Optimal Multicast with Packetization and Network Interface Support *

Ram Kesavan and Dhabaleswar K. Panda
Department of Computer and Information Science
The Ohio State University, Columbus, OH 43210-1277
Email: {*kesavan,panda*}*@cis.ohio-state.edu*

Abstract: *Modern networks typically limit the size of the largest packet for efficient communication. Thus, 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. This paper presents a concept of smart network interface support for packetization and an optimal multicast algorithm for systems with such support. Two implementations of smart network interface, First-Child-First-Served (FCFS) and 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 components of multicast latency under FPFS implementation are analyzed by using a pipelined model. A concept of k-binomial tree is introduced, and proved to be optimal for multicasting under the FPFS scheme. A method to construct contention-free k-binomial trees on contention-free orderings of the nodes is presented. For a 64-node system with irregular network, simulation results indicate that the optimal k-binomial tree is upto 2 times better than the conventional binomial tree for a range of multicast set sizes and message lengths. 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.*

1 Introduction

Multicast/broadcast is a common collective communication operation as defined by the MPI standard. 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. There have been many multicast/broadcast algorithms proposed in the literature in recent years [5, 6, 9, 11]. All these algorithms are designed assuming arbitrarily long single-packet messages. Typically, modern networks limit the size of the largest packet to minimize network contention and support efficient buffer utilization in the network and the network interfaces. Therefore, large messages are broken up and transmitted as multiple packets. Modern networks also 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 true of nodes on a NOW, as well as on systems like the IBM SP2.

Recently, a solution for implementing multicast under packetization has been proposed in [2]. However, this work assumes host processor handling of packetization and user/system control of determining optimal packet size for a given multicast set and message length. Thus, this result is not practical for modern systems with fixed packet lengths and network interface support for packetization. This leads to the challenge of designing optimal multicast algorithms under packetization using network interface support.

In this paper we take on this challenge. First, the features of conventional and smart network interfaces are analyzed. 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 calculate multicast latency a pipelined model of multi-packet multicast is built for the FPFS implementation. Then, a new concept of k-binomial tree is defined and shown to be optimal for multi-packet multicast. A method to construct contention-free k-binomial trees is developed using the concept of contention-free ordering of nodes [9]. The k-binomial tree is evaluated and compared to the conventional binomial tree on a 64-node irregular switch-based network using simulation experiments. Results show an improvement by a factor of 2 when using the k-binomial tree. Also, the benefit of using the k-binomial tree is shown to increase with increase in number of packets of a multicast. These results are quite general and 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 reviews packetization and network interface. Section 3 discusses FCFS and FPFS implementations of smart network interface support, and compares them. Section 4 presents the optimal multicast algorithm. Performance analysis results are presented in Section 5.Related work is described

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

in Section 6, and concluding remarks are made in Section 7.

2 Packetization and Network Interface Support

In this section, we discuss multicast of packetized messages over conventional and smart network interfaces, and show that existing multicast algorithms are not optimal on systems with smart network interface 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 buffer utilization in the network and the network interfaces. If a node needs to send a large message to another node, the message is broken up into packets of fixed size. Typically, the maximum packet size is dictated by the design of the network and the associated communication protocol. The sender fragments the message and sends 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. The destination collects the packets and assembles them into the complete message.

Communication support can be 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 provide a network interface associated with each node. The network interface, 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 contains a coprocessor, send/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.

An application is typically linked to a communication library in the host, and a portion of the host memory is allocated for DMA to and from the network interface. Recent implementations of high performance messaging systems show a trend of circumventing the operating system and providing applications direct access to the network device [3, 8, 10]. This reduces the send and receive overheads for messaging, so that the low latency and high bandwidth

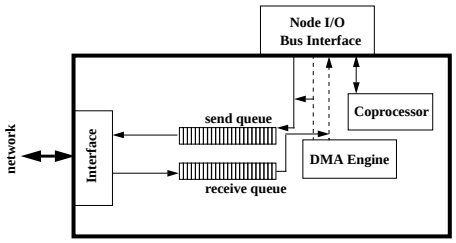

Figure 1: Generic diagram of a typical network interface at a host node.

requirements of cluster computing can be achieved. The programmable coprocessor at the network interface controls the actual sending and receiving of the messages.

