Where to Provide Support for Efficient Multicasting in Irregular Networks: Network Interface or Switch?*

Rajeev Sivaram† Ram Kesavan† Dhabaleswar K. Panda† Craig B. Stunkel‡

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

‡IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598
Phone: (914) 945-3090
Email: stunkel@watson.ibm.com

Contact Author: Rajeev Sivaram

Abstract

Recent research has proposed methods for enhancing the performance of multicast in networks with irregular topologies. These methods fall into two broad categories: (a) network interface (NI) based schemes that make use of enhanced functionality of the software/firmware running at the network interface (NI) processor, or (b) switch-based methods that use enhancements of the switch architecture to support hardware multicast. However, it is not clear how these methods compare to each other and when it makes sense for an architect to use one over the other.

In order to answer such questions, in this paper we compare the performance of three efficient multicasting schemes: an NI-based multicasting scheme that uses a k-binomial tree [16], a switch-based multicasting scheme that uses path-based multidestination worms [15], and a switch-based multicasting scheme that uses a single tree-based multidestination worm [35]. We first identify the parameters involved in multicast communication and derive analytical values for the multicast latency under each of the three schemes. We then perform a number of simulation experiments to study the performance of the three schemes for single multicast traffic while changing a number of system parameters one at a time to isolate their impact. Finally, we study the performance of these schemes under increasing multicast load.

Our results show that the switch-based multicasting scheme using a single tree-based multidestination worm performs the best among the three schemes. However, the NI-based multicasting scheme is capable of delivering high performance compared to the switch-based multicast using path-based worms especially when the software overhead at the network interface is less than half of the overhead at the host. We therefore conclude that support for multicast at the NI is an important first step to improving multicast performance. However, there is still considerable gain that can be achieved by supporting hardware multicast in switches. Finally, while supporting such hardware multicast, it is better to support schemes that can achieve multicast in one phase.

Keywords: Parallel computer architecture, cut-through routing, multicast, broadcast, collective communication, switch-based networks, irregular networks, performance evaluation.

*This research is supported in part by NSF Career Award MIP-9502294, NSF Grant CCR-9704512, an IBM Cooperative Fellowship, and an Ohio State University Presidential Fellowship.

Contents

1 Introduction **1**

2 System Model **3**
2.1 Network Model . . . 3
2.2 Routing Issues . . . 3

3 Multicasting Approaches **4**
3.1 Traditional Communication Support . . . 4
3.2 Lightweight Communication Software . . . 5
3.3 Multi-phase Software Approaches to Multicast . . . 6
3.4 Enhanced Multicasting Schemes . . . 7
3.4.1 Multicasting using Network Interface Support . . . 7
3.4.2 Multicasting using Switch Support . . . 8
3.4.3 Tree-Based Multicasting using Multidestination Worms . . . 9
3.4.4 Path-based Multicasting using Multidestination Worms . . . 10
3.5 Comparison of Enhanced Multicasting Schemes . . . 11

4 Analytical Evaluation **12**
4.1 Multicast using Unicast Messages . . . 13
4.2 Multicast with Smart Network Interface Support . . . 13
4.3 Multicast with Switch Support . . . 14
4.4 Analytical Comparison . . . 14

5 Simulation-based Evaluation **14**
5.1 Experiments and Performance Measures . . . 14
5.2 Generating Irregular Topologies . . . 16
5.3 Single Multicast Performance . . . 16
5.3.1 Effect of R . . . 16
5.3.2 Effect of Software Overhead at Host Processor . . . 17
5.3.3 Effect of System Size . . . 17
5.3.4 Effect of Switch Size . . . 18
5.3.5 Effect of Number of Switches . . . 18
5.3.6 Effect of Message Length . . . 19
5.3.7 Effect of Packet Size . . . 19
5.4 Latency versus Applied Load for Multicast . . . 20
5.4.1 Effect of R . . . 20
5.4.2 Effect of Number of Switches . . . 21
5.4.3 Effect of Message Length . . . 21

6 Related Work **21**

7 Conclusion **22**

1 Introduction

In a world where computing needs are increasing by the day, there is a constant search for cost-effective high-performance computing solutions. Using a network of commodity workstations as a parallel computing environment has the potential to provide such a cost-effective high-performance computing environment. Much recent research has therefore focussed on extending parallel processing solutions to such networks of commodity workstations (NOWs).

