Optimal Multicast with Packetization and Network Interface Support [1]

Ram Kesavan and Dhabaleswar K. Panda

Dept. of Computer and Information Science
The Ohio State University, Columbus, OH 43210-1277
Tel: (614)-292-5199, Fax: (614)-292-2911
E-mail: {kesavan,panda}@cis.ohio-state.edu

Contact Author: Dhabaleswar K. Panda

Abstract: Most multicast algorithms proposed in the literature assume an arbitrarily long message being communicated as a single packet. Modern networks typically limit the size of the largest packet, and long messages are packetized and transmitted. Such networks also provide network interface support for nodes, which typically includes a coprocessor and memory, to implement the lower layers of the communication protocol. Such network interfaces can be programmed to support efficient multicasting to eliminate software overhead for higher layers during absorb and retransmit. In this paper, we present an optimal multicast algorithm for systems with such smart network interface support for packetization. Two implementations of smart network interface, the First-Child-First-Served (FCFS) and the First-Packet-First-Served (FPFS), are studied and compared. It is shown that the FPFS network interface support is more practical and efficient. Next, the multicast latency is modeled for the FPFS implementation. A concept of k-binomial tree is introduced, and proved to be optimal for multicasting using such network interface support. A method to construct contention-free k-binomial trees on contention-free orderings of the nodes is also shown. The performance of the optimal k-binomial tree is compared with the standard binomial tree for a range of multicast set size and number of packets. For a 64 node system, and parameters of $t_s = 12.5\mu$sec (message send overhead), $t_r = 12.5\mu$sec (message receive overhead), and $t_{step} = 7.2\mu$sec (overhead of preparation and transfer of a packet by the network interface), the k-binomial tree is shown to be upto 3 times better than the binomial tree for 24 packet long messages. The k-binomial tree shows increased benefits with increase in number of packets. Thus, these results demonstrate significant potential to be applied to current and future generation high performance systems including MPPs and NOWs, where network interface support for multicast is provided.

Keywords: Interconnection Network, Collective communication, Multicast, Broadcast, Wormhole Routing, Link Contention, Networks of Workstations, Network Interface Support, and Packetization.

[1] This research is supported in part by NSF Grant MIP-9309627 and NSF Career Award MIP-9502294.

Contents

1 Introduction

Multicast/broadcast is a common collective communication operation as defined by the MPI standard [23]. Parallel systems supporting distributed memory or distributed-shared memory programming paradigms require fast implementation of multicast and broadcast operations in order to support various application and system level data distribution functions. Multicast and broadcast also get used for other collective communication operations like barrier synchronization and global combining [21, 25].

There have been many multicast/broadcast algorithms proposed in the literature in recent years [3, 5, 10, 11, 14, 15, 16, 19, 22, 26, 6, 5, 7, 27, 17, 18]. All these algorithms are designed assuming arbitrarily long single packet messages. In modern networks, the size of the maximum packet is limited to minimize network contention and support efficient buffer utilization of buffer spaces both in the network and at the network interfaces. Therefore, long messages are broken up and transmitted as multiple packets. Such networks [1, 2, 4, 9] provide network interface support associated with each node. Such support, which includes a coprocessor and a small amount of memory, implements the lower layers of the communication protocol. This is not just true of workstations on a NOW, but even on high performance systems like the CrayT3D [9] and IBM SP2 [1]. This leads to a challenge for designing optimal multicast algorithms under packetization using network interface support.

Recently, a solution for implementing multicast under packetization has been proposed in [8]. However, this work assumes that packetization is handled by the host processor which sends and receives messages packet by packet. Moreover, this framework assumes user/system control over determining optimal packet size for a given multicast set and message length. Thus, this result is not applicable for systems with fixed packet lengths and for systems with network interface support for packetization. Some recent research focus on implementing efficient collective communication operations on a NOW over ATM using the network interface support [13]. Since ATM switches provide hardware multicast capability, the network interface support in this work is primarily is geared towards achieving reliable multicast over the unreliable ATM layer. A recent work [28] describes an implementation of packetized multicast over the Myrinet network interface. This work is also geared towards development of a reliable multicast communication layer and does not provide any formal multicast algorithms. To the best of our knowledge, there has been no work on designing optimal multicast algorithms for systems with packetization and network interface support.

In this paper we take on such a challenge. First, the features of a typical network interface support are analyzed and differences between conventional and smart network interfaces to support multicast are established. Next, two implementations of smart network interface, First-Child-First-Served (FCFS) and First-Packet-First-Served (FPFS), are investigated for packetized multicast. It is shown that the FPFS network interface support is more practical and efficient in terms of buffer requirement. It is also shown that the binomial tree is not optimal for an arbitrary multicast set size and an arbitrary number of packets using the FPFS implementation. In order to derive the optimal multicast tree the properties of the FPFS implementation are studied to calculate multicast latency. Then, a new concept of k-binomial tree is defined and shown to be optimal for multi-packet multicast. The concept of a contention-free ordering of nodes has been developed in [22] to implement contention-free multicast with minimum latency. We develop a method to construct contention-free k-binomial trees on a similar ordering. The k-binomial tree is experimentally shown to be better than the conventional binomial tree on a 64 node system for a range of multicast set sizes and number of packets. With parameters representative of current technology, the k-binomial tree is shown to perform upto upto 3 times better than the binomial tree for 24 packet long messages. The k-binomial tree shows increased benefits with increase in number of packets. These results can be used in any kind of network (regular or irregular) which provides network interface support for multicast packetization.

The rest of the paper is organized as follows. Section 2 provides an overview of packetization and the network interface. Section 3 discusses two implementations of smart network interface support, and compares them. Section 4 presents the optimal multicast algorithm. Performance analysis of the optimal tree and some other trees is presented in Section 5. Finally, concluding remarks are made in Section 6.

2 Packetization and Network Interface Support

In this section, we discuss packetization of messages, conventional network interface support for transfer of packetized messages, and its drawback with respect to efficient multicast implementation. Then, we discuss a smart network interface support for multicast of long messages, and show that existing multicast algorithms are not optimal for systems with such support.

2.1 Packetization

Modern day networks typically limit the size of the largest packet. This is done to minimize network contention and support efficient utilization of buffer spaces in the network and at the network interfaces. If a node needs to send a long message to another node, the message is broken up into packets of fixed size. The message is sent as a sequence of packets, with each packet header carrying the routing information required for the message, and a packet sequence number. Typically, the maximum packet size is dictated by the design of the network and the associated communication protocol. For example, the packet size of the IBM SP2 is 256 bytes [1], and Fast Messages on Myrinet has a packet size of 128 bytes [24]. The sender is responsible for fragmenting the message and sending out the individual packets into the network. The packets are routed as individual messages to the destination depending on the routing information contained in the headers. At the destination, the receiver is responsible for collecting the packets, and assembling them into the complete message. Depending on the flow control scheme being used, the receiver also takes care of packets that might arrive out of order.