A typical message transfer in these systems is done in the following way. At the sender side, one of two schemes is used. The host processor at the sender fragments the message into fixed size packets and transfers them to the send queue of the network interface [10]. Alternatively, the host processor copies data into the host DMA memory, writes the message pointers to the network interface, and the coprocessor uses DMA to copy the packets to the send queue [3]. Subsequently, the software executing at the coprocessor detects entries in the send queue, and sends the packets out to the network channel. At the receiver side, the incoming packets join the receive queue at the network interface. The coprocessor detects the received packets and uses DMA to copy them to the host memory. The discussions in this paper are applicable with either scheme at the sender side.

2.3 Multicasting over Conventional Network Interface

When a multicast tree is implemented on a high performance system, some destinations serve as intermediate sources. This means that when they receive a message, they forward copies of it to other destinations. Let us consider the multicast of a large message spanning multiple packets. Figure 2 shows the forwarding of a 2-packet multicast message at an intermediate node of a multicast tree. All the packets of the message are received at the network interface and copied to host memory using DMA. 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, from where it is sent into the network. Therefore, an intermediate node undergoes the message send 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 large messages.

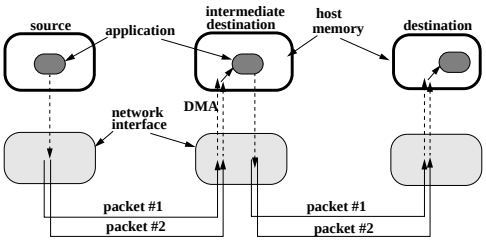

Figure 2: Forwarding of a 2-packet long multicast message by intermediate node using conventional network interface support.

2.4 Smart Network Interface Support

Here, the forwarding process at the source and intermediate nodes is completely handled by the software running at the network interface coprocessor. This software is given the capability to identify a multicast packet. If the next outgoing packet in the send queue of the source node is a multicast packet, the network interface coprocessor forwards replicas of the packet to the nodes adjacent to the root of the multicast tree. When a multicast packet is received at the network interface of an intermediate node, the network interface coprocessor, after it starts copying (using DMA) of the packet to host memory, forwards replicas of the packet to its children in the multicast tree. Figure 3 shows such forwarding.

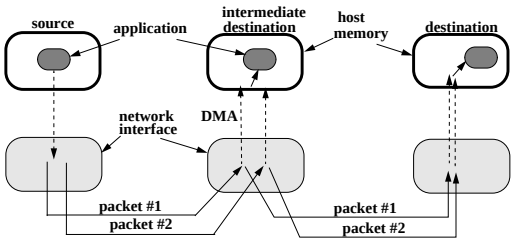

Figure 3: Forwarding of a 2-packet long multicast message by intermediate node using smart network interface support.

2.5 Multicasting over Smart Network Interface

Let us estimate 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 at each destination undergoes the software overhead, t_r, for receiving the message. Although, these software overheads are large, they are independent of the choice of the multicast tree. However, the overhead incurred at the network interfaces of the participating nodes depends on the choice of the multicast tree. Therefore, the latency of a multicast tree is determined by the time required for the actual transmission of all the packets of the multicast message to the network interfaces of the destinations. Hereafter, we refer to the transmission of a packet from the network interface of one processor to another as a *step*. This time, denoted t_{step}, 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 us take a simple example of a single-packet multicast using a binomial tree over three destinations to illustrate the advantage of using smart network interface support. Figures 4(a) and 4(b) show the multicast over conventional and smart network interface, respectively. It can be easily observed that the multicast latencies using conventional and smart network interfaces are $2(t_s+t_{step}+t_r)$ and $(t_s+2t_{step}+t_r)$, respectively. For an arbitrary multicast set size of n nodes, these values will be $\lceil log_2 n \rceil (t_s + t_{step}+t_r)$ and $(t_s+\lceil log_2 n \rceil t_{step}+t_r)$, respectively. Therefore, multicast latency can be lowered by using smart network interface support.