Traditionally, parallel processing machines have been built with processing nodes interconnected in regular topologies such as a mesh [12], torus [5], hypercube [11], multistage interconnection network (MIN) [22, 40], etc. Such regular topologies have important mathematical properties that make message communication easier/better by making message routing simpler, lowering the average distance per communication, and/or increasing the bisection bandwidth [7]. However, networks of workstations environments have typically evolved from computing environments where communication requirements are much lower (both in terms of latency and bandwidth). In such an environment, the workstations that form the processing nodes are typically interconnected in arbitrary, irregular topologies. Using such topologies allows easy addition and deletion of nodes to the computing environment making the overall environment more amenable to network reconfigurations and resistant to faults. However, these topologies do not possess many of the attractive mathematical properties of the regular topologies. A lot of the problems relating to inter-processor/inter-node communication that were solved for regular topologies (such as the deadlock-free cut-through routing of packets [1, 31, 33]) are therefore being revisited for such irregular topologies.

Collective communication is an important type of communication operation in parallel systems, and involves communication among groups of (3 or more) processes [20, 26]. Examples of collective operations include multicast [24], barrier synchronization [36], reduction, etc. The importance of such operations is underlined by the inclusion of several primitives for collective communication in the newly drafted Message Passing Interface (MPI) standard [23]. Such collective operations are also used for system level operations in Distributed Shared Memory systems, such as for cache invalidations, acknowledgment collection, and synchronization [6].

Of these collective operations, multicast is most fundamental and important and is used for implementing several of the other collective operations. Traditionally, multicast has been implemented using the underlying support for point-to-point (unicast) communication. In the naivest multicast implementation, the multicast source individually sends a unicast message to every one of its n destinations. However, such a scheme leads to poor performance because it consists of n sequential communication steps, where every communication step involves the high software overhead for sending and receiving a message. To improve multicast performance, many schemes have been proposed that use a multi-phase approach starting with the source sending out a multicast message to one of its destinations [21, 44]. In succeeding phases, the destinations that have received the message act as secondary sources and transmit the message to other destinations. Such a scheme reduces the impact of the high overhead associated with sending/receiving point-to-point messages by allowing multiple nodes to simultaneously transmit/receive messages in a given phase. The number of communication steps is reduced from n to $log_2(n+1)$ by such a multicasting scheme.

A number of schemes have been recently proposed to further improve multicast performance. These schemes can be classified into two main categories. The first category of schemes is referred to as *switch-based multicasting schemes*. These schemes use enhancements at the routers/switches of the network to allow an incoming packet to be forwarded to multiple output ports of the router/switch [34, 39]. Such a method allows

messages (christened *multidestination messages*) to be communicated to multiple destinations incurring a single software overhead for sending the message. There are two subclasses of switch-based multicasting schemes: tree-based [35] and path-based [15]. In the best tree-based multicasting scheme, multicast can be performed in a single phase using one multidestination worm from the source node to all the destinations [35, 39]. The path-based multicasting schemes typically require multiple multidestination worms to perform multicast to arbitrary destination sets. These worms are transmitted in multiple phases with the destinations in a phase acting as secondary sources in succeeding phases of the multicast [15].

The second category of multicasting schemes is referred to as *network interface-based* or *NI-based multicasting schemes*. Typically, communication between two nodes incurs software overhead at both the host and at the network interface for both sending and receiving a message. The NI-based schemes use enhancements at the network interface so that multicast messages are forwarded to the next destination as soon as they are received at the network interface of an intermediate destination (and the message has begun to be transferred to the memory of the local host) [41]. This hides the significant software overhead for receiving a message at an intermediate destination host, and eliminates the overhead at the host for sending the message to other destinations in the succeeding phases of the multicast.

