Practical Multicasting on a Nonbroadcast Subnetwork

Simon Walton and Mario Gerla
Computer Science Dept., University of California, Los Angeles
simonw,gerla@cs.ucla.edu

Abstract

LAN multicast is now an essential service for users of the MBone and other IP multicast tools. Traditional IP multicast has travelled over two types of subnetwork: point-to-point links and subnetworks with inherent multicast capability, such as Ethernet and token ring. More recent LANs may be arbitrary mesh networks with no built-in, efficient, multicast capability. The authors describe a successful implementation of multicasting over a 640 Mbit/s, wormhole-routing LAN. The implementation is integrated with the wider-area multicast IP service. Consideration is given to practical issues such as interoperation with existing multicast routing protocols and software, and multicast-group setup latencies.

1 Introduction

Within the IP community multicast service has gone from being purely a research field to being an everyday, relied-upon tool. The MBone in particular is used by workers in networking and nonnetworking fields as part of their productivity. IP multicast can be used in support of digital virtual environments (DVEs) to reduce the load on the Internet as compared with conventional unicast methods [1].

More local multicast services are also used in support of distributed computations. For these applications low latency is often of paramount importance.

For internet multicasting several routing protocols have been created to allow multicast packet forwarding amongst several subnetworks. An assumption in the design of current multicast routing is that components of the multicast path are point-to-point links (or, equivalently, IP tunnels) or subnetworks with inherent multicast capability. In the case of traditional local area networks such as Ethernet or token ring this assumption of inherent multicast capability is valid, since the multicast capability is built upon the underlying physical-broadcast nature of the LAN. Future high-speed LANs cannot be expected to provide automatic multicast in the same way. For example, local-area ATM, and wormhole-routing networks such as Myrinet (described later), have a point-to-point mesh physical topology. Even giga-Ethernet [2] may move towards such a topology because of the reduced collision radius. Inherent multicast may not be taken for granted in such environments.

To achieve transparent multicast over such subnetworks we have interposed a sublayer above the subnetwork but below multicast IP to provide control of multicasting. This cooperates with the Internet gateway management protocol (IGMP) to dynamically control multicast groups.

2 Myrinet

Myrinet is a commercial wormhole-routing, high-speed (640 Mb/s) electronic LAN [3]. 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 or 32-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 such as STOP, GO (for backpressure) and FRES (for resetting the network).

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.

In wormhole routing [4], 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

0-8186-8061-X/97 $10.00

the propagation delay on the links traversed plus the minimal switching latency).

Myrinet uses hop-by-hop backpressure flow-control. The unit of flow control is a single byte, and a small 'slack buffer' on the input side of each link allows for the round-trip propagation time of the stop command. Such a flow-control scheme on an arbitrary mesh is prone to deadlock, so a restricted form of routing (up/down routing [5]) is employed.

2.1 Myrinet broadcasting

Myrinet provides no physical support for multicasting or broadcasting. Multicasting is difficult to perform over a wormhole-routing network due to the problems inherent in performing flow control over a multicast tree, and the increased likelihood of deadlock. The standard Myrinet software, however, does provide a form of broadcast by performing repeated unicast to all hosts on the network from the broadcast source. This can be used to provide a multicast IP service by mapping all multicast packets to broadcast packets and relying on IP filtering at each host. 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.

3 Wormhole multicasting

In order to provide a multicast service using an inherently nonbroadcast physical subnetwork (such as a wormhole-routing network), the authors have already investigated a number of schemes [6]. The proposed schemes involve the multiple forwarding of a single multicast packet amongst several hosts on the same subnetwork. The two major variants that have been investigated are forwarding around a cycle of hosts and forwarding down a binary tree of hosts. These two variants are illustrated in Figure 1.

Versions of the described cycle-forwarding and tree-forwarding schemes have been implemented. Experimental results for multicasting within a single physical subnetwork have been obtained. For this implementation the cycles of multicast hosts are designated by hand and a utility is executed on each host of the group to enter the cycle into the network software's tables.

