Broadcast/Multicast over Myrinet using NIC-Assisted Multidestination Messages*

Darius Buntinas   Dhabaleswar K. Panda   Jose Duato   P. Sadayappan

Dept. of Computer and Information Science
The Ohio State University
Columbus, OH 43210
email:{buntinas, panda, duato, saday}@cis.ohio-state.edu

Contact Author: Darius Buntinas

Abstract

Broadcasting and multicasting are common operations in parallel and distributed programs. It would be beneficial to be able to reduce the latency of this operation as much as possible. Some modern network interface cards (NICs) have programmable processors which can be used to provide support for collective communications such as broadcast/multicast. However these processors are 5-15 times slower than the host processor.

In this paper we design and implement a *multi-send* primitive to support efficient broadcast/multicast that requires minimal assistance from the NIC. Our scheme is designed with the idea that as much processing as possible should be done by the host processor. This gives us more flexibility with, for example, creating multicast trees which would be optimal for a particular message size, or choosing a multicast tree dynamically based on requirements of bandwidth versus latency for a particular message. Also, because the proposed scheme does less processing at the NIC, the impact of adding such NIC-supported multicast operation to a run-time system is very small (less than 500ns for non-multi-send packets) when compared with others (a $5\mu s$ additional per-packet overhead in the NIC alone in [10]).

We have designed a multi-send primitive and implemented it as an addition to Illinois Fast-Messages (FM) [5, 4] 2.1 running over a Myrinet [2] network. To fully utilize the benefits of this primitive, we propose a method for constructing an optimal multicast tree using the new primitive. We have evaluated this scheme and got up to a 173% speedup for multicasting 16K messages with 16 nodes.

Keywords: Cluster Computing, Fast-Messages, NIC-Assisted, Broadcast, Multicast, Myrinet

*This research is supported in part by an NSF Career award MIP-9502294, NSF Grant CCR-9704512, and Ameritech Faculty Fellowship Award, and grants from the Ohio Board of Regents

1 Introduction

Broadcasting and multicasting are common operations in parallel and distributed programs. For example, MPI has a broadcast operation defined as part of the standard. It would be beneficial to be able to reduce the latency of this operation as much as possible. Some modern network interface cards (NICs) have programmable processors which can be used to provide support for collective communications such as broadcast/multicast.

Using programmable network interface cards (NICs) to support broadcasting/multicasting in a cluster is not a new idea. Bhoedjang [1] and Verstoep [10] have implemented NIC-supported multicasting, and Kesavan [3] and Sivaram [8] have evaluated different aspects of NIC-supported multicasting. In each of these schemes, the NIC is responsible for all aspects of the operation, including creating the tree and handling flow control. However, the processors found on NICs are usually much slower than the host processors, for example, the processors on current generation clusters are 300-500MHz while the processors on the Myrinet NICs are 33-66MHz, this is roughly 5-15 times slower. This limits the amount of work that can be done by the NIC without compromising performance.

This raises a challenge whether new communication mechanisms can be developed for clusters to support broadcast/multicast with minimal NIC assistance, while delivering good performance. In this paper we take on such a challenge. We introduce a multidestination message passing mechanism. Such a mechanism has been developed earlier for router-based parallel systems [7, 6, 9] to support effective collective communication.

In this paper we design and implement a *multi-send* primitive to support efficient broadcast/multicast that requires minimal assistance from the NIC. Our scheme is designed with the idea that as much processing as possible should be done by the host processor. This gives us more flexibility with, for example, creating multicast trees which would be optimal for a particular message size, or choosing a multicast tree dynamically based on requirements of bandwidth versus latency for a particular message. Also, because the proposed scheme does less processing at the NIC, the impact of adding such NIC-supported multicast operation to a run-time system is very small (less than 500ns for non-multi-send packets) when compared with others (a 5μs additional per-packet overhead in the NIC alone in [10]).

We have designed a multi-send primitive and implemented it as an addition to Illinois Fast-Messages (FM) [5, 4] 2.1 running over a Myrinet [2] network. To fully utilize the benefits of this primitive, we propose a method for constructing an optimal multicast tree using the new primitive. We have evaluated this scheme and got up to a 173% speedup for multicasting 16K messages with 16 nodes.

Section 2 describes the new multi-send primitive, followed by Section 3 which describes broadcasting and multicasting using the new primitive. Construction of the optimal multicast tree is described in Section 4. The implementation details are discussed in Section 5 followed by our experimental results in Section 6. Finally we conclude and discuss future work in Section 7.

