets are not included in the throughputs of Figure 12. It was observed that packet loss was only significant if hosts were originating multicast packets as well as forwarding. Since this loss is at *each hop* of the multicast circuit, clearly the total loss rate host-to-host would be unacceptable. If simple backpressure were used in the implementation to prevent buffer overflow then, as discussed above, deadlock could occur. This illustrates the fact that, if high utilization is to be achieved, some sort of deadlock prevention scheme such as has been described in this paper will be required.

9 Conclusions

In this paper, we have proposed several solutions for reliable, efficient, low-latency multicasting in wormhole routing networks. Our philosophy has been to use fully distributed schemes, without a centralized controller. Furthermore, an 'optimistic' approach to resource acquisition has been followed; that is, resources are acquired as we go, without prior reservations, assuming that they will be available with high probability. In case of contention, deadlocks are prevented by carefully defining the rules of allocation of paths (e.g. up/down routing) and buffers (e.g. two-buffer class), and by ordering the hosts in order of increasing ID.

We have explored both switch fabric and host adapter based solutions. The switch fabric based solutions provide the lowest latency and do not require host adapter buffer allocation. However, they require various degrees of complexity in crossbar switch implementation, in order to process source multicast addresses and prevent new types of deadlocks caused by the simultaneous use of multiple paths. None of the proposed schemes can be directly implemented in Myrinet. However, some of the schemes (in particular broadcasting) deserve further investigation in terms of complexity and performance tradeoffs.

The host adapter based solutions are more amenable to implementation in existing systems, since they mostly require software modifications/expansions. Again, the challenge here is to prevent deadlocks (in this case, buffer deadlocks). We have proposed two solutions — the Hamiltonian and the Tree — which are actually closely related. The Hamiltonian offers lower latency in light loads (by virtue of cut-through at each node), while at heavier loads it degrades to store and forwarding routing. At heavy loads, the Tree scheme provides lower latency than the Hamiltonian (in spite of reassembly at each interface) because of parallelism. Both schemes support the total ordering option.

Preliminary experiments (both simulation and measurements) have been carried out to evaluate some of the proposed host adapter schemes. Via simulation, we have confirmed the latency advantage of the tree over the Hamiltonian, in the store and forward case (i.e. no cut-through). A very simple experiment has shown the performance of the Hamiltonian implementation on four Myrinet switches and eight hosts.

Work is in progress in evaluating (via simulation) the actual contention for buffers (and the probability of deadlocks) in various load and traffic pattern conditions. When the probability of deadlocks is not very significant, and the application tolerates it, it may be possible to use less reliable multicast schemes which have a (low) probability of dropping messages, but are much simpler to implement and are much less demanding of network resources. In fact, in some situations it may be more cost effective to relax altogether reliability in network level multicasting (i.e. in the switch or in the host interface) and enforce it at the transport level, using techniques such as the request/repair algorithm reported in [FJM+95].

References

[BCF+95] N. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1), February 1995.

[BGK+96] R. Bagrodia, M. Gerla, B. Kwan, J. Martin, P. Prasasth, and S. Walton. Parallel simulation of a high-speed wormhole routing network. In *Tenth ACM Workshop on Parallel and Distributed Simulation (PADS '96)*, Philadelphia, PA, May 1996.

[BtL94] R. Bagrodia and Wen toh Liao. Maisie: A language for design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering*, April 1994.

[DS87] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[FJM+95] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A reliable multicast framework for lightweight sessions and application level framing. In *Proceedings ACM SIGCOMM '95, Cambridge, MA*, August 1995.

[fST94] Institute for Simulation and Training. Standard for Distributed Interactive Simulation – Application protocols (Draft IEEE standard). Technical Report IST-CR-94-50, University of Central Florida, Orlando, March 1994.

[Jac94] Van Jacobson. Multimedia conferencing on the Internet. In *ACM SIGCOMM '94*, August 1994. Tutorial 4.

[KK79] P. Kermani and L. Kleinrock. Virtual Cut-through: A New Computer Communication Switching Technique. *Computer Networks*, 3(4):267–286, September 1979.

[NM93] L. M. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, 26(2):62–76, February 1993.

[PGL96] P. Palnati, M. Gerla, and E. Leonardi. Deadlock-free routing in an optical interconnect for high-speed wormhole routing networks. In *Proc. Int'l Conf. on Parallel and Distributed Systems (ICPADS '96)*, Tokyo, June 1996.

[PLC95] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing '95*, December 1995.

[PLG95] Prasasth Palnati, Emilio Leonardi, and Mario Gerla. Bidirectional Shufflenet: A Multihop Topology for Backpressure Flow Control. In *Proc. Int'l Conf. on Computer Communications and Networks (ICCCN '95), Las Vegas*, September 1995.

[SBB+91] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker. Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–35, October 1991.

[VLB96] K. Verstoep, K. Langendoen, and H. Bal. Efficient reliable multicast on Myrinet. Technical Report IR-399, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, January 1996.