Support for packetization can provided at one of two levels in a typical high performance node: system level, or network interface level. System level support involves host processors at the sender and receiver sides executing some software to accomplish the above tasks. Such support is inefficient due to the large overheads of software execution, buffer copying, and loss of computing time at the host processor. To improve performance, modern network systems [1, 2, 4, 9] provide network interface associated with each node. Such support, which includes a coprocessor and a small amount of memory, implements the lower layers of the communication protocol.

2.2 Conventional Network Interface Support

Let us consider the conventional network interface support for message transfer in a typical high performance system. Figure 1 shows a generic network interface at a host node. The interface typically has a processor (to control the network interface), send and receive queues to store packets that are being sent to or received from the network, DMA engines to transfer the packets between the host memory and the send/receive queues, and other hardware to interface with the network. The processor at the network interface is independent of the host processor and can be programmed to execute the desired communication protocol.

An application is typically linked to a communication library in the host, and a portion of the host memory is allocated for DMA access to and from the network interface. In recent implementations of high performance messaging systems, there has been a trend of circumventing the operating system and providing applications direct access of the network device [12, 20, 24]. This has been done to reduce the send and receive overheads for messaging, so that the low latencies and high bandwidths required for cluster computing can be achieved. The programmable coprocessor at the network interface controls the actual sending and receiving of the messages. These network interface coprocessors are typically slower than host processor

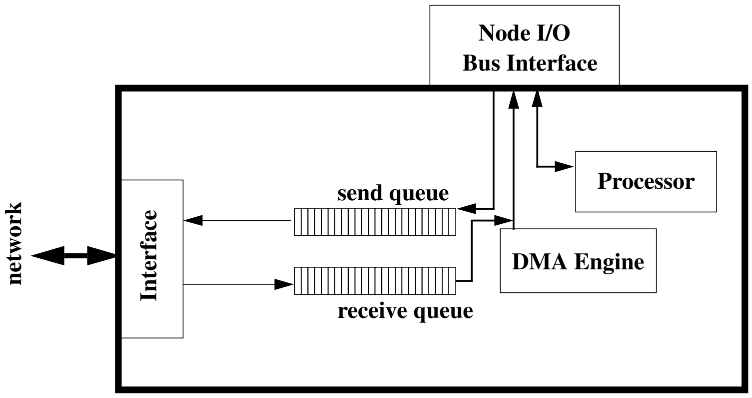

Figure 1: Generic diagram of a typical network interface at a host node.

to ensure inexpensive network interface cards (NICs).

A typical message transfer in these systems is done in the following way. At the sender side, one of two schemes can be used. One, the library at the sender host processor can fragment the message into fixed size packets, and transfer them to the send queue of the network interface [24]. Alternatively, the host processor can copy data into the DMA region, and write the message pointers to the network interface for the coprocessor to DMA the packets to the send queue [12]. Subsequently, the software executing at the coprocessor detects entries in the send queue, and sends the packets out to the network channel. The first scheme avoids an extra buffer copy of the data and hence reduces message latency, whereas the second scheme reduces the burden on the host processor, thus saving computing time. At the receiver side, the incoming packets join the receive queue at the network interface. The coprocessor detects the received packets and DMAs them to the host memory. The discussions in this paper are applicable to either scheme at the sender side.

2.3 Multicasting over Conventional Network Interface

If a typical multicast tree is built on such network interface support, some destinations serve as intermediate sources. This means that when they receive a message, they forward copies of the multicast message to other destinations. In the case of a multicast of a small message (which is less than the maximum packet size), a binomial tree can be used for optimal performance. Let us look at the forwarding process at the intermediate node in the case of a multicast of a long message which spans multiple packets. Figure 2 shows the forwarding of a two packet multicast message at an intermediate node of a multicast tree. All the packets of the message are received, and DMAed to host memory. The host processor at the intermediate node receives the complete message, and then initiates send operations to each of its children in the multicast tree. For each of the send operations, a copy of the message is sent to the network interface, where it is sent into the network. Therefore, an intermediate node undergoes the complete message overhead for every copy of the message that it forwards to other destinations. This overhead includes the software start-up overhead and the overhead at the network interface for each packet transmission. Smart network interface support can reduce this overhead for multicast of long messages.

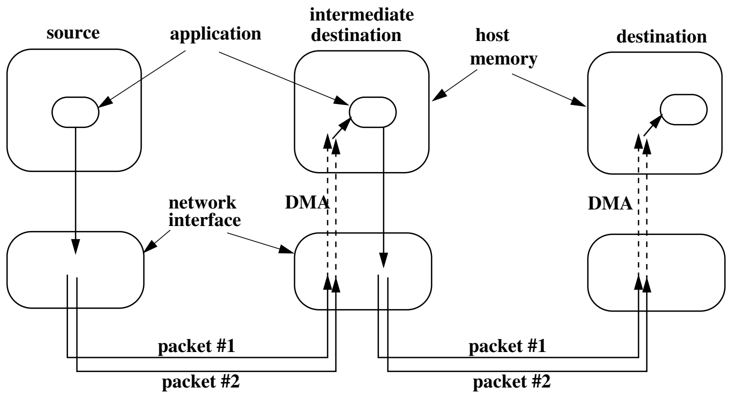

Figure 2: Forwarding of a 2-packet long multicast message by intermediate node using conventional network interface support.

2.4 Smart Network Interface Support

Here, the multicast tree forwarding process is completely handled by the software running at the network interface processor. The software running at the network interface processor is given the capability to identify a multicast packet. If the next outgoing packet in the send queue is a multicast packet, the network interface processor forwards replicas of the packet to the nodes adjacent to the root of the multicast tree. If a multicast packet is received, the network interface processor, after it starts the DMA of the packet to host memory, forwards replicas of the packet to its children in the multicast tree. Figure 3 shows the forwarding of a two packet multicast message at an intermediate node of a multicast tree using smart network interface support.