3.1 Multicasting via tree-based packet forwarding

The second scheme to be investigated is that of tree-based multicasting. In this scheme we form a logical tree interconnecting the hosts in the multicast group. The tree is formed over the complete graph of host connectivity, with each node of the graph representing a single host, as for the cycle-based case. A single multicast packet is then multiply forwarded along the arcs of this tree.

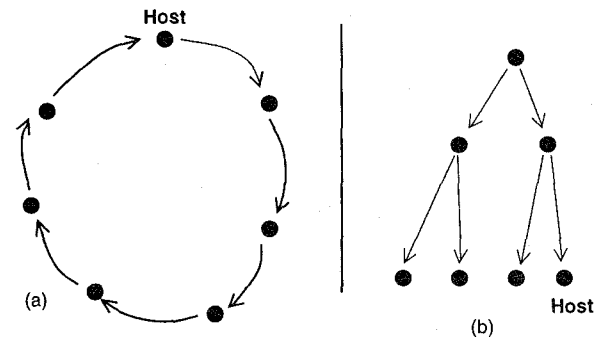

Figure 1: Two proposed schemes for achieving multicast through repeated forwarding: (a) a Hamiltonian cycle of hosts; (b) a binary tree of hosts

The actions for hosts in order to effect this are illustrated in Figure 2. For the host that originates a multicast packet, its actions are to send copies to all its neighbors in the multicast tree.

For intermediate hosts, their actions are to send copies of the incoming multicast packet to all neighbors other than the neighbor from which it was just received. This implies that intermediate hosts must know which host forwarded the multicast packet to them; remember that all packets are received on the same port to the network.

In addition, the intermediate host will pass a copy of the packet up to its own network layer, for normal processing.

If the maximum degree of the multicast tree is bounded then the resources required at each host per multicast group are fixed in size. That is, each hosts only needs to know its neighbors in the tree — it does not need to know the entire tree.

Deadlock in the tree-based scheme. If reliable multicast delivery is required (assuming the absence of line errors), then backpressure applied to any outgoing, forwarded packet must be capable of being propagated to incoming packets. Since each arc of Figure 2 is in reality several links of the underlying network, this propagation of backpressure is liable to cause deadlock in the network.

To remove this cause of deadlock we proceed as before for the cycle-based scheme. We use buffer reservation by the sender, at the receiver, for each individual hop of the multicast tree. Because the sender knows that the receiver will be able to receive the entire packet without blocking then it can safely transmit it. Once the packet has been transmitted then the buffer that it had been residing in can be freed for other incoming multicast packets.

Although this approach removes deadlock from the physical network, it introduces the possibility of deadlock at the buffer-reservation level. This is because dependency cycles of hosts waiting for neighbors' buffers

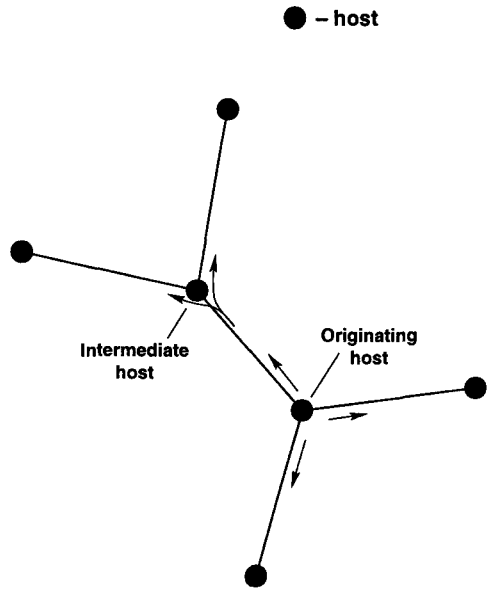

Figure 2: Outline of tree-based scheme for multicasting. The nodes represent hosts and the arcs represent single paths (of possibly several switches) through the network.

may arise. The simplest case is that of two hosts both wanting to transmit multicast packets to each other, and both waiting for a currently-unavailable buffer at the other end.