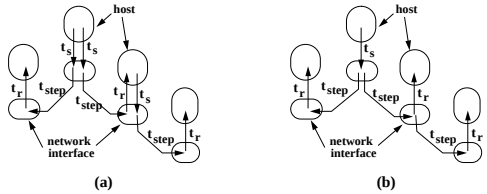

Figure 4: Performance benefits of the smart network interface: (a) binomial single-packet multicast tree over the conventional network interface and (b) binomial single-packet multicast tree over the smart network interface.

This improvement in performance is due to two main reasons. First, the host processor at the intermediate node is not involved in the forwarding of multicast packets, thereby reducing the forwarding overhead. Second, an intermediate node can forward a packet of the message as soon as it arrives, independent of the arrival of the remaining packets. As it can be observed, such network interface support requires 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 coprocessor has injected all required copies of it into the network. An example of the use of such smart network interface has been described in [12].

2.6 Optimal Multicast Trees over Smart Network Interface

Prior work in the literature has shown the binomial tree to be optimal (in terms of number of start-ups) for multicast on systems with conventional network interface support [9]. However, it is not clear whether this is true for systems with smart network interface. Let us consider an example multicast of a 3-packet long message to three destinations on a system with smart network interface support. Figures 5(a) and 5(b) show the number of steps taken to

complete such a multicast using a binomial and a linear tree, respectively. In the figures, the numbers in brackets indicate the step numbers and the subscripts indicate the packet numbers. 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 $(t_s + 5 * t_{step} + t_r)$. This simple example shows that the binomial tree is not the optimal tree for multicast of packetized messages with smart network interface support.

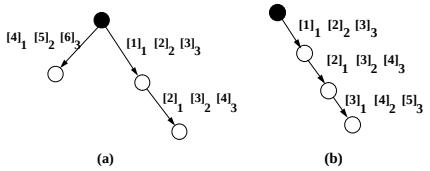

Figure 5: The number of steps to complete multicast of a 3-packet long message to 3 destinations using: (a) a binomial tree and (b) a linear tree.

3 Implementations of Smart Network Interface Support

In this section, we describe two implementations of smart network interface support for multicast: 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 network interface at the source node sends all packets of the multicast message to its first child in the multicast tree, then to its 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 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, one packet at a time, to the first child. Subsequently, the network interface forwards the message to the second child, followed by the third child, and so on. Figure 6 formally expresses this implementation in a pseudo-code format.

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

In this implementation, the network interface forwards the message on a per-packet basis. 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. 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, and so on till the last packet is forwarded. Figure 7 formally expresses this implementation in a pseudo-code format.

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

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

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

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

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

```
Receiver with Forwarding
(Intermediate Node)
for j= 1 to num_packets {
    receive(packet_j);
    for i= 1 to num_children {
        send(child_i, packet_j);
    }
}
```

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

3.3 Comparison of FCFS and FPFS Implementations

Let us evaluate and compare these two implementations of smart network interface support with respect to ease of implementation and buffer requirement.

3.3.1 Ease of Implementation

The FPFS is an easier implementation than the FCFS. Let us consider packets of multiple messages coming into the receive queue of the network interface at an intermediate node. To implement FCFS, the network interface processor has to maintain a counter for each incoming message. Each arriving packet increments the counter corresponding to its message. When the counter value becomes equal to the message length, all the packets are sent to the remaining children. To implement FPFS, the network interface processor 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. The processor does not have to maintain a counter for each incoming multicast message. Therefore, the FPFS is an easier implementation than the FCFS.