2.5 Multicasting over Smart Network Interface

Let us model the latency of a multicast operation using smart network interface support. The software start-up overhead, t_s, is incurred once at the host processor of source of the multicast to transfer the data to the network interface memory. Consequently, the multicast tree is implemented at the network interfaces of the participating processors. The host processor

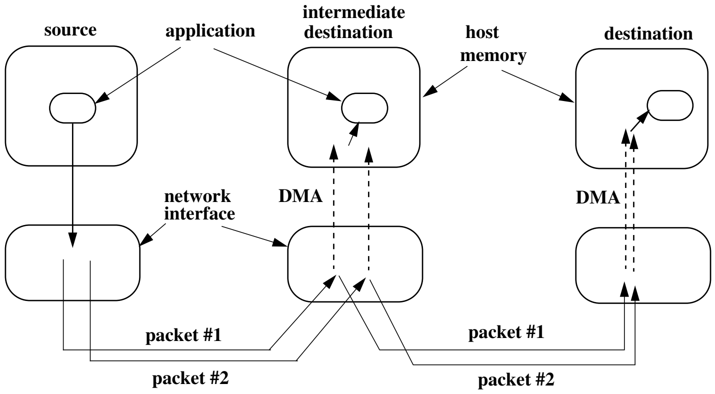

Figure 3: Forwarding of a 2-packet long multicast message by intermediate node using smart network interface support.

at each destination undergoes the software overhead, t_r, for receiving the message when the network interface DMAs the received packets to the host memory and sets the appropriate flags. Although, these software overheads are large, they are independent of the choice of the multicast tree. The overhead incurred by the network interface processor at each of the destinations for receiving a packet, identifying it as a multicast packet, and forwarding it to the appropriate destination depends on the choice of the multicast tree. Therefore, the latency of a multicast tree is determined in terms of the actual transmission of the packets of the multicast message to the network interfaces of the destinations. In the remainder the paper we refer to the transmission of a packet from the network interface of a processor to the network interface of another processor as a *communication step*. This includes the overhead at the sender network interface for sending a packet, propagation overhead, and the overhead at the receiver network interface for receiving the packet. Let this overhead be denoted as t_{step}.

Let us take a simple example of a single packet multicast using a binomial tree over three destinations to illustrate the reduction in multicast latency with the smart network interface support. Figure 4(a) shows the multicast over the conventional network interface, and Fig. 4(b) shows the multicast over the smart network interface. It can be easily observed by adding up the latencies along the critical paths in the figures that the multicast latency over the conventional network interface is $2(t_s + t_{step} + t_r)$, whereas the multicast latency over the conventional network interface is $t_s + 2t_{step} + t_r$. For an arbitrary multicast set size of n these values will be $\lceil log_2 n\rceil(t_s + t_{step} + t_r)$ and $t_s + \lceil log_2 n\rceil t_{step} + t_r$ over the conventional and smart network interfaces, respectively. Figure 4(c) compares the multicast latency obtained using a binomial tree over the two network interfaces for a single packet multicast across a range of multicast set sizes. The values of t_s and t_r were set to be 10 units, and the ratio between t_s and t_{step} was assumed to be 2 and 4 units for these comparisons. It is clear from these results that the smart network interface reduces multicast latency.

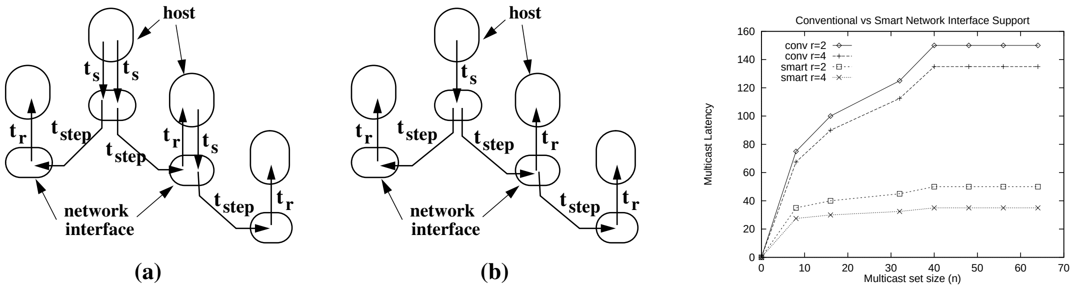

Figure 4: Performance benefits of the smart network interface: (a) binomial single-packet multicast tree over the conventional network interface, (b) binomial single-packet multicast tree over the smart network interface, and (c) comparison of multicast latency over the two network interfaces for a single packet multicast with $t_s = t_r = 10$ units, and $r = \frac{t_s}{t_{step}} = 2$ and 4.

This improvement in performance over the smart network interface is due to two main reasons. First, the host processor at the intermediate node is not involved in the forwarding of multicast packets. This means that, from the time one packet arrives at the network interface until it gets forwarded there is no extra overhead of interrupting the host processor, copying the data into host memory, and recopying the data in the send queue of the network interface. Second, an intermediate node can forward a packet of the message as soon as it arrives, even though the complete message has not arrived. This means that the first packet gets forwarded to other destinations even while the second packet is being received.

Such network interface support requires longer buffering of multicast packets at the network interface. This is because each packet is forwarded to multiple destinations, and the packet data requires buffering at the network interface until the network interface processor has injected all required copies of it into the network. Therefore, there is need for larger buffering at the network interface. An example of such an implementation of a smart network interface has been described in [28]. Although this paper describes the buffering policy and flow control it does not propose any optimal multicast algorithm over such a network interface.

2.6 Optimal Multicast Trees over Smart Network Interface

Although we showed in Section 2.5 that the smart network interface reduces multicast latency, there is still the question about whether the binomial tree is the optimal multicast tree for multi-packet multicast over such smart network interface. To answer this question let us take an example of a multicast of a 3-packet long message to three destinations. Figure 5(a) shows the number of steps taken to complete such a multicast using a binomial tree, and Fig. 5(b) shows the number of steps using a linear tree. In the figures, the number in brackets indicates the step number, and the subscript indicates the packet number being transmitted. For example $[4]_2$ indicates the second packet being transmitted in the fourth time step. It can be easily observed that the binomial tree takes 6 steps and the linear tree takes 5 steps. The multicast latency for the binomial tree is $t_s + 6 * t_{step} + t_r$, and the multicast latency for the linear tree is $t_s + 5 * t_{step} + t_r$. The t_r term denotes the software receiving overhead at the host processor of the last destination to get all the packets of the multicast message. This simple example shows that the binomial tree is not the optimal tree for multicast of packetized messages with smart network interface support. Thus, there is a need for designing optimal multicast algorithms for long messages with such network interface support.