2 NIC-Assisted Multidestination Message Passing

The basic idea is to create a *multi-send* primitive in which the host writes the packet data to the NIC only once followed by a list of destinations. The NIC would then transmit copies of the packet to each of those destinations. Figure 1 shows two diagrams where host 0 is sending packets to hosts 1 through 3. The left diagram shows host 0 making three unicast (point-to-point) sends, each of which is forwarded by the NIC to its destination. The diagram on the right shows the host making a single multi-send operation to the NIC which then forwards a copy of the packet to each of the destinations.

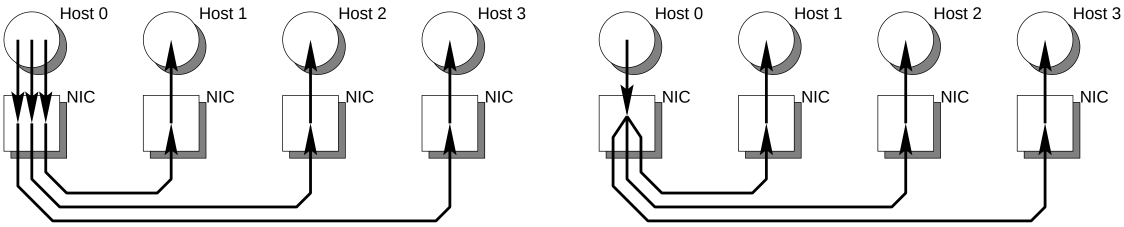

Figure 1: Multiple host-based point-to-point operations (left) and a NIC-assisted multi-send operation (right) to four destinations.

Figure 2a shows the timing diagram for a multi-send operation sending a packet to four destinations, and the receive time for the last destination. Figure 2b shows the corresponding timing diagram for host-based point-to-point operations. In the figure the interval marked *Send* corresponds to the time it takes the host to assemble a packet and write it to the send queue on the NIC, the interval marked *Xmit* corresponds to the time it takes the NIC to transmit the packet from the send queue to the network, and the interval marked *Recv* corresponds to the combined time for the NIC to receive the packet, including the network latency, for the NIC to then forward it to the host and for the host to process the packet. Notice that in both diagrams the receive time for a packet at the last receiver is overlapped with the transmission time at the sender for that same packet. In Figure 2b, for the first three packets, the network transmit time of one packet is overlapped with the host send time of the next and so it is not shown. As indicated below, timing parameters for FM over Myrinet are such that this will always be the case, regardless of the packet size.

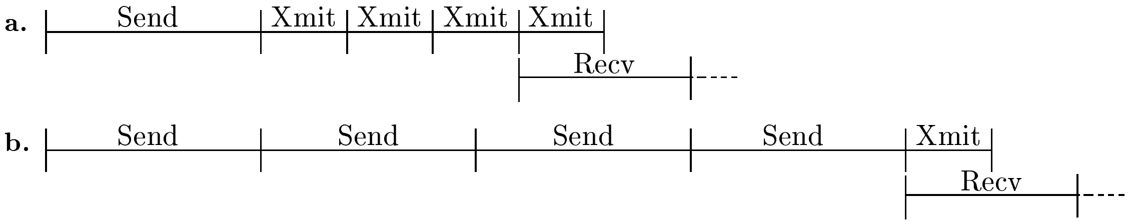

Figure 2: Timing diagram comparing latencies for sending one packet to four destination using a) a multi-send operation and b) host-based point-to-point operations.

Let us compare the latency of sending a packet to k destinations using a multi-send operation with the latency of sending a packet to k destinations using the usual host-based point-to-point operations. The time for the kth destination to receive the packet using a multi-send operation would be $(t_{Send} + (k-1) \times t_{Xmit} + t_{Recv})$, and $(k \times t_{Send} + t_{Recv})$ using host-based point-to-point operations. We have timed the host send, the NIC transmit and the receive operations in FM 2.1 on our cluster of 300MHz Pentium II machines with 33MHz LANai 4.3 Myrinet cards. We estimate the send time to be $(2.7863\mu\text{s}+0.0301\mu\text{s}\times m)$, the NIC transmit time to be $(1.3958\mu\text{s}+0.0075\mu\text{s}\times m)$ and the receive time to be $(2.8862\mu\text{s}+0.0155\mu\text{s}\times m)$, where m is the packet size in bytes. Thus for sending a 1,536 byte packet to five destinations, t_{Send} would be 49.0199μs, t_{Xmit} would be

12.9158μs, and t_{Recv} would be 26.6942μs. The multi-send operation would take 127.3773μs and the usual host-based point-to-point method would take 271.7937μs. This leads to a 213% speedup. We can see that multi-send is a powerful primitive.