It is obvious that a multicasting scheme with enhanced support at the network interface *and* the switches will perform better than a scheme that makes use of support at *either* the network interface or at the switches. However, it is necessary to first compare the effect of improved support for multicast at the switches to support for multicast at the network interface to decide which of these schemes results in better performance and when. While making such a comparison, we must therefore assume no network interface support for the switch-based multicasting schemes, i.e., every communication phase under the tree-based and path-based schemes should incur software overhead at the host and network interface of the source and (intermediate) destinations. Such a comparison is needed for an architect to evaluate the relative merits of these two categories of multicasting schemes and to decide which scheme to support and when. In order to make such a decision, a number of questions need to be answered, such as the following. How quickly can multicast be performed with support at the network interfaces and low overhead messaging layer support? How does multicast with switch/router support (alone) compare with multicast implemented with network interface support? For what range of system parameters does it make sense to use one over the other?

The goal of this paper is to provide answers to these questions by comparing the performance of the single phase tree-based multicasting scheme to the best path-based multicasting scheme and to an optimal NI-based multicasting scheme. After describing these three schemes, we parameterize the various components that contribute to multicast latency. We then derive analytical values for multicast latency under each of the schemes in terms of these parameters. Next, we perform extensive simulations to evaluate the impact of the various system parameters on the performance of the three schemes by considering the performance of single multicasts and by varying each of the parameters one at a time. These parameters include, system size, switch size, message length, packet size, number of switches, software overhead at the hosts, and the ratio of the overhead at the host to the overhead at the network interface. Finally, we study the latency of these schemes under increasing multicast load with a variation of a few selected parameters.

Our analysis and simulations show that the single phase tree-based multicasting scheme studied in this paper is the most powerful multicasting scheme. However, the NI-based scheme can deliver extremely high performance, especially when the software overhead for absorbing and retransmitting messages at the interface is considerably lower than the corresponding overheads at the host. The path-based multicasting scheme's performance varies with variation in a number of network parameters and it can perform worse

than the NI-based scheme in a number of cases. We therefore conclude that support for multicast at the NI is an important first step to improving multicast performance. However, there is still considerable gain that can be achieved by supporting hardware multicast in switches. Finally, while supporting such hardware multicast, it is better to support schemes that can achieve multicast in one phase.

The remaining part of the paper is organized as follows. In Sec. 2 we present our network model, and the underlying routing algorithm assumed in this paper. We then present an overview of efficient communication in a NOW environment and of the multicasting schemes used for our comparison in this paper in Sec. 3. An analytical comparison of these schemes is then presented in Sec. 4 and an exhaustive simulation based comparison of the three schemes is presented in Sec. 5. We then present an overview of some of the related work in Sec. 6. Finally, we present our conclusions in Sec. 7.

2 System Model

In this section, we present the network model assumed in this paper. The related deadlock-free routing issues for such a network are also discussed.

2.1 Network Model

Figure 1(a) shows a typical parallel system using a switch-based interconnect with irregular topology. Such a network consists of a set of switches where each switch has a set of ports. The system in the figure consists of eight switches with eight ports per switch. Some of the ports in each switch are connected to processors, some ports are connected to ports of other switches to provide connectivity between the processors, and some ports are left open for further connections. Such connectivity is typically irregular and the only thing that is guaranteed is that the network is connected. Thus, the interconnection topology of the network can be denoted by a graph G = (V,E) where V is the set of switches, and E is the set of bidirectional links between the switches [3, 32]. Figure 1(b) shows the interconnection graph for the irregular network in Fig. 1(a). It is to be noted that all links are bidirectional and multiple links between two switches are possible.

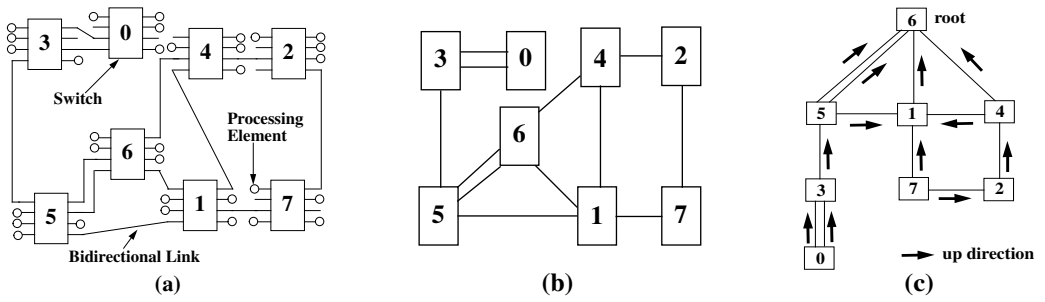

Figure 1: a) An example system with switch-based interconnect and irregular topology; b) corresponding interconnection graph G; c) corresponding BFS spanning tree rooted at node 6.