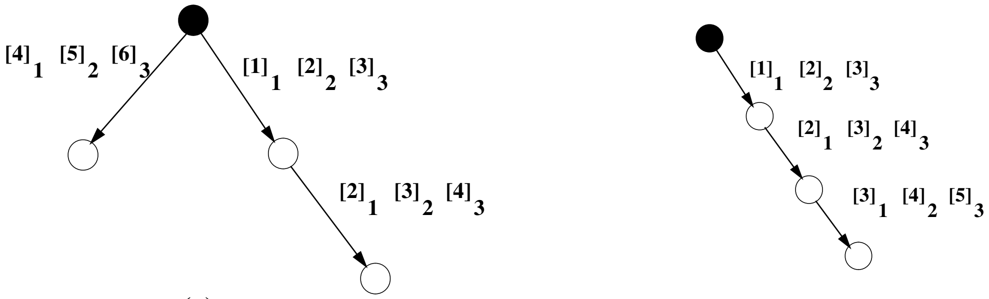

Figure 5: The number of steps to complete multicast of a 3-packet long message to 3 destinations using: (a) the binomial tree, and (b) the linear tree.

3 Implementations of Smart Network Interface Support

In this section, we describe two implementations of smart network interface support for multicast. These implementations are: First-Child-First-Served (FCFS) and First-Packet-First-Served (FPFS). We compare both implementations, and show why the FPFS implementation is more efficient and practical.

3.1 First-Child-First-Served (FCFS) Implementation

In this implementation, the intermediate node forwards the multicast message to its first child, and then to the second child, and so on. When the network interface of an intermediate node receives the first packet of a multicast message, it forwards the packet to to its first child. When the second packet of the multicast message arrives at the network interface, it also forwards this packet to the first child. Similarly, the complete multicast message is forwarded packet by packet to the first child. Subsequently, the network interface forwards the message packet by packet to the second child, and so on. The network interface at the source node sends all the packets to the first child of the source, then sends all the packets to the second child of the source, and so on. Figure 6 formally expresses this implementation in a pseudo-code format. It should be noted that these pseudo-code are executed completely at the network interface processor.

The FCFS Implementation

Sender

```
for i = 1 to num_children {
    for j = 1 to num_packets {
        send(child_i,packet_j);
    }
}
```

Receiver with Forwarding (Intermediate Node)

```
for i = 1 to num_children {
    for j = 1 to num_packets
    {
        if i == 1
            receive(packet_j);
        send(child_i,packet_j);
    }
}
```

Receiver

```
for j = 1 to num_packets {
    receive(packet_j);
}
```

Figure 6: Pseudo-code description of the FCFS implementation of the smart network interface for multicast.

3.2 First-Packet-First-Served (FPFS) Implementation

In this implementation the network interface forwards the message on a per-packet basis. When the first packet of the multicast message arrives at the network interface of an intermediate node, it forwards the packet to each of the children of the intermediate node. Subsequently, when the second packet of the multicast message arrives at the network interface, it forwards the packet to each of the children. Similarly, the network interface forwards each packet till the last packet of the message is forwarded. The network interface at the source node sends the first packet to all the children of the source, then sends the second packet to all the children of the source, and so on. Figure 7 formally expresses this implementation in a pseudo-code format. Again, it should be noted that these pseudo-code are executed completely at the network interface processor.

3.3 Comparison of FCFS and FPFS Implementations

We evaluate and compare these two implementations of smart network interface support with respect to ease of implementation, buffer requirement, and the software overhead at the network interface processor.

The FPFS Implementation

Sender

```
for j = 1 to num_packets {
    for i = 1 to num_children {
        send(child_i, packet_j);
    }
}
```

Receiver with Forwarding (Intermediate Node)

```
for i = 1 to num_packets {
    receive(packet_i);
    for j = 1 to num_children
    {
        send(child_j,packet_i);
    }
}
```

Receiver

```
for i = 1 to num_packets {
    receive(packet_i);
}
```

Figure 7: Pseudo-code Description of the FPFS implementation of the smart network interface for multicast.

3.3.1 Ease of Implementation

The FPFS implementation is easier to implement in the software running at the network interface processor than the FCFS implementation. Let us assume that there are packets of multiple messages coming into the receive queue of the network interface. The FCFS implementation needs to maintain a counter for each of the incoming messages. Each arriving packet increments the counter corresponding to its message. When a counter value becomes equal to the message length, the software sends the packets to the remaining children. On the other hand, the FPFS implementation handles the forwarding of the multicast message on a per-packet basis. When the network interface processor reads the header of a multicast packet from the receive queue, it forwards the packet to all the children in the multicast tree. Thus, the FPFS implementation does not require to maintain a counter for each incoming multicast message. Therefore, the FPFS mechanism lends itself to easier implementation than the FCFS mechanism.

3.3.2 Buffer Requirement at the Network Interface

It can be quantatively shown that the FPFS implementation is more efficient than the FCFS implementation in terms of buffer requirement. Let us take an example of an intermediate node with k children in the multicast tree of a p-packet multicast. Let us take the following simple model of the network interface: t_{rq} is the time for a packet to reach the receive queue from the network adaptor, and t_{sq} is the time for a copy of a packet to be sent out from the queue to the network adaptor. Let T_{fcfs} denote the time in the FCFS implementation from when the network interface processor reads the ith incoming packet from the receive queue till when the ith packet does not require buffering at the network interface. Similarly, let T_{fpfs} denote the corresponding time in the FPFS implementation.

In the FCFS implementation, let us assume the best case that when the ith packet is read by the network interface processor the remaining $(p-i)$ packets of the multicast message have already arrived at the receive queue. Then, the ith packet has to be buffered till the ith packet and the remaining $(p-i)$ packets are forwarded to the first child of the intermediate node, all p packets are forwarded to the next $(k-2)$ children, and the first i packets are forwarded to the kth child. Equation 1 shows expression for T_{fcfs}. In the FPFS implementation, from the time when the ith packet is read by the network interface processor it has to be only buffered till it is forwarded to the k children of the intermediate node. Equation 2 shows expression for T_{fpfs}.

$$T_{fcfs} = (p-i+1)t_{sq} + (k-2)pt_{sq} + it_{sq} \tag{1}$$

$$T_{fpfs} = kt_{sq} \tag{2}$$

Here we have assumed best case conditions of zero delay between incoming packets. If there is delay between incoming packets, each packet will have to be buffered for a longer time in the FCFS implementation. It can be easily observed that even with best case assumptions, each packet requires buffering for a longer time with the FCFS implementation compared to the FPFS implementation. This translates to larger buffer requirement for the FCFS implementation as compared to the FPFS implementation.

3.3.3 Software Overhead at the Network Interface Processor

It can also be shown that the FPFS implementation is more efficient than FCFS implementation in terms of software overhead at the network interface. The process of forwarding a packet can be approximately broken up into two parts - the first part involves reading the packet header, and the second part involves preparing the outgoing packet. Let t_h and t_p denote the overheads for these parts, respectively. In the FPFS implementation, each packet header is read only once and its copies are created. In the FCFS implementation, each packet is read multiple number of times, depending on the number for children the processor has in the multicast tree. For example, a processor with k children in the multicast tree undergoes an overhead of $(t_h + t_p)k$ units using FCFS implementation, and an overhead of $t_h + kt_p$ units using the FPFS implementation. Therefore, the FPFS implementation is more efficient in terms of latency.