3 Broadcast/Multicast with Multi-send Primitive

While the multi-send primitive is powerful, for covering a large number of destinations we need to perform broadcasts and multicasts hierarchically in order to minimize the overall broadcast/multicast latency. This can be done by having the host at each intermediate node receive the message, then issue another multi-send operation to forward the message to its child nodes. This raises a challenge: how to create an optimal tree? It is also interesting to analyze how this scheme is different from the NIC-based multicast schemes described in [1, 10, 8, 3].

In the NIC-based scheme [1, 10, 8, 3], the incoming multicast packet is transmitted to the child nodes by the NIC after it has been forwarded to the host. Figure 3 illustrates the difference in the two methods. This figure shows two diagrams where node 0 sends to multiple destinations, one of which is node 1. Node 1 then sends to nodes 2 and 3. The diagram on the left shows a completely NIC-based scheme where the packet coming into node 1 from node 0 is sent to nodes 2 and 3 by the NIC after being forwarded to the host. The diagram on the right shows node 1 forwarding the incoming packet to the host. The host then issues a multi-send operation to transmit the packet to nodes 2 and 3.

While it may seem that the completely NIC-based scheme would always be better than the method we are proposing, we believe that that is not the case. The completely NIC-based approach puts more responsibility on the NIC which, as previously mentioned, has a processor 5-15 times slower than the host. We believe that the additional processing power available at the host will allow greater flexibility in, for instance, tree construction. So that depending on the message size and quality of service requirements, a tree optimal in either bandwidth or latency, depending on the requirements, for that message size can be used for sending the message. This could be done on a per-message or per-packet basis.

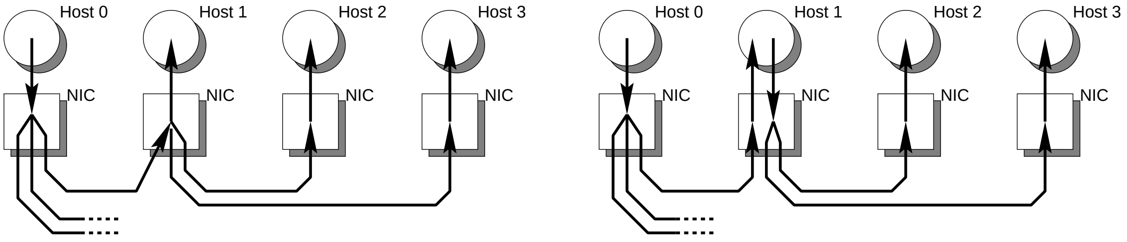

Figure 3: NIC-based multicast (left) and a NIC-assisted multicast (right).

4 Constructing Optimal Multicast Tree

In this section, we show how to construct an optimal multicast tree using the proposed new multi-send primitive. The basic idea is to construct a tree such that that the maximum number of nodes will be sending at any time, so as soon as a node receives a packet it should start sending to its child nodes. Such a tree would be optimal in terms of latency.

The shape of the multicast tree that is optimal in terms of latency would depend on $(t_{Send} + t_{Recv})/t_{Xmit}$. We will call this ratio s. Let us consider the NIC transmit time as one time step, and that time step 0 is when the NIC of the first node starts transmitting. Now, let's assume that it takes an integer s time steps for a node to receive a packet and start re-transmitting it. Another way of looking at this is that s steps after sending a packet to a node that node will start transmitting. The number of nodes transmitting at time step x, assuming all nodes transmit as soon as possible, can be given by the recurrence:

$$T(x) = \begin{cases} 1 & \text{if } x < s \\ T(x-1) + T(x-s) & \text{otherwise} \end{cases}$$

Intuitively, one can think of this recurrence in this way: If x is the current time step, s time steps ago $T(x-s)$ nodes were sending to other nodes. Now, s time steps later, there are $T(x-s)$ new nodes sending for the first time. This is in addition to the $T(x-1)$ nodes which were already transmitting.