Buffer reservation. Our basic approach for removing deadlock is to divide the buffers at each host into different classes in order to break resource reservation cycles. Considering initially a single multicast tree, each link of the resource reservation cycle consists of a sending host waiting for a buffer at a receiver to become available. This buffer contains another multicast packet and is itself waiting for another buffer to become available, and so forth. To break the cycle we divide buffers into two classes. A node in the tree is arbitrarily chosen as the root of the tree. At each host, one class of buffers is used for receiving packets from nodes closer to the root, and the other class for receiving packets from nodes further away from the root. Note that this is similar to the up/down algorithm for routing in the underlying network, and proof of deadlock freedom follows in the same way as for that algorithm. For any valid path of a packet through the multicast tree, there will be a transition from the 'up' buffer class to the 'down' buffer class, but never a transition in the opposite direction. Thus there can be no reservation cycles, since each link of the cycle must be a valid path.

3.2 IP multicast

In this section we give an overview of IP multicasting, covering the areas pertinent to the future discussion.

A multicast-capable host is at any given time a member of zero or more multicast groups. Hosts will attempt to join and leave multicast groups as required by the processes that are executing upon them. A host that wishes to join a multicast group sends a 'join group' message as a local multicast on the group it wishes to join. This message is part of the Internet group management protocol (IGMP) [7], and informs any multicast routers attached to the same local subnet that the host will be sending and receiving multicast IP packets on that particular address.

Hosts leave multicast groups in a passive way. Multicast routers periodically send inquiries concerning group membership on their locally attached subnets. Hosts that have left a multicast group simply don't respond to the inquiries. Furthermore, hosts do not respond to the inquiries if they see that another member of the group has already multicast a reply, so as to reduce traffic on the local subnet. Thus the multicast router cannot keep track of the number of hosts currently in a multicast group — it merely knows if there are one or more.

The multicast routers will forward packets amongst several subnets in order to provide Internet-wide multicasting. A number of protocols are used to coordinate the forwarding between the multicast routers, notably DVMRP and multicast OSPF. These routers assume that their network interfaces are attached to either a local subnetwork, with inherent multicast (or broadcast) capability, or a point-to-point link to another multicast router. This point-to-point link may use encapsulated IP in order to traverse nonmulticast-capable routers.

In the case of a locally attached subnet, the IGMP standard says little about the underlying multicast protocol that provides multicast services to the IP layer, except in the specific case of Ethernet. In this case, a simple mapping of the multicast IP address (a class D address) to an Ethernet multicast address is defined.

Since Ethernet possesses an inherently broadcast physical layer there is no particular preparation needed for each local multicast group. IGMP will allow multicast routers to know that local hosts are using a particular multicast group. However, that multicast group must first exist on that local subnet, since the 'join group' message will be sent on that group to the router.

4 IP multicasting for Myrinet

To briefly review the situation just described: we have a Myrinet connecting several workstations. A multicast service has previously been created for the Myrinet. However, to use this service it is necessary to manually configure the multicast group. We wish to perform the transparent integration of this local subnetwork multicasting and the wider area IP multicasting service (for example, the MBone).

4.1 Possible approaches

Observing that the Myrinet has an arbitrary mesh topology, and that the multicast routing program (the Unix `mrouted` daemon) performs multicasting over an arbitrary mesh of networks, one possible approach suggests itself. That is to run a multicasting daemon on each workstation and allow them to create the connectivity necessary to form the group. However, we considered this too inefficient to be a desirable to solution. One of the goals of the original design was to achieve throughput commensurate with the actual network speed. Using the multicast daemon to create multicast internets out of a single Myrinet would involve IP forwarding of multicast packets within the kernel, whereas the existing Myrinet multicast design performs forwarding of multicast packets within the network interface card. As well as the delay caused by kernel processing of the packet up to the IP layer, this approach would entail twice as many transfers of the packet across the workstations' peripheral busses. For the workstations under consideration, the peripheral bus (an SBus) already has a lower throughput than the physical network.

Furthermore, the task of the multicast routing daemon would be considerably larger that in the typical situation, since every other multicast host on the Myrinet will logically be a next-neighbor, connected via a point-to-point link. Multicast routers typically have a small number of neighbors, connected as either point-to-point links or as routers sharing a common local subnetwork.

The eventual chosen approach uses a single multicast daemon on one host attached to the Myrinet. This daemon is responsible for controlling the multicast circuits on the Myrinet, but is only active during the addition of members to a cycle or tree. IP multicast forwarding is not used for multicasting within the Myrinet.