2.2 Routing Issues

Several deadlock-free routing schemes have been proposed in the literature for irregular networks [3, 10, 31, 32]. Without loss of generality, in this paper we assume the routing scheme for our irregular network to be similar to that used in Autonet [32] due to its simplicity and its commercial implementation. Such routing allows adaptivity, and is deadlock-free.

In this routing scheme, a breadth-first spanning tree (BFS) on the graph G is first computed using a distributed algorithm. The algorithm has the property that all nodes will eventually agree on a unique spanning tree. Deadlock-free routing is based on a loop-free assignment of direction to the operational links. In particular, the "up" end of each link is defined as: 1) the end whose switch is closer to the root in the spanning tree; or 2) the end whose switch has the lower ID, if both ends are at switches at the same tree level. The result of this assignment is that the directed links do not form loops. Figure 1(c) shows the BFS spanning tree corresponding to the interconnection graph shown in Fig. 1(b), and the assignment of the "up" direction to the links. To eliminate deadlocks while still allowing all links to be used, this routing uses the following up/down rule: a legal route must traverse zero or more links in the "up" direction followed by zero or more links in the "down" direction. Putting it in the negative, a packet may never traverse a link along the "up" direction after having traversed one in the "down" direction. Details of this routing scheme can be found in [32].

In order to implement the above routing, each switch has an indexed forwarding table. The forwarding tables can be constructed to support both minimal and non-minimal adaptive routing. In this paper, we only consider minimal adaptive routing. Thus, the forwarding tables allow only legal routes with the minimum hop count. When multiple minimal routes exist from the source to the destination, the forwarding table entry shows alternative forwarding ports. The choice of the outgoing port is decided dynamically based on the ports which are free when the header flits arrive at the switch. In the case of multiple outgoing ports being free, the routing scheme randomly selects one of them.

3 Multicasting Approaches

In this section, we present an overview of three multicasting schemes that we compare in this paper. We begin by describing the traditional support for point-to-point communication in NOW environments. We then describe the current-day state-of-the-art lightweight communication layers used to enhance the performance of unicast communication. Next, we describe the traditional approach to multicasting using multiple phases of unicast messages. Finally, we present two categories of techniques for providing enhanced support for efficient multicast and present the three schemes studied in this paper.

3.1 Traditional Communication Support

Both hardware and software are used for providing efficient communication support in NOWs. The hardware includes cut-through switches, network interface cards (NICs) at the host workstations, and the physical wiring that connects the switches to each other and to the adaptors at the hosts. The software enables the sending of messages through this hardware, and implements the desired communication protocols.

Figure 2(a) shows the architecture of a generic cut-through switch with k ports. Each port consists of one input and one output link. A port can be connected to the port of another switch, a host workstation, or kept open. Each port consists of an input and an output buffer. Although these buffers only need to be big enough to capture the header flit of an incoming worm so that the routing decision can be made as soon as the header flit arrives, deeper buffers are usually required to perform flow control efficiently across long links and to reduce link contention. A k-port switch typically provides a $k \times k$ crossbar connectivity in order to enable concurrent transfer of messages from the input buffers to any of the output buffers [3, 32, 37, 38, 40].

Figure 2(b) shows the architecture of a generic NIC which typically plugs into the I/O bus of the host. A dedicated communication processor is embedded in the NIC. This allows the sophisticated processing of the

network transactions to be off-loaded to the NI processor, thus freeing up the host processor for computation. The NIC also contains some amount of memory, up to four DMA engines, and the link interface. All messages sent in/out of the workstation are staged through send/receive queues implemented in this memory. The memory is also used to store the instructions and data associated with the NI processor. Two DMA engines are used to transfer the messages between the host memory and the send/receive queues on the NI memory, and the other two DMA engines are used to transfer data between the link interface and the send/receive queues on the NI memory.