The above discussion shows that the FPFS network interface support is more practical and efficient than the FCFS network interface support. The challenge is whether we can develop optimal multicast trees for systems using the FPFS network interface support. We take on this challenge in the following section.

4 Optimal Multicast

In this section, we propose an optimal multicast tree on a system with FPFS network interface support. We also show how these trees can be constructed in a contention free manner if all the nodes in the system can be ordered in a contention free manner.

4.1 Multicast Latency Model

Let us first see the components of the latency of a packetized multicast using the FPFS smart network interface support. At the beginning of the multicast, the host processor at the source of the multicast undergoes a software start-up latency to send the message to its network interface. Subsequently, the network interface of each node (participating in the multicast) forwards the multicast packets to the destinations. Finally, the host processor of the last destination to receive the last packet of the multicast message incurs a software latency to receive the complete message. Therefore, the multicast latency can be written as a sum of three terms: $(t_s + num\_steps * t_{step} + t_r)$. The terms t_s and t_r constitute the constant components of the overhead for any multicast algorithm. Therefore, an optimal multicast tree is one which produces the minimum latency incurred at the network interface to deliver all the packets of the multicast message to all the destinations. The tree must be optimal for an any number of packets and any size of the multicast. In the following discussion, as mentioned earlier in Section 2.4, we refer to the time required to send a packet from the network interface of one node to another as one communication step.

4.2 Multicast Using FPFS Network Interface Support

Let us model the multicast latency using the FPFS network interface support. If we consider each packet of the multicast message individually, it can be easily observed that the complete multicast can be treated as a sequence of single packet multicasts following one another. Figure 8 shows the break up of the multicast of a 3-packet message over 7 destinations over a sample binomial multicast tree. Figure 8(a) shows the multicast of the first packet, Fig. 8(b) shows the multicast of the second packet, and Fig. 8(c) shows the multicast of the third packet. It can be easily observed that the 3-packet multicast is equivalent to three single packet multicasts which lag one after the other by three steps.

Let $T = (V, E)$ be a multicast tree, and let the function $\delta : V \rightarrow N$ be such that $\delta(v)$ is the number of children v has in T. Starting at time zero, let L_i denote the time at which

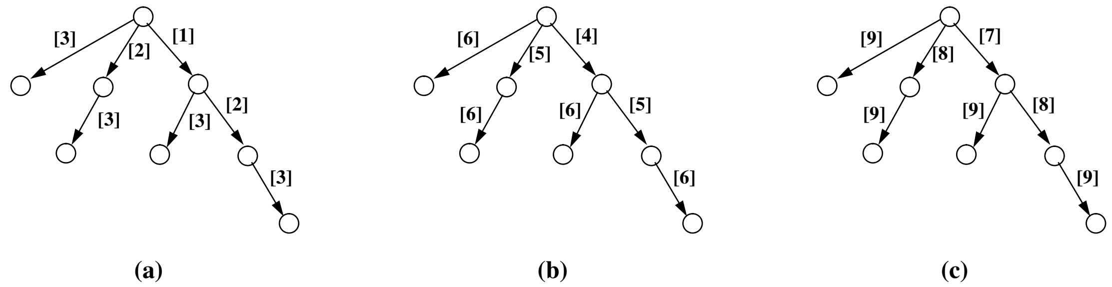

Figure 8: The break up of a 3-packet multicast over 7 destinations using a binomial multicast tree. The sequence of steps for (a) the first packet, (b) the second packet, and (c) the third packet of the multicast message.

the multicast of the ith packet is completed, i.e. the time at which the ith packet has been received by the network interface of each destination. Let the set $Children(v)$ be defined as the set of vertices which are the children of the node v in T, and let the set $Leaves(T)$ be the set of leaves in T. When designing multicast trees, it is a desired feature that a vertex in the multicast tree does not have a larger number of children than any of its ancestors. This is obvious since vertices higher up in the tree get the message earlier, and can therefore cover more destinations. We limit the following discussion to such trees.

Definition 1 *The function $delay_i(v)$ is defined as the time elapsed (in terms of number of steps) between the arrivals of the ith packet and the $(i+1)$th packet at the network interface of node v.*

Lemma 1 *For any vertex $v \in V$, and any vertex $c \in Children(v)$, the value of $delay_i(c)$ is given by the function $Max(delay_i(v), \delta(v))$.*

Proof: Let the ith packet arrive at the network interface of vertex v at time zero. The $(i+1)$th packet arrives at at time $delay_i(v)$ steps. The network interface of v sends the ith packet to each of its children in $\delta(v)$ steps. If $delay_i(v) \leq \delta(v)$, the network interface of v is available to send out a replica of the $(i+1)$th packet to the first child only at time $\delta(v)$. Therefore, $\forall x \in Children(v), delay_i(x) = \delta(v)$. If $delay_i(v) > \delta(v)$, although the network interface of v is available to send out the next packet at time $\delta(v)$, the $(i+1)$th packet has not arrived. The network interface of v can only send out a replica of the $(i+1)$th packet to its first child at time $delay_i(v)$. Therefore, $\forall c \in Children(v), delay_i(c) = Max(delay_i(v), \delta(v))$. ■

Lemma 2 *The time interval $(L_{i+1} - L_i)$, i.e. the time between the completions of multicast of any two successive packets of a multicast, is given by the function $Max_{v \in Leaves(T)}(delay_i(v))$, for a multicast tree $T = (V, E)$.*

Proof: The time at which the multicast of the ith packet is completed, L_i, is determined by the time at which all leaves of T have received the ith packet. The value of $delay_i(v)$ determines the interval between the arrivals of the ith and the $(i+1)$th packet at the leaf v. Therefore, the maximum delay between the adjacent packets seen at any leaf determines the difference $(L_{i+1} - L_i)$. ■

Theorem 1 *The time interval $(L_{i+1} - L_i)$, i.e. the time between the completions of multicast of any two successive packets of a multicast, for a multicast tree $T = (V, E)$ is given by the function $\delta(r)$ where r is the root of T.*

Proof: Since r is the source of the multicast $delay_i(r) = 0$. It can be easily observed that $\forall x \in Children(r), delay_i(x) = \delta(r)$. Similarly, if the $delay_i$ value is calculated for all vertices of T by iteratively applying the equation in Lemma 1, and then combined with the result in Lemma 2, it can be easily shown that the time $(L_{i+1} - L_i)$ is equal $\delta(v_m)$ where $v_m \in V$ and $\forall v \in V, \delta(v_m) \geq \delta(v)$. Since the discussion is restricted to trees with the property that no vertex has more children than any of its ancestors, $v_m = r$. Therefore, the time interval $(L_{i+1} - L_i)$ is given by $\delta(r)$. ■

Let L denote the time for completion of the multicast for the first packet. From the result of Theorem 4.2 it can be observed that the difference in time between the completions of multicast of successive packets of the multicast message, i.e. the time interval $(L_{i+1} - L_i)$, is independent of i. Let d denote this value. Therefore, an m-packet long multicast can be modeled as m single packet multicasts pipelined through the multicast tree.