5 Operation of the multicast scheme

5.1 IP multicast address mapping

The Myrinet multicast scheme uses its own numbering system, separate from the host numbering system. Distinct multicast cycles are numbered 0–255. This is a tradeoff between the desire to be able to identify a large number of distinct multicast groups within a Myrinet, and the limited memory space of the network interface card (128 Kbytes, for our models). Since forwarding is performed by the network interface's processor, it must store state for each multicast cycle.

IP multicast addresses are mapped to Myrinet multicast cycle numbers by simply taking the lower byte of the IP address. This does not preclude using different IP multicast addresses with a common lower byte, but multicast cycles must be created which are the union of such IP multicast groups. Obviously this is an efficiency tradeoff, since such groups are, in general, larger than they need to be. IP multicast packets that are not destined for an application at that host will be filtered out by the kernel.

It should be noted that this approach is similar to that taken by the standard for mapping IP multicast addresses to Ethernet multicast addresses [7]. In the Ethernet case, however, the lower three bytes of the IP address are used for the subnetwork address.

5.2 Computation of multicast cycles

As discussed in Section 3.2, a multicast routing daemon that receives IGMP messages cannot keep accurate track of the members of the a group, just that there is at least one member on the subnet. This is unfortunate as it would be useful to know the precise membership of the group in order to create multicast cycles on the Myrinet. However, it is possible to detect whenever a new host joins a multicast group, since it will always send an IGMP 'join group' message. Also, the routing daemon will detect when the final member of the group leaves the group, since it will receive no response to its periodic query.

Thus, from the point of view of the multicast routing daemon, the size of a local multicast group grows monotonically larger and then drops to zero. The Myrinet multicast cycles will reflect this. Hosts that leave a group with at least one remaining member will continue to forward multicast packets. However, most of the effort is performed within the network interface card. The received multicast packets will be dropped by the kernel with the IP layer.

The hosts of the cycle are arranged in order of ascending host ID number. As discussed in detail in our previous paper [6], in order to achieve reliable, deadlock-free delivery it is necessary to impose a total ordering on all multicast hosts. The current implementation does not in actuality provide reliable delivery, (it achieves freedom from deadlock by selective dropping of packets) but we wish to allow for the eventual provision of this service.

Use of a total ordering also simplifies the addition of a new member to the group, as is discussed later.

5.3 Software architecture

The overall architecture employed to control the multicast cycles is depicted in Figure 3. In this architecture, one host on the Myrinet runs a modified version of the standard multicast DVMRP daemon (the `mrouted` program). This host is also part of the MBone, connected via an Ethernet or ATM interface. Each multicast-capable host also runs a small daemon process which communicates with the local Myrinet device driver and network interface card. This communication sets up the multicast cycles. This daemon process may execute as a static server (permanently running on the host) or a dynamic server (executed each time it is needed,

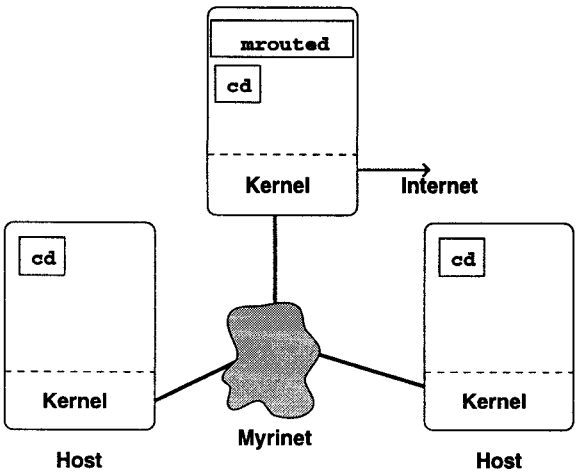

Figure 3: Software architecture for the control of Myrinet multicasting; `mrouted` is the multicast routing daemon and `cd` is the cycle-control daemon

and then exiting). A static server generally provides a faster response, but requires more configuration effort.