The timing diagram for a multicast tree using the above recurrence might look something like Figure 4. This diagram shows node 0 transmitting to nodes 1, 2, 3, 4, 6 and 9, node 1 transmitting to nodes 5 and 8, and node 2 transmitting to node 7. In this diagram the value of s is 4.

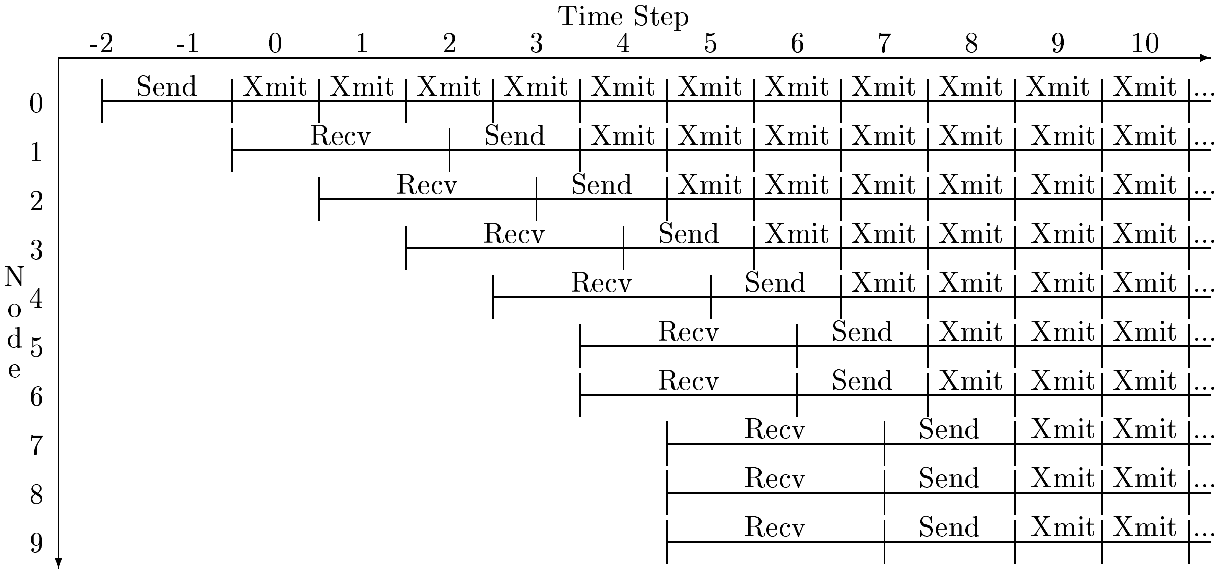

Figure 4: Timing diagram for the first 10 nodes of a multicast tree using the multi-send primitive.

Figure 5 shows an algorithm for constructing a multicast tree using the recurrence above, which is optimal in terms of latency. The main loop iterates once per time step. At time step ***step*** there are $T(\textit{step})$ nodes sending. The inner loop iterates through each sender at the current step, assigning a new destination to each. Notice that when s is 1 we have a binomial tree. The tree constructed is optimal in terms of latency, because at each time step, all available nodes are transmitting.

In the actual implementation it is unlikely that s will be an integer value. If s is rounded up to the next highest integer, then nodes lower down in the tree will start transmitting earlier than expected, and so the number of nodes actually transmitting for some time steps x will be larger than the expected $T(x)$. This would result in nodes lower in the tree finishing earlier than

```
step = 0
dest = 1
loop
    for i = T(step) - 1 downto 0
        append node dest to destination list of node i
        dest = dest + 1
        if dest > MAXDESTS
            break
    step = step + 1
```

Figure 5: An algorithm for constructing an optimal multicast tree for MAXDESTS nodes.

expected, and so for a tree with a large number of nodes, it may be the case that some nodes lower in the tree, which could be transmitting, are sitting idle, while other nodes higher up at the tree are still transmitting. If we round s down to the next lower integer, we would have a similar problem, except that it may be possible that the higher level nodes may be idle while lower number nodes are transmitting. Also, the host send time for a packet depends on the number of destinations that it will be sent to. So the send time for each node may be different. Our model above assumes a constant overhead.

Calculating a value for s is difficult for the reason stated above and because it varies with message size. The value s is also dependent on the speed of the processors of a particular node as well as the processing and communication load for that node. Longer send or receive queues, for instance, may affect the values of t_{Send}, t_{Xmit} or $t_{Receive}$, which then affect s. For this reason, we have determined values of s for different message sizes and numbers of processors experimentally.