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 t_{ns} be the time for a copy of a packet to be sent out from the queue to the network adaptor. Let us consider the time interval starting from when the network interface coprocessor reading an incoming packet until all copies of this packet have been sent to its children. Let T_c and T_p denote this time interval for FCFS and FPFS implementations, respectively.

Let us assume the best case of zero time delay between incoming packets. In the FCFS implementation a packet needs to be buffered at the network interface of an intermediate node until all packets of the corresponding message have been forwarded to all the children of the node. Thus, the ith packet needs 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. This leads to $T_c = (p - i + 1)t_{ns} + (k - 2)pt_{ns} + it_{ns}$.

In the FPFS implementation, a packet only needs to be buffered at the network interface of an intermediate node until it has been forwarded to all the children of the node. Thus, the ith packet needs to be buffered only until it is forwarded to the k children of the intermediate node. This leads to $T_p = kt_{ns}$.

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

The above discussion shows that the FPFS implementation is a more practical and efficient approach. In the next section, we take on the challenge of developing optimal multicast trees for systems with such FPFS network interface support.

4 Optimal Multicast with FPFS

In this section, we propose an optimal multicast tree on a system with FPFS network interface support and discuss the related implementation issues.

4.1 A Pipelined Model for Estimating Multicast Latency

The discussion in Section 2.5 clearly shows that multicast latency for a single packet on a system with smart network interface support can be written as $(t_s + num\_steps * t_{step} + t_r)$. The same formula can be extended to multi-packet multicast latency where t_s (t_r) can denote the send (receive) overhead at the host processor to transfer all packets of the message to (from) the network interface. In this section, we analyze multicast latency in terms of $steps$ occuring at the network interface layer, as discussed in Section 2.5.

Let us model the multicast latency at the network interface layer assuming the FPFS implementation. The multicast of the complete message 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 to 7 destinations over an example binomial multicast tree. The numbers in brackets indicate the step numbers, and the subscripts indicate the packet numbers. It can be easily observed that the 3-packet multicast is equivalent to three single-packet multicasts where each packet lags the previous one by three steps.

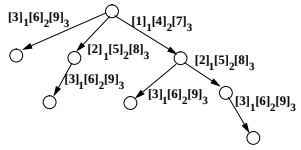

Figure 8: The break up of a 3-packet multicast over 7 destinations using a binomial multicast tree.

Let T be a multicast tree, and let δ_r be the number of children of the root of T. Let the multicast begin at time zero. Let L_i denote the time at which 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. Then, the following theorem can be derived.

Theorem 1 *The time interval $(L_{i+1} - L_i)$, i.e. the time between the completions of multicast of any two successive packets, for a multicast tree is given by δ_r.*

Proof: Due to lack of space, we are not able to present this proof here. Interested readers are requested to read [7] for the detailed proof. ■

From Theorem 1 it can be observed that the time interval $(L_{i+1} - L_i)$ is independent of i. Also, each successive packet completes its multicast δ_r steps after the completion of the previous one. Therefore, an m-packet multicast can be modeled as m single-packet pipelined multicasts. This leads to the following theorem.

Theorem 2 *The time for completion of these m pipelined single-packet multicasts is $L_1 + (m - 1)\delta_r$ steps.*

It can also be observed from Fig. 8 that the multicast of each packet lags the previous one by exactly 3 steps, which is equal to the number of children of the root. Also, the complete multicast takes 9 steps, which is $3 + (3 - 1) * 3$.