5.4 Addition of a host to a group

In this section we consider the actions involved in the addition of a host to a new or existing multicast group on the Myrinet. A multicast application running on one of the Myrinet hosts will make a request to the kernel indicating that it should join a particular multicast group. Assuming the host is not already a member of that group, it will send an IGMP 'join group' message with a destination IP address that is the desired multicast group address. The IGMP packet is then passed to the network interface that is used for multicasting. In the case of multicasting over Ethernet this IGMP packet would then be sent out using a multicast Ethernet address. However, in the case of Myrinet multicasting, it is not possible send the IGMP request on the appropriate multicast cycle, since that cycle does not, in general, exist yet. The multicast routing daemon needs to receive this request in order to initiate the cycle, creating something of a chicken-and-egg situation. It is unfortunate that the IGMP standard specifies the use of the destination multicast address in this manner, rather than using the 'all multicast hosts' address. The desired multicast group address is also carried within the data portion of the IGMP packet, so it is not necessary to use the IP packet's destination address to identify the multicast group being referred to.

To overcome this problem the Myrinet network interface code maps the destination address of all IGMP packets to the broadcast address for Myrinet. The Myrinet network interface card subsequently implements this broadcast by multiple unicast of the same packet, to all hosts on the subnet. This allows the local multicast router to receive all IGMP packets. It does, however, slightly violate the layering principle, since the network device driver must examine the IP header being passed to it. This is felt to be preferable to altering the IP layer of the kernel. Conversion of the multicast address to the broadcast address also creates a greater load on the network, since the packet is sent to all hosts on the Myrinet net, not just multicast-capable hosts. This should not be a problem, since IGMP will normally represent a small portion of the total multicast network load.

Thus the multicast router will receive the IGMP message via the normal IP path, as for reception from an Ethernet interface. However the routing software recognizes IGMP messages from Myrinet interfaces as a special case. The router adds the Myrinet host to the appropriate multicast cycle, creating a new cycle if necessary. It will add itself to any new cycle, since it will be required to perform any bridging of multicast packets to the wider Internet. Recall that there is a many-to-one mapping of groups to cycles, so the appearance of a new group does not always cause the creation of new multicast cycle. After deriving the new or altered multicast cycle the router must update the forwarding tables of all the hosts in the cycle.

Updating of multicast group members. A representation of the complete multicast cycle is *not* stored at each network interface. Instead each cycle is associated with the tuple `<cycle length, next hop>`. For example, Figure 4 shows the entries at each network interface in the multicast table for the cycle shown. For a multicast packet originating at the host containing the network interface, the interface code enters the hop count into the header of the Myrinet packet and sends it to the next host in the cycle. Intermediate hosts decrement the hop count and, if nonzero, look up the next host in the cycle. These intermediate hosts do not examine their stored hop count.

It is not possible for the multicasting tables in all hosts of a cycle to be simultaneously updated. In order to create minimum disruption to a cycle that is already in use the hosts are updated in the manner shown in Figure 5. The new host in the cycle is updated first, followed by the remaining hosts in the order of the cycle. In this way the new host will be able to successfully multicast to all existing members of the group as soon as it is updated. During the period of the update hosts that have been updated will continue to be able to multicast to existing members of the cycle, but their Myrinet packets will have an incorrect hop count until the last member of the cycle is updated. This incorrect hop count will cause the sending host to receive copies of its own multicast transmissions. Since IP allows the delivery of duplicate packets in general, the host software must be able to accept this. Therefore this should not cause a problem.

The routing daemon effects these updates to each host by communicating with the small cycle-control daemon on each multicast host, which performs the

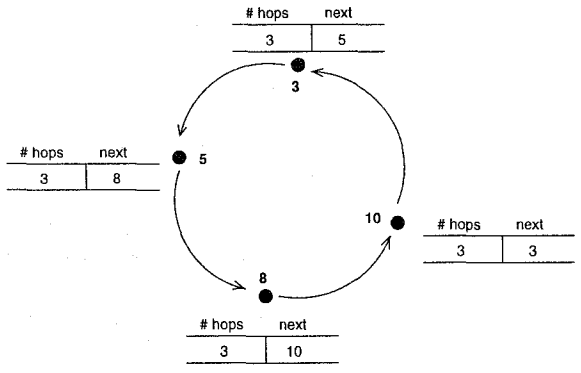

Figure 4: Example multicast cycle with forwarding table entries at each network interface shown