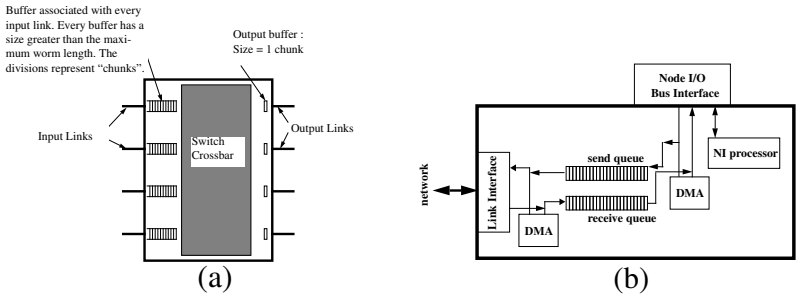

Figure 2: Generic diagrams of (a) a typical cut-through switch and (b) a typical network interface at a workstation.

The software portion of the communication support has multiple responsibilities. First, it enables transmission of messages from the sender host node to the receiver host node by adding appropriate header information to data. Modern day networks typically limit the size of the largest packet. This is done to minimize network contention and to support efficient buffer utilization in the network and the network interfaces. The communication software on the sender side is responsible for breaking up a long message into multiple packets and sending them out into the network. Similarly, the software on the receiver side is responsible for collecting all the arrived packets and assembling them into the complete message. Second, the software provides protection between various processes sharing the same physical NIC. Finally, the software is often responsible for ensuring that messages are reliably delivered.

Conventionally, communication protocols like TCP/IP are implemented by the communication software. Using such traditional communication protocols involves operating system calls to send and receive messages. These calls introduce hundreds of microseconds or more of overhead in a message transmission which is orders of magnitude higher than the latency of the message through the actual network. This has led to a considerable amount of research devoted to the design and development of lightweight communication protocols.

3.2 Lightweight Communication Software

We now examine how the software overhead is reduced by using such lightweight messaging software. Lightweight high performance messaging layers often bypass the operating system, mapping the network interface device directly into the user address space and accessing it directly via load/store operations. Protection is achieved by virtualizing the physical NI device to provide a virtual NI device for each process trying to access the physical NI device. Each process has a communication endpoint consisting of a send queue, a receive queue, and a certain amount of state information. Each process can deliver a message to any of the receive queues by depositing the message in its own send queue with an appropriate destination identifier. Such messaging systems also try to minimize buffer copying which contributes to a major part of

the overhead. Examples of such messaging systems are Active Messages [19, 43], U-Net [42], Fast Messages (FM) [25], and SHRIMP [2, 8].

Let us examine how these lightweight messaging systems achieve message transfer. 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. A typical message transfer in these systems is done in the following way. At the sender side, one of two schemes is used: (a) programmed I/O, or (b) DMA. The host processor at the sender fragments the message into fixed size packets and transfers them to the send queue of the NI using programmed I/O [25]. Alternatively, the host processor copies data into the host DMA memory, writes the message pointers to the network interface, and the processor uses DMA to copy the packets to the send queue [42]. The DMA setup overhead makes the programmed I/O method more efficient for small message sizes, but DMA proves more efficient for large message sizes. Subsequently, the software executing at the NI processor 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 processor detects the received packets and uses DMA to copy them to the host memory. Without loss of generality, the discussions in this paper assume DMA to transfer data between host memory and network interface device on the sender and receiver side.

Collective communication operations like multicast can be built on such messaging systems. Let us examine how the multicast operation is implemented on such a messaging system.

3.3 Multi-phase Software Approaches to Multicast

Efficient multicast algorithms are typically hierarchical in nature. This means that some destinations serve as intermediate sources, i.e., when they receive a message, they forward copies of it to other destinations. Many such hierarchical algorithms have been proposed in the literature [13, 20, 21, 30] to implement multicast. Figure 3 shows an example of a multicast from a source node to seven other destinations. In the figure, the numbers in brackets indicate the step numbers.