4.2 Deriving Optimal Multicast Tree

The optimal multicast tree is one that produces the minimum value for the expression $L_1 + (m-1)\delta_r$. Let us consider a multicast set of size n nodes. The value of δ_r in a multicast tree determines the value of L_1. In the case of a linear tree (Fig 5(b) for example), δ_r = 1 which leads to L_1 = $(n-1)$. If δ_r of a tree is increased, the value of L_1 decreases. In the case of the binomial tree [9] where $\delta_r = \lceil log_2 n \rceil$, L_1 reaches a minimum of $\lceil log_2 n \rceil$ since this tree recursively doubles the number of destinations covered in each step. However, on further increase of δ_r of a multicast tree beyond $\lceil log_2 n \rceil$, the value of L_1 increases. Therefore, for getting the minimum value for $L_1 + (m-1)\delta_r$, we need to only consider the interval $[1, \lceil log_2 n \rceil]$ to compute the optimal value of δ_r. If δ_r of a tree is less than $\lceil log_2 n \rceil$, we get the special case of a restricted binomial tree. Let us call this tree a k-binomial tree.

Definition 1 *A k-binomial tree is defined as a recursively doubling tree where each vertex has atmost k children, i.e. $\delta_r \leq k$.*

Figures 9(a) and 9(b) show examples of 3-binomial and 4-binomial trees with multicast set size of 16. To calculate the optimal value of δ_r which produces the minimum value for $L_1 + (m-1)\delta_r$, let us derive a relationship between L_1 and δ_r using the k-binomial tree.

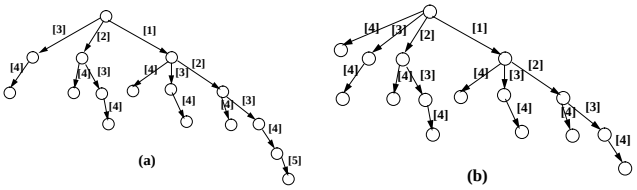

Figure 9: Examples of k-binomial trees on a multicast set size of 16: (a) the 3-binomial tree, and (b) the 4-binomial tree.

Lemma 1 *Let $N(s,k)$ denote the number of nodes covered in s steps by a k-binomial multicast tree. The value of $N(s,k)$ is given by*

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

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 after s steps. The root has k subtrees, and each of the subtrees is recursively a k-binomial tree. It can be seen that after s steps the number of nodes in the first subtree is given by $N(s-1,k)$ since the depth of this subtree is $(s-1)$. Similarily, the number of nodes in the second subtree 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). ■

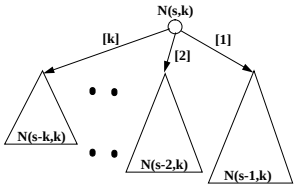

Figure 10: The number of nodes covered by a k-binomial tree in s steps when $s > k$. The number of nodes in the ith subtree from right is given by $N(s-i,k)$.

Thus, for a given δ_r and n, the value of L_1 is the minimum value of s such that $N(s,\delta_r) \geq n$. Using this relation, the optimal multicast tree for a given n and m can be calculated as follows.

Theorem 3 *Given n and m, the k-binomial tree which produces the minimum value of $L_1 + (m-1)k$ is the optimal multicast tree, where $1 \leq k \leq \lceil log_2 n \rceil$, and L_1 is equal to the minimum value of s for which $N(s,k) \geq n$.*

4.3 Implementation Issues

There are two major issues for implementing k-binomial trees for packetized multicast in a given system. These issues are: a) computing the optimal value of k for given n and m, and b) constructing contention-free k-binomial multicast trees on the interconnection network of the system.

4.3.1 Computing Optimal k

For given n and m, it can be shown using Theorem 3 and Lemma 1 that there is no closed form solution for the optimal value of k which produces the minimum value for $L + (m-1)k$. However, this value can be easily computed by checking all possible values of k in the interval $[1, \lceil log_2 n \rceil]$ Thus, the optimal value of k can be precomputed and stored in a table for all possible values of n and m. As we will see in Section 5.1, the optimal value of k is identical for a range of m values and the optimal value of k converges to 1 with increase in m. Thus, this table requires less than $O(mn)$ memory. Therefore the precomputation of the optimal value of k for n and m is a feasible implementation.

4.3.2 Constructing Contention-free k-binomial Trees

For optimal multi-packet multicast performance, the multicast tree should be *depth contention-free* [9]. 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 contention-free ordering of nodes in a system has been used to construct contention-free binomial trees [9]. A similar approach can be used to construct contention-free k-binomial trees.