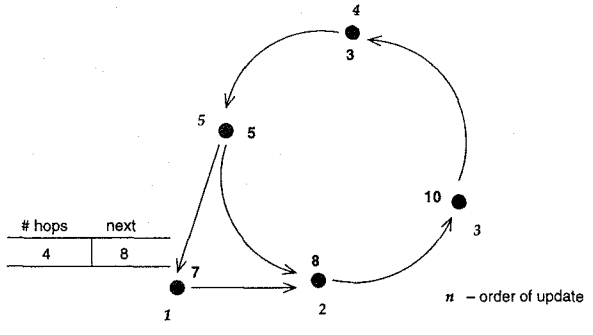

Figure 5: Addition of a new member to the multicast cycle of Figure 4

necessary communication with the kernel. The communication between the routing daemon and the cycle-control daemon uses a simple protocol on top of TCP/IP (which in turn is on top of normal unicast Myrinet). Although a connection-oriented protocol such as TCP involves more processing overhead than alternatives, such as UDP, there are a couple of reasons why TCP is preferred. One is that it is reliable, since a lost update could cause the entire cycle to fail. Another reason is that the successive use of TCP connections ensures that the correct sequence is enforced, since the circuit control daemon will perform the update before closing the connection. Connectionless transmissions can become reordered within hosts, causing the cycle update to take place in an order other than that intended. These factors were felt to outweigh the possibly slower operation of TCP.

5.5 Deletion of members from cycle

As previously discussed, a routing daemon employing IGMP does not normally know when individual members of a multicast group leave that group. Thus a multicast cycle may include members that are no longer actively part of a multicast group. The routing daemon will detect that *all* members have left the group, however. When the membership of a multicast cycle drops to zero, the daemon does not actively update any of the multicast forwarding tables, since having unused entries in the table is harmless. However, it is possible for a multicast cycle to shrink in size without becoming empty. This occurs because several multicast groups may be mapped to a single multicast cycle, and so when a multicast group is deleted it is possible that there are other multicast groups still using that cycle.

In this case the remaining hosts in the cycle have their forwarding tables updated, using the same mechanism as for the addition of a member to the cycle. The update operation is performed in the order of the cycle, starting at an arbitrary host. During this update operation hosts that are ahead of the host currently being updated have incorrect initial hop counts, and so may receive copies of their own transmitted packets. Hosts may furthermore receive duplicate copies of other hosts packets if they travel around the cycle more than once. Again, the higher layers of the communication architecture should be able to handle this. As before, no packets are lost during the update procedure.

5.6 Implementation of tree-based multicasting

A tree-based multicasting scheme has also been implemented. For this scheme the operation at each host on the Myrinet is similar to that of the cycle-based scheme. Forwarding of the multicast packet is again performed by the host interface card. As with the implementation of cycle-based multicasting, there is no reliable delivery; deadlock freeness is guaranteed by dropping incoming packets, if necessary.

Myrinet hosts in the multicast group are formed into trees of maximum degree three. These trees are numbered in the same manner as the multicast cycles were in the previous case. That is, using an eight-bit integer. Each host interface will associate the tree number with its immediate neighbors in the tree. Since the maximum number of neighbors is three this information can be accommodated in a small fixed-size data structure.

The actual operation of a multicast takes place as follows. For hosts that are originating a multicast packet, the packet is transmitted up to three times; once to each of its neighbors. Upon reception of a multicast packet, the host interface will extract the tree number and Myrinet source address from the packet. The interface then forwards the packet to all neighbors other than that from which it was immediately received. In addition, the packet is passed up to the kernel device driver, as for the normal unicast case.