Theorem 2 *The time for completion of these m single packet multicasts pipelined is $L + (m-1)d$ steps.*

Proof: Follows from the above discussion. ■

4.3 Deriving Optimal Multicast Tree

The optimal multicast tree will be one that produces the minimum value for the expression $L + (m-1)d$. Let us consider a multicast set (source and destinations) of size n. The number of children of the root node in the multicast tree determines the value of d. The binomial tree produces the minimum L, which is $\lceil log_2 n \rceil$. The value of d for such a tree is also $\lceil log_2 n \rceil$. If the value of d for the multicast tree is reduced, the value of L increases. In the case of the linear tree, $d = 1$ and $L = n - 1$. If the value of d is increased beyond $\lceil log_2 n \rceil$, the value of L also increases. Therefore, finding the optimal multicast tree for an m-packet multicast reduces to finding the d which produces the minimum value for the expression $L + (m-1)d$. It can also be easily observed that this optimal value of d lies in the interval $[1, \lceil log_2 n \rceil]$.

Definition 2 *A k-binomial tree is defined as a recursively doubling tree where no vertex has more than k children.*

In other words, the k-binomial tree describes a multicast where the number of nodes covered double recursively at every step, but no node covers more than k other nodes. It can be easily seen that for a given d, the d-binomial tree produces the minimum value of L. Figure 9(a) and 9(b) show examples of 3-binomial and 4-binomial trees with multicast set size of 16.

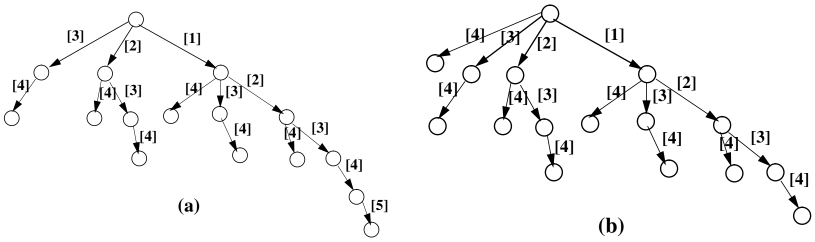

Figure 9: Examples of k-binomial trees on a multicast set size of 16: (a) the 3-binomial tree, and (b) the 4-binomial tree.

Lemma 3 *Let $N(s,k)$ denote the number of nodes (including the source node) that are covered in s steps by a k-binomial multicast tree. The function $N(s,k)$ can be determined as follows.*

$$N(s,k) = \begin{cases} 2^s & \text{if } s \leq k \\ 1 + N(s-1,k) + N(s-2,k) + \ldots + N(s-k,k) & \text{if } s > k \end{cases}$$

Proof: If $s \leq k$, the k-binomial tree is like a binomial tree, so $N(s,k) = 2^s$. For the case of $s > k$, Fig. 10 illustrates the structure of a k-binomial tree. It can easily observed that the root has k subtrees, and each of the subtrees is a k-binomial tree itself. Let the depth of the k-binomial tree be s. Then, the number of nodes in the k-binomial tree is obviously given by $N(s,k)$. Similarly, the number of nodes in the first subtree is is given by $N(s-1,k)$, the second tree is given by $N(s-2,k)$, and so on. Therefore, $N(s,k)$ is equal to the summation of the nodes in each of the subtrees, and one (the source). ∎

Theorem 3 *Given n and m, the k-binomial tree with the minimum value of $L + (m-1)k$ is the optimal multicast tree where $1 \leq k \leq \lceil log_2 n \rceil$ and L is equal to the minimum value of s for which $N(s,k) \geq n$.*

Proof: Follows from the results of Lemma 3 and Theorem 2.

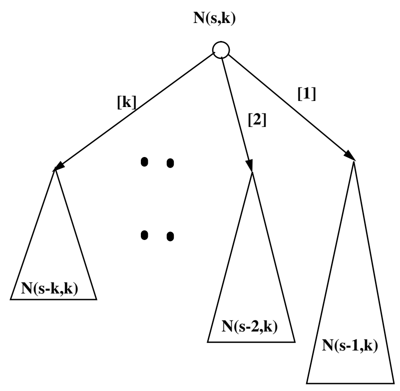

Figure 10: The number of nodes covered by a k-binomial tree in s steps when $s > k$. The number of nodes in the ith subtree from right is given by $N(s-i,k)$.

4.4 Constructing Contention-free k-binomial Trees

For optimal multi-packet multicast performance, the multicast tree should be *depth contention-free* [22]. This means that the paths that the tree edges get mapped to in the network should be edge-disjoint with respect to each other. The concept of a contention free ordering of nodes in a system has been used to construct contention-free binomial trees [22]. The same concept can be used to construct contention-free k-binomial trees.

Let the $N(s,k)$ participating nodes of a multicast be ordered in a contention free ordered chain. Let the symbol $<_d$ denote the ordering. A contention free ordering has the following property. If $\exists w, x, y, z$ in the ordered chain such that $w \ <_d \ x \ <_d \ y \ <_d \ z$, then messages between processors w and x will not contend for any links with messages between processors y and z. Without loss of generality, let us assume that the source of the multicast is the first node in the ordering. Figure 11 gives a pictorial representation of the construction of a contention free, k-binomial tree on this ordering in a recursive manner. In the first step, the source sends the message to the node, a, which is $N(s-1,k)$ places from the right end of the chain. In the second step, the source sends the message to the node, b, which $N(s-2,k)$ from the previous recipient. Similarly, the source sends messages to each of k nodes. The intermediate nodes, like a and b cover the destinations to their right by building k-binomial trees in a recursive fashion.

This construction can be applied to different types of contention-free ordered chains. For k-ary n-cubes, the dimension-ordered chain [22] can be used to construct contention-free k-binomial trees. For irregular switch-based networks with no complete contention-free ordering, the concept of a POC [15] can be used to construct k-binomial trees with minimal contention. For the performance evaluation section, we assume the existence of a complete contention-free ordering among the nodes of the system.