Let the n participating nodes of a multicast be ordered, and let the symbol $<_d$ denote the ordering. An ordering is said to be contention-free if $\forall w,x,y,z$ in the ordered chain such that $w <_d x <_d y <_d z$, messages between

processors w and x do 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, where s is computed from Theorem 3. In the second step, the source sends the message to the node, b, which is $N(s-2,k)$ places away from the previous recipient. Similarly, the source sends messages to $k-2$ other nodes. The intermediate nodes, like a and b cover the destinations to their right by building k-binomial trees in a recursive fashion.

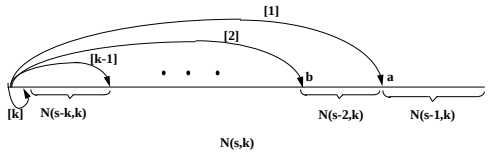

Figure 11: The construction of contention-free k-binomial tree from a given contention-free ordering of participating nodes.

This construction can be applied to different types of systems. For k-ary n-cubes, the dimension-ordered chain [9] can be used to construct contention-free k-binomial trees. For irregular networks, we have recently shown that no contention-free ordering exists for *up*/down** routing[5]. A concept of Partial Ordered Chain (POC) has been proposed to create an ordering with minimal contention on these networks. Such ordering can be used to construct k-binomial trees with minimal contention on irregular switch-based networks.

5 Performance Analysis

In this section, we present the results which show the behavior of the optimal value of k with varying n (multicast set size) and m (number of packets in the multicast message). We also present simulation results comparing the performance of the k-binomial tree with the binomial tree for multicast.

5.1 Optimal k

We analytically studied the variation of the optimal value of k with change in multicast set size, n, and number of packets, m. We conducted two experiments. 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. Figures 12(a) and 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. After a certain point, $k = 1$ (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 $ML_2 = O(logn) + 2(m-1)$. A crossover occurs at the minimum value of m for which $ML_2 > ML_1$. It can be easily seen that smaller the value of n, the smaller the value of m at which $ML_2 > ML_1$. This can be seen in Fig. 12(a) where the optimal k for multicast set size 16 (number of destinations = 15) 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. Even for messages with 4 or 8 packets, it can be analytically determined the optimal value of k increases with increase in n beyond 64.

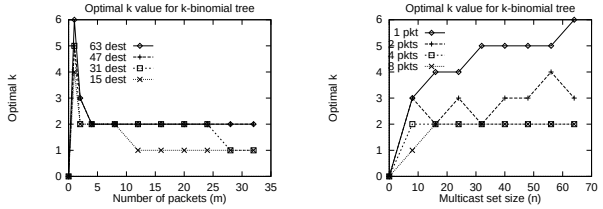

Figure 12: Variation of the optimal value of k: (a) with fixed n and changing m, and (b) fixed m and changing n.

5.2 Multicast Latency

For comparing multicast latency of k-binomial trees with binomial trees, we simulated these algorithms an irregular switch-based network with 64 processors connected by 16 eight-port switches. 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 64 bytes packet size, t_{ns} (overhead at network interface for sending a packet) = 3.0 microseconds, and t_{nr} (overhead at network interface for receiving a packet) = 2.0 microseconds. For each data point, the multicast latency was averaged over 30 different random sets of destinations for each of 10 different random network switch interconnection topologies. We used the Chain Concatenated Ordering (CCO), described in [5], as the base ordering for irregular switch-based networks.

First, we studied the multicast latency for the k-binomial tree (using the optimal values of k from the previous study) for varying m and n. Figures 13(a) and 13(b) show the results of these experiments, respectively. It can be observed that increase in multicast latency is less when the corresponding optimal value of k reduces. In Fig. 13(a), the slope for the graph for 15 destinations reduces when $k = 1$. Similarly, the slopes of the graphs in Fig. 13(b) reduce when k converges to 2.