So far we have focused on single packet messages. For multi-packet messages, it is important for the host at the intermediate nodes to pipeline the reception and re-sending of the message packet-wise. In other words, the host should not wait for the entire message to be received before initiating the multi-send to its child nodes. Instead, the host should multi-send a packet as soon as it receives it.

5 Our Implementation of the Multi-send Primitive

We used Illinois Fast Messages (FM) version 2.1 from the HPVM 1.0 distribution for the base of our experiments. This version was used because it is the last version of FM available for Linux.

In FM 2.1, a message is associated with a stream. In order to send a message, a stream is created between the node and the destination with the FM API call `FM_begin_message()`. Data can then be sent on the stream using the `FM_send_piece()` API call. When enough data is sent to fill a packet, FM creates a packet and writes it to the NIC which in turn transmits it to the destination node. On the receiving side, FM calls a message handler to optionally re-assemble the message, packet by packet, into the user buffer. A stream is terminated using the `FM_end_message()` API call, at which time any remaining data is assembled into a packet and written to the NIC. Before any packet is sent FM checks if the sender has sufficient credits for the receiving node and blocks if there aren't enough. This is used to prevent buffer overflow at the receiving side.

We modified FM by adding a new API call, `FM_begin_message_multi()` which creates a stream between the sender and all destinations specified in the destination list given as an argument. As with a regular FM stream, data is sent using `FM_send_piece()` and the stream is terminated with `FM_end_message()`. The flow control mechanism was also modified. Before a packet is sent credits are checked for all destinations. If there are not enough credits for every destination, the operation blocks until there are.

When a packet is ready to be sent, the packet is assembled for the first destination and written to the send frame on the NIC. Next a list of information on each additional destination is assembled and written to a new field which was added to the send frame. Each entry in the list holds the logical node number, the physical node number and the returned credits for that destination. These will be used to update the corresponding fields in the packet header so the packet can be sent to each destination.

Since the host has assembled the packet such that it is ready to be sent to the first destination, the NIC can transmit the packet without checking whether it is a multi-destination packet. Only after the NIC has initiated the transmit DMA and is waiting for it to complete, will it check for additional destinations. This way our modifications add no overhead at the NIC for sending standard FM messages. The NIC then updates the logical node number, credit and route fields of the packet for the next destination. This is done as soon as the transmit DMA pointer has passed those fields but without waiting for the entire DMA to finish. This allows the header field updates to be overlapped with the DMA for all but the smallest messages. After updating the fields, the NIC waits for the DMA to finish then the DMA is immediately initiated to transmit the packet to the next destination. This is repeated for each additional destination.

6 Experimental Results

The performance tests were run on a cluster of 16 300MHz Pentium II machines each with 128MB of RAM running RedHat 5.2 Linux with kernel version 2.0.36. The machines are connected by a Myrinet LAN network via a 16 port switch. Fourteen of the machines have LANai 4.3 cards and two have LANai 4.1 cards.

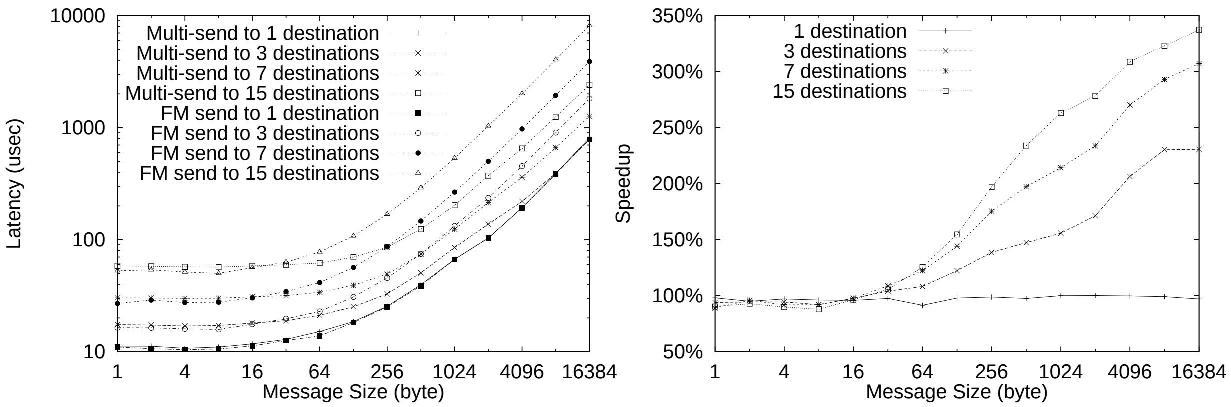

Figure 6: Latency and speedup for NIC-assisted multi-send operation and multiple FM send operations.