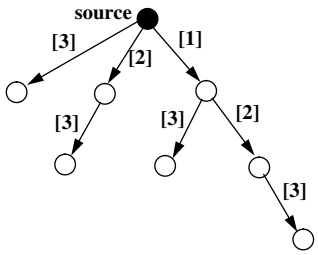

Figure 3: Example of a hierarchical multicast algorithm on a destination set size of 7.

It can be easily be observed that $\lceil log_2 n \rceil$ communication steps are required for such a binomial tree based hierarchical multicast to be completed [21]. A communication step is the time required for a message to sent from one host node to another. Even with lightweight messaging layers, the latency of such a multicast operation is still dominated by the communication software overhead. Recent research to reduce the effect of this overhead has been in two main directions.

3.4 Enhanced Multicasting Schemes

The two major directions in which enhancements have been proposed for improving multicast latency are: (a) increasing the functionality of the software/firmware running at the NI processor, and (b) hardware multicast at the switch. We describe these concepts in greater detail in the following subsections.

3.4.1 Multicasting using Network Interface Support

The software/firmware at the NI processor controls the sending/receiving of packets into/from the network. A packet received from the network is copied from the receive queue of the NIC to host memory using DMA. A packet to be sent is copied from the host memory to the send queue in the NIC using DMA. It is then transferred from the send queue to the link interface (using DMA) to be sent out into the network.

Let us consider the multicast of a message that spans multiple packets. Figure 4(a) shows the forwarding of a 2-packet multicast message at an intermediate node of a multicast tree. Each of the packets of the message is 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 NI for each packet transmission.

Smart NI support can reduce this overhead for multicast, especially for multi-packet messages. Current lean messaging layers allow modification of part of the software running on the NI processor. This part of the software can be modified to allow it to identify a multicast packet [16, 41]. If the next outgoing packet in the send queue of the source node is a multicast packet, the NI processor forwards replicas of the packet to the nodes adjacent to the root of the multicast tree. When a multicast packet is received at the NI of an intermediate node, the NI processor starts copying (using DMA) the packet to host memory. Simultaneously, it forwards replicas of the packet to its children in the multicast tree. Thus, the overhead at the intermediate node's host to receive a packet is hidden, and the overhead at the host to send the packet to its children in the multicast tree is eliminated. Figure 4(b) shows such forwarding with smart network interface support. An example of the use of such a smart network interface has been described in [41].

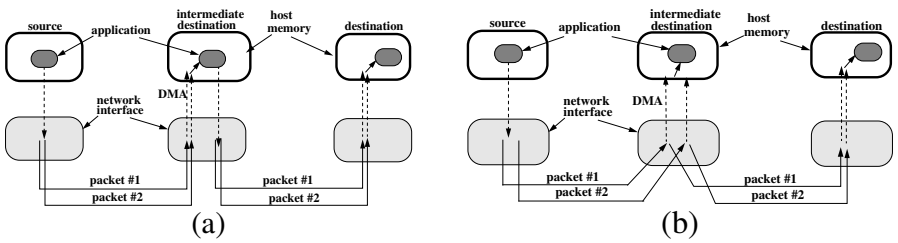

Figure 4: Forwarding of a 2-packet long multicast message by intermediate node using (a) conventional NI support, and (b) smart NI support.

Let us estimate the latency of a multicast operation using smart network interface support. We assume that the software overhead for message initiation at the host processor is t_{hs}, and the overhead for receiving a message at the host processor is t_{hr}. The software start-up overhead, t_{hs}, is incurred once at the host processor of the source of the multicast to transfer the data to the NI 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_{hr}, 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 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 NI of one node to the NI of another node as a *step*. The time for this step, denoted t_{step}, includes the overhead at the sender NI for sending a packet, propagation overhead, and the overhead at the receiver NI for receiving the packet.

There are two possible implementations of the smart network interface support described above. These implementations are discussed in [16], and the FPFS (First-Packet-First-Served) implementation is shown to be superior in terms of ease of implementation and buffer requirements at the NIC. The FPFS implementation is described briefly below.

In this implementation, the NI forwards the message on a per-packet basis. The NI 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 NI 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 NI, it forwards the packet to each of the children, and so on till the last packet is forwarded.