Next, we compared the performance of the k-binomial trees with the standard binomial trees. Figures 14(a) and

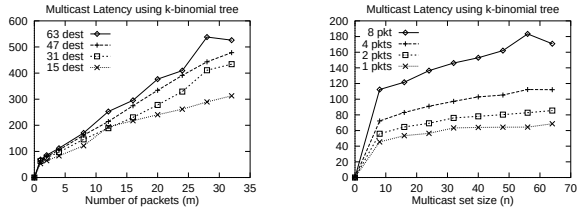

Figure 13: Multicast latency (in microseconds) using the optimal k-binomial tree: (a) with fixed n and changing m, and (b) fixed m and changing n.

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 up to 2 when compared to the binomial tree. From Fig. 14(b), 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.

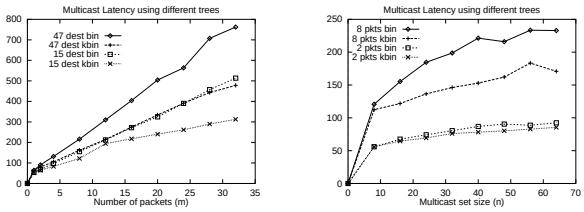

Figure 14: Comparison of multicast latencies (in microseconds) using k-binomial and binomial trees: (a) with fixed n and changing m, and (b) fixed m and changing n.

6 Related Work

A recent work [4] studies the efficient implementation of collective communication operations on a NOW over ATM using the network interface support. Since ATM switches provide hardware multicast capability, the network interface support in this work is primarily geared towards achieving reliable multicast over the unreliable ATM layer. Another recent work [12] describes an implementation of packetized multicast over the Myrinet network interface [1]. This work is also geared towards development of a reliable multicast communication layer and does not provide any formal multicast algorithms.

7 Conclusions

In this paper we have analyzed the features of network interface to support packetized multicast. Two implementations, FCFS and FPFS, using smart network interface support have been presented. The FPFS scheme has been shown to be more practical and efficient than the FCFS. 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. Components of multicast latency using FPFS implementation have been analyzed. A new concept of k-binomial tree has been introduced and it has been proved that the k-binomial tree is optimal for multi-packet multicast. A method to construct contention-free k-binomial tree has been proposed. The k-binomial algorithm has been evaluated through simulation and its performance has been shown to be better than that of 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 other collective communication operations with such packetization and network interface support.

Additional Information: A number of related papers and technical reports can be obtained from the home page of *Parallel Architecture and Communication* (PAC) research group. The URL is *http://www.cis.ohio-state.edu/~panda/pac.html*.

References

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

[2] L. De Coster, N. Dewulf, and C.-T. Ho. Efficient Multipacket Multicast Algorithms on Meshes with Wormhole and Dimension-Ordered Routing. In *ICPP*, pp III:137–141, Aug 1995.

[3] T. V. Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *ACM SOSP*, 1995.

[4] Y. Huang and P. K. McKinley. Efficient Collective Operations with ATM Network Interface Support. In *ICPP*, pp I:34–43, Aug 1996.

[5] R. Kesavan, K. Bondalapati, and D. K. Panda. Multicast on Irregular Switch-based Networks with Wormhole Routing. In *HPCA-3*, pp 48–57, Feb 1997.

[6] R. Kesavan and D. K. Panda. Minimizing Node Contention in Multiple Multicast on Wormhole k-ary n-cube Networks. In *ICPP*, pp I:188–195, Aug 1996.

[7] R. Kesavan and D. K. Panda. Optimal Multicast with Packetization and Network Interface Support. Technical Report OSU-CISRC-2/97-TR10, Jan 1997.

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

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

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

[11] 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 *ICPP*, Aug 1996.

[12] K. Verstoep, K. Langendoen, and H. Bal. Efficient Reliable Multicast on Myrinet. In *ICPP*, pp III:156–165, Aug 1996.