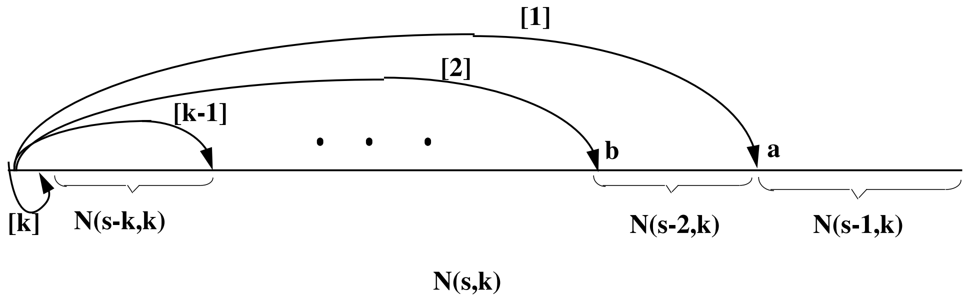

Figure 11: The construction of contention-free k-binomial tree from a given contention free ordering of participating nodes.

5 Performance Analysis

We performed experiments to study the variation of optimal value of k with change in multicast set size, n, and number of packets in the multicast message, m. We compared the multicast latency obtained using optimal k-binomial trees with the multicast latency obtained using binomial trees. For all experiments, we assumed system and technological parameters representing the current trend in technology. The following default parameters were used: t_s (software start-up overhead at host processor of sender) = 12.5 microseconds, t_r (software overhead at host processor of receiver) = 12.5 microseconds. We assumed a 80 Mbytes/sec network, with 256 bytes packet size, t_{ns} (overhead at network interface for sending a packet) = 2 microseconds, and t_{nr} (overhead at network interface for receiving a packet) = 2 microseconds. This resulted in t_{step} (time for a packet transfer from the network interface of one node to the network interface of another node) $= 2.0 + \frac{256}{80*10^6} + 2.0 = 7.2$ microseconds. The experiments were done on a 64 node system.

5.1 Optimal k

We studied the variation of the optimal value of k with change in multicast set size, n, and number of packets, m. The value of m was first fixed, and the multicast set size was varied. Then, the value of n was fixed, and the number of packets in the multicast message was varied. Figure 12(a) and Fig. 12(b) show the results of these experiments, respectively. It can be observed from Fig. 12(a) that for $m = 1$, the optimal value of $k = \lceil log_2 n \rceil$. As the value of m is increased, the value of k comes down. At a certain point, $k = 1$ becomes optimal, i.e. the linear tree becomes optimal. For $k = 1$, the multicast latency $ML_1 = (n-1) + (m-1)$, and for $k = 2$, the multicast latency $ML_2 = O(logn) + (m-1) * 2$. This crossover point happens at the minimum m for which $L_2 > L_1$. It can be easily seen that smaller the value of n, the smaller the value of m at which $L_2 > L_1$. This can be seen in Fig. 12(a) where the optimal

k for multicast set size 16 becomes 1 before multicast set size 32. It can also be observed in Fig. 12(b) that for multicast messages of length of 4 or 8 packets, the optimal value of k is 2 as the multicast set size is increased. It can be analytically determined that if the multicast set is increased further, the optimal value of k is expected to increase.

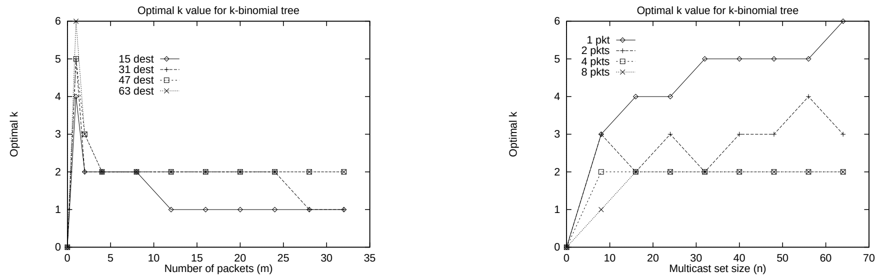

Figure 12: Variation of optimal value of k: (a) with fixed n and changing m, and (b) fixed m and changing n.

5.2 Multicast Latency

We studied the multicast latency for the k-binomial tree using the optimal values of k from the previous study. Again, the value of m was fixed, and the multicast set size was varied. Then, the value of n was fixed, and the number of packets in the multicast message was varied. Figure 13(a) and Fig. 13(b) show the results of these experiments, respectively. The slopes of the curves in Fig. 13(a) are constant in the region where the value of k is constant in the corresponding curves in Fig. 12(a). The slopes for the curves for multicast set sizes 16 and 32 change at the point when the optimal value of k becomes 1 in the corresponding curves in Fig. 12(a).

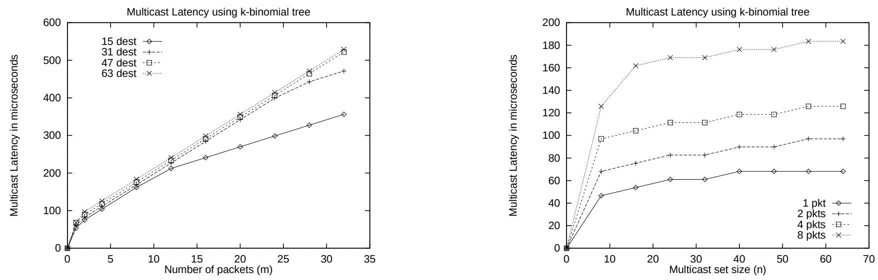

Figure 13: Multicast latency using the optimal k-binomial tree: (a) with fixed n and changing m, and (b) fixed m and changing n.

Next, we compared the performance of the k-binomial trees with the standard binomial trees. Figure 14(a) and Fig. 14(b) show the results of the comparison. It can be clearly observed in Fig. 14(a) that the performance of the k-binomial tree is better by a factor of upto 3 when compared to the binomial tree for message length of 24 packets. It can also be observed that with increase in number of packets in the message, the performance improvement of k-binomial tree over the binomial tree increases. This can also be seen by the relative differences in the k-binomial and binomial curves for 2 and 8 packet message length in Fig. 14(b).