We tested the performance of the multi-send operation and compared it with multiple unicast sends. To produce the results for k destinations, out test program sent messages to $k-1$ nodes, then a message to itself. The last message will go out to the switch, then be received back into the host, so the latency of this message should be the similar to the latency for any other destination. The test program performed this 10,000 times for each message size. Figure 6 shows the results of this test. It can be observed that the multi-send primitive achieves a 337% speedup for sending a 16K message to 15 destinations.

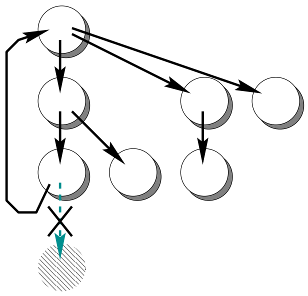

Figure 7: Eight node binomial tree with the root node substituted for a leaf node.

To test the performance of the multi-send operation using a multicast tree, for each configuration we wanted to test, we timed the latency for 10,000 messages, each time waiting for the previous message to be received before sending the next message, then took the average over all messages. We directly timed the latency by constructing the tree and then substituting the root node for the node which would receive the message last. Figure 7 shows an example for an 8 node binomial tree in which this was done. Since we can not be sure which node will be the last node, we ran the test several times substituting the root node for a different leaf node each time, then took the maximum of those latencies. Figure 8 shows the results of this test. Again, it can be observed that a 173% speedup is achieved for multicasting an 8K message to 16 nodes using an optimal tree with the multi-send primitive compared to a binomial tree with the unicast primitive.

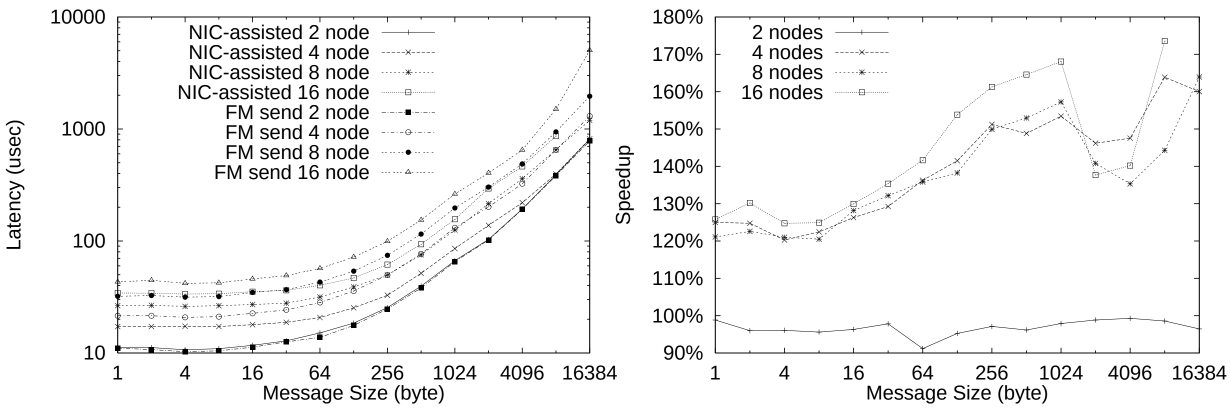

Figure 8: Multicast latency and speedup for NIC-assisted multicast using the best tree and FM send multicast using binomial tree.

Notice the dip in speedup for 2K and 4K messages in Figure 8. We believe this dip is due to the fact that the messages were not pipelined through the intermediate nodes. Unfortunately, we were not able to add that functionality at the time of this writing. We expect to add this feature soon and that the performance of multicasting with the multi-send primitive will improve as a result. The new results will be added to the final version of this paper and presented at the workshop.

Since trees constructed using different s values will give varying results, we ran the test for a range of s values then took the minimum for each message size for the values used in Figure 8. Ideally, a sophisticated multicast algorithm implemented at the host could determine the best tree to use given parameters such as message size and number of destination nodes. Figure 9 shows the latency curves for an 8 node multicast using two trees which gave the best results for some message sizes. In this case the s values were 6 and 7 respectively.