Setup of multicast trees. The automatic creation of multicast topologies uses the same mechanisms for both cycle and tree based schemes. In both schemes the control of the topology is performed by a modified version of the IP multicast daemon. In the tree-based

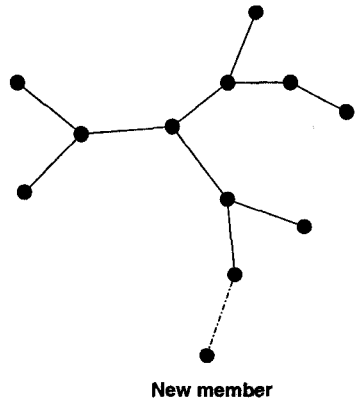

Figure 6: Addition of new node to multicast tree

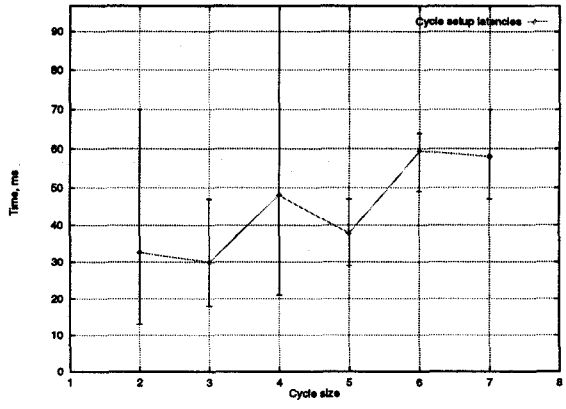

Figure 7: Total delays for the setup of multicast circuits of different sizes, shown with total variation of times

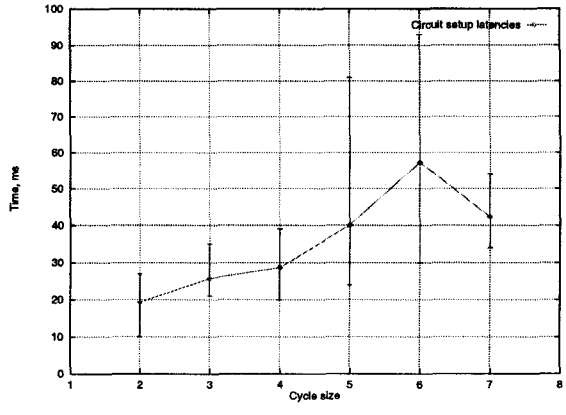

Figure 8: The same measurements as Figure 7, but using a static daemon rather than a dynamic daemon

scheme a minimum depth tree consisting of the group members is constructed. This is done by choosing one member as the root of the tree, and adding a new member as a neighbor of the member closest to the root that can accept another neighbor. Where there is a choice between candidate members, one that is a leaf node is preferred, to reduce overall average latency. Figure 6 shows an example of this. The forwarding tables at the hosts' interface cards are then updated using the same mechanism as before, that is, a TCP connection is opened to a small control daemon on the host and the new information passed to it.

Since new members are always added to the periphery of the current tree there is no disruption to existing hosts by the addition of a new member to the group. Furthermore, the time taken to update the group is independent of the size of the group, because only two nodes (the new node and the node that it is connected to) need be updated.

There is no provision for deleting a general subset of the members of a group. When the group membership drops to zero then the entire tree is deleted.

6 Results

The implementation has been tested on a relatively small Myrinet connecting seven hosts, all of which are Sun SPARCstation 5s. The metric considered most important is the latency involved in creating the multicast cycles. This is mainly composed of the delays in connecting from the multicast routing daemon to the cycle-control daemon on each host of the group.

The time from the multicast router receiving an IGMP 'join group' request to initiating the cycle update sequence is negligible. Measured latencies for the time from the initiation of the cycle update until its completion are shown graphically in Figure 7. In this figure the abscissa is the size of the cycle resulting from the update or creation of a cycle, and the ordinate gives the average latency in milliseconds. The error bars give the total variation observed. The cycle-control daemons were dynamic servers, being executed separately for each request from the routing daemon.

The considerable variation in latencies can be accounted for by the effort of loading and executing the cycle-control daemon, which will depend on the local machine load as well as the load on the NFS backbone network. The high latencies for creating a cycle of size two (the minimum possible size) are caused by the time required to initially load the daemon on request; for subsequent the daemon will be cached for existing members of the group, as well as at the hosts' NFS server. Such delays may occur for larger cycles, if the daemon has been purged from the machines' caches. This could account for the large maximum latency measured for a cycle of size 4.