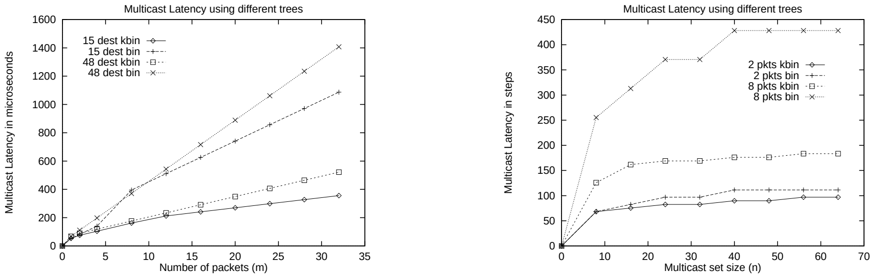

Figure 14: Comparison of multicast latencies using k-binomial and binomial trees: (a) with fixed n and changing m, and (b) fixed m and changing n.

6 Conclusions

In this paper we have analyzed the features of a typical network interface support, and established the differences between conventional and smart network interfaces that support multicasting. Two implementations, the FCFS and the FPFS, of smart network interface have been described in detail. We have shown that the FPFS is more practical and efficient than the FCFS implementation. Next, we have shown that the binomial tree is not optimal for an arbitrary multicast set size and an arbitrary number of packets using the FPFS implementation. We have modeled the multicast latency over FPFS network interface support. Then, we have introduced a new concept of k-binomial tree and have proved that the k-binomial tree is optimal for multi-packet multicast over the FPFS implementation of the network interface. Assuming existence of a contention-free complete ordering of the nodes in the system we have shown a method to construct contention-free k-binomial tree. The k-binomial algorithm has been experimentally evaluated and it compared very favorably with the conventional binomial tree. These results demonstrate significant potential to be applied to current and future generation of high performance systems with network interface support.

In this paper we have proposed an optimal algorithm for multicast on networks with network interface and packetization. It will interesting to see how this framework can be mapped to different network topologies and routing schemes in a contention-free manner. It will also be challenging to design optimal algorithms for multiple multicast with such packetization and network interface support.

Additional Information: A number of related papers and technical reports are available electronically through the home page of *Parallel Architecture and Communication* (PAC) research group. The URL is `http://www.cis.ohio-state.edu/~panda/pac.html`. Alternatively, the papers can be obtained by anonymous ftp from: `ftp.cis.ohio-state.edu, cd pub/communication`.

Acknowledgements: The authors would like to thank other members of the *Parallel Architecture and Communication* (PAC) research group in the department for providing comments, criticisms, and suggestions to this work.

References

[1] The SP2 Communication Subsystem. August 1994.

[2] ATM Forum. *ATM User-Network Interface Specification, Version 3.1*, September 1994.

[3] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. van de Geijn, and J. Watts. Interprocessor Collective Communication Library (Intercom). In *Scalable High Performance Computing Conference*, pages 357–364, 1994.

[4] N. J. Boden, D. Cohen, and et al. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, pages 29–35, Feb 1995.

[5] R. V. Boppana, S. Chalasani, and C. S. Raghavendra. On Multicast Wormhole Routing in Multicomputer Networks. In *Symposium on Parallel and Distributed Processing*, pages 722–729, 1994.

[6] Y. M. Boura and C. R. Das. Modeling Virtual Channel Flow Control in Hypercubes. In *International Symposium on High Performance Computer Architecture*, pages 166–175, 1995.

[7] C.-M. Chiang and L. M. Ni. Efficient Software Multicast in Wormhole-routed Unidirectional Multistage Networks. In *Symposium on Parallel and Distributed Processing*, 1995.

[8] L. De Coster, N. Dewulf, and C.-T. Ho. Efficient Multi-packet Multicast Algorithms on Meshes with Wormhole and Dimension-Ordered Routing. In *International Conference on Parallel Processing*, pages III:137–141, Aug 1995.

[9] Cray Research, Inc. *Cray T3D System Architecture Overview*, 1993.

[10] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[11] J. Duato. A Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 976–987, September 1995.

[12] T. V. Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *ACM Symposium on Operating Systems Principles*, 1995.

[13] Y. Huang and P. K. McKinley. Efficient Collective Operations with ATM Network Interface Support. In *Proceedings of the International Conference on Parallel Processing*, pages I:34–43, Chicago, IL, Aug 1996.

[14] S. L. Johnsson and C.-T. Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, pages 1249–1268, September 1989.

[15] R. Kesavan, K. Bondalapati, and D. K. Panda. Multicast on Irregular Switch-based Networks with Wormhole Routing. In *Proceedings of the Int'l Symposium on High Performace Computer Architecture (HPCA-3)*, 1997. accepted to be presented.

[16] R. Kesavan and D. K. Panda. Minimizing Node Contention in Multiple Multicast on Wormhole k-ary n-cube Networks. In *Proceedings of the International Conference on Parallel Processing*, pages I:188–195, Chicago, IL, Aug 1996.

[17] Y. Lan, A.-H. Esfahanian, and L. Ni. Multicast in Hypercube Multiprocessors. *Journal of Parallel and Distributed Computing*, 8:30–41, 1990.

[18] X. Lin and L. M. Ni. Multicast Communication in Multicomputer Networks. In *International Conference on Parallel Processing*, pages 114–118, 1990.

[19] X. Lin and L. M. Ni. Deadlock-free Multicast Wormhole Routing in Multicomputer Networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 116–124, 1991.

[20] R. P. Martin. HPAM: An Active Message Layer for a Network of HP Workstations. In *Proceedings of the Hot Interconnectes Symposium*, 1994.

[21] P. K. McKinley and D. F. Robinson. Collective Communication in Wormhole-Routed Massively Parallel Computers. *IEEE Computer*, pages 39–50, Dec 1995.

[22] P. K. McKinley, H. Xu, A.-H. Esfahanian, and L. M. Ni. Unicast-based Multicast Communication in Wormhole-routed Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1252–1265, Dec 1994.

[23] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[24] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of the Supercomputing*, 1995.

[25] D. K. Panda. Issues in Designing Efficient and Practical Algorithms for Collective Communication in Wormhole-Routed Systems. In *1995 Workshop on Challenges for Parallel Processing*, pages 8–15, 1995.

[26] D. K. Panda, S. Singal, and P. Prabhakaran. Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme. Technical Report OSU-CISRC-6/94-TR33, Dept. of Computer and Information Science, The Ohio State University, 1994.

[27] J. Y. L. Park, H. A. Choi, N. Nupairoj, and L. M. Ni. Construction of Optimal Multicast Trees Based on the Parameterized Communication Model. In *Proceedings of the International Conference on Parallel Processing*, Chicago, IL, Aug 1996.

[28] K. Verstoep, K. Langendoen, and H. Bal. Efficient Reliable Multicast on Myrinet. In *Proceedings of the International Conference on Parallel Processing*, pages III:156–165, Aug 1996.