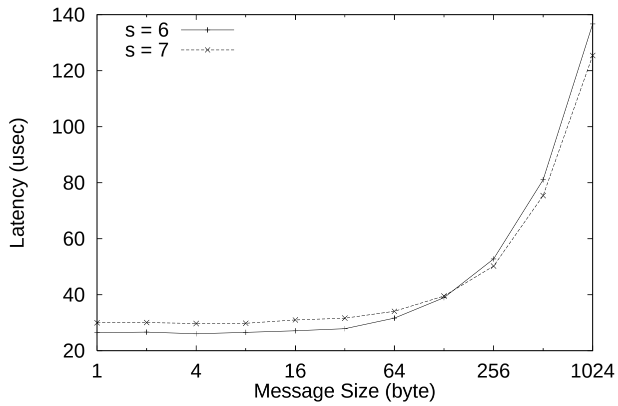

Figure 9: Latency for an 8 node multicast using two different trees.

7 Conclusions and Future Work

We have introduced a new multi-send primitive, which uses the NIC to transmit multiple copies of a packet to different destinations. We then showed how this primitive can be used in a multicast tree to further improve performance for large destinations sets and proposed a method for constructing optimal multicast trees.

The multi-send primitive gave us 337% speedup over conventional host-level iterative sends for 16K messages. We also observed a 173% speedup for 8K messages and 16 nodes when using the primitive in a multicast operation using an optimal tree versus using a binomial tree with the traditional host-level unicast sends.

We intend to improve the performance of the multicast by adding message pipelining capability for the intermediate nodes of the tree. We also intend to study the effects of message and destination list sizes on the shape of the optimal multicast tree. The current evaluations are based on a 16-node cluster with a 16 port myrinet switch, thus, there is no network contention during the multicast operation. For larger systems, network contention will play an important role. We plan to develop a method for constructing trees for multicast using multi-send primitives which would give minimal contention for arbitrary irregular networks.

On a more general level we intend to investigate whether other collective communication operations, such as barriers, reductions, or personalized multicast could benefit from similar NIC-assisted primitives.

References

[1] R. A. F. Bhoedjang, T. Ruhl, H. E. Bal, "Efficient Multicast On Myrinet Using Link-Level Flow Control". *International Conference on Parallel Processing*, pp. 381-390, August 1998.

[2] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su, "Myrinet: A Gigabit-Per-Second Local Area Network". *IEEE Micro* 15(1), pp. 29-36, February 1995.

[3] R. Kesavan and D. K. Panda, "Optimal Multicast with Packetization and Network Interface Support". *International Conference on Parallel Processing (ICPP 1997)*, pp. 370-377.

[4] M. Lauria, S. Pakin and A. Chien, "Efficient Layering for High Speed Communication: Fast Messages 2.x". *Proceedings of the 7th High Performance Distributed Computing (HPDC7) Conference*, July 28-31, 1998.

[5] S. Pakin, M. Lauria and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet". *In Supercomputing '95.*

[6] D. K. Panda, "Fast Barrier Synchronization in Wormhole k-ary n-cube Networks with Multidestination Worms". *International Symposium on High Performance Computer Architecture (HPCA '95)*, pp. 200-209, 1995.

[7] D. K. Panda, S. Singal, and R. Kesavan, "Multidestination Message Passing in Wormhole k-ary n-cube Networks with Base Routing Conformed Paths". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 1, pp. 76-96, January 1999.

[8] R. Sivaram, R. Kesavan, D. K. Panda, and C. B. Stunkel, "Where to Provide Support for Efficient Multicasting in Irregular Networks: Network Interface or Switch?" *International Conference on Parallel Processing (ICPP 1998)*, pp. 452-459.

[9] C. B. Stunkel, R. Sivaram, and D. K. Panda "Implementing Multidestination Worms in Switch-Based Parallel Systems: Architectural Alternatives and their Impact". *International Symposium on Computer Architecture (ISCA'97)*, Vol. 25, No. 2, pp. 50-61, June 2-4 1997.

[10] K. Verstoep, K. Langendoen, and H. E. Bal, "Efficient Reliable Multicast on Myrinet". *1996 International Conference on Parallel Processing*, Vol. 3, pp. 156-165, August 1996.