These latencies may be improved somewhat by using the cycle-setup daemon in its static configuration. In this mode the daemon runs permanently on each host, thus avoiding the execution start-up delay each time it is connected to. Figure 8 shows the same measurements as Figure 7 but using a static daemon. As can be seen, the improvements are relatively small, be-

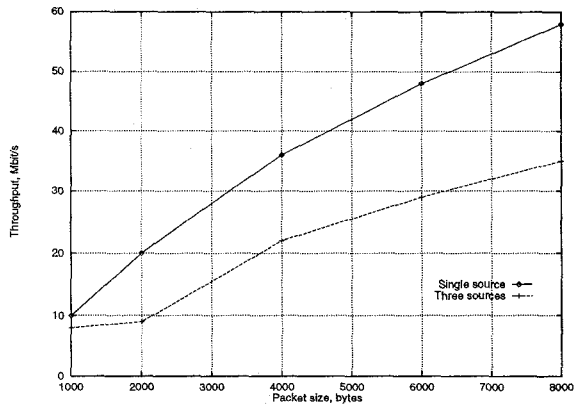

Figure 9: Measured throughputs at the receiver for the tree-based scheme

ing most pronounced for the cycle-creation case of two members. This can be attributed to the effectiveness of the code caching in the case of the dynamic daemon.

It should be noted that the measured latency was in all cases less than 0.1 seconds. This is compared to a typical multicast group lifespan of several minutes. This delay will be insignificant, for example, in the case of the multimedia MBone tools.

Once a multicast cycle has been established the maximum achievable throughputs are as previously reported [6], *viz.* greater than 120 Mbit/s for a single sender and large packet sizes. This was achieved using an application programmers' interface directly to the network adapter; using the more normal kernel IP interface throughputs are limited by the host's CPU to around 30 Mbit/s.

6.1 Tree-based scheme

Some measured multicast throughputs for the tree-based scheme are presented in Figure 9. The testbed is the same as that described in the previous section. These were obtained by using an application-level interface to the Myrinet, rather than kernel multicast IP. They are, however, true application-to-application multicast measurements.

These measurements were accompanied by high rates of packet loss, mainly occurring at the central node of the tree, due to the difficulty in forwarding data at twice the rate that it is received. The overall loss rate observed was as high as 87%.

Tree-setup times. When the multicast tree is created by the multicast routing daemon, the time to add a new node to the tree is independent of the number of nodes currently in the tree, since in all cases only two nodes have their forwarding tables updated. For our experimental testbed, using a dynamic control daemon, the observed latencies for this updating operation varied from 7 ms to 121 ms, with an average time of 34 ms.

7 Conclusions

We have shown that it is possible to extend the IP multicast infrastructure to include high-speed networks that lack built-in provision for such a service. The implementation described is efficient in terms of both host and network resources, involves minimal changes to existing routing software, and no changes to the IP multicast protocol. The implementation gives good performance in terms of latency and packet loss or duplication. We have identified problems with the internet group-management protocol when it is applied to a nonbroadcast subnetwork; these problems could perhaps be addressed in future revisions of the standard.

In our comparison of multicast forwarding using two different topologies, we have found that the cycle topology gives better performance when implemented, despite the tree topologies better theoretical performance. This is most likely due to the network's link speeds outstripping those of the network interface card's processor.

References

[1] B. Roehle, "Channeling the data flood," *IEEE Spectrum*, vol. 34, Mar. 1997.

[2] Gigabit Ethernet Alliance, *Gigabit Ethernet (White Paper)*, 1996. http://www.gigabit-ethernet.org.

[3] 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*, vol. 15, Feb. 1995.

[4] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, vol. 26, pp. 62–76, Feb. 1993.

[5] 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*, vol. 9, pp. 1318–35, Oct. 1991.

[6] M. Gerla, P. Palnati, and S. Walton, "Multicasting protocols for high-speed, wormhole-routing local area networks," in *Proc. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '96)*, (Palo Alto, CA), Aug. 1996.

[7] S. Deering, "Host extensions for IP multicasting," tech. rep., Request for Comments 1112, Aug. 1989.