The k-binomial tree has been shown to be optimal for multi-packet multicast on systems with smart NI support [16]. A k-binomial tree is defined as a recursively doubling tree where each vertex has at most k children. Figures 5(a) and 5(b) show examples of 3-binomial and 4-binomial trees with multicast set size of 16. The value of k is a function of the size of the multicast set (n_D) and the number of packets in the multicast message (n_p). Since this function is not a closed-form function, the value of k can be precomputed and stored in a table for all possible values of n_D and n_p. It has been argued that the value of k is identical for a range of n_p values and the value of k converges to 1 with increase in n_p. Thus, this table requires less than $O(n_D n_p)$ memory. A method for constructing k-binomial trees with minimized contention on irregular switch-based networks has also been proposed in the literature [16].

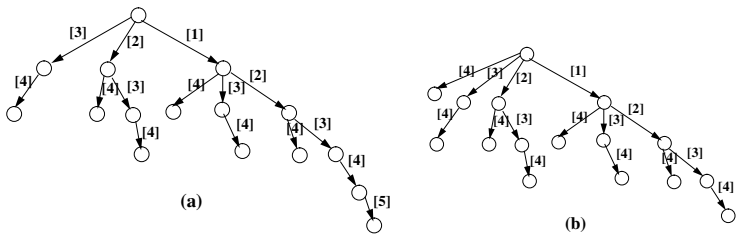

Figure 5: Examples of k-binomial trees on a multicast set size of 16: (a) the 3-binomial tree, and (b) the 4-binomial tree.

3.4.2 Multicasting using Switch Support

Another method for improving multicast performance is to provide switches with support for replicating incoming messages to multiple output ports. The basic idea behind such a scheme is to communicate a single message (called a *multidestination message*) from a source to multiple destinations in almost the same time it takes to send a unicast message to another node [18, 29]. The number of destinations covered depends

on the type of encoding/decoding used for the message header.

Replication of messages in switches that support cut-through routing is deadlock-prone. A deadlock-free replication method that uses distributed arbitration but guarantees that there is space for the multicast packet to eventually be completely buffered at the switch has been proposed in [34, 39]. This ensures that the available resources can be used and freed up while continuing to wait for other resources (output ports in our case) to be freed. In an input buffered switch, this guarantee reduces to the requirement that the buffers in the switch be larger than the largest multicast packet. We will assume such input buffered switches in the rest of this paper. For simplicity, we will assume that the buffers use a single FIFO queue to store packets (as in the switches of [9]).

Given such support for replication at the switches, there are two schemes that have been proposed for carrying out multicast in switch based irregular networks: tree-based [35] and path-based [15]. The two schemes differ in the restrictions placed on worm replication, the number of worms required to perform multicast to an arbitrary destination set, the complexity of multicast header formation, and the complexity of the header decoding logic required at the switches. However, we assume the multidestination worms under either scheme conform to the base up*/down* routing algorithm, i.e., the path followed by a multicast packet does not violate any of the rules for routing unicast packets in the system [28]. These schemes are discussed in greater detail in the following two subsections.

3.4.3 Tree-Based Multicasting using Multidestination Worms

Tree-based multicasting places no restriction on the replication of a worm at a given switch. The basic idea of tree based replication is to have multiple copies of a packet propagate down a tree which has switches as internal vertices and destinations nodes as leaf vertices.

Two methods have been proposed for tree-based multicast in irregular networks [35]. In this paper, we use the method that uses a single multidestination worm with a bit-string encoded header to perform multicast [35, 39] as a representative of the tree-based multicasting method. The basic idea is to encode the multidestination worm header using an N-bit string where N is the number of nodes in the system. The destinations of the multicast are identified by setting the ith bit of the the bit string to '1' if node i is a destination of the multicast (all other bits are set to '0'). Every switch has a similar N-bit string (called the *reachability string*) associated with every one of its output ports that lead to links in the "down" direction. The reachability string specifies the set of destination nodes that are reachable through the given ("down") output port (subject to the restrictions of the base routing algorithm). After traveling adaptively to a *least common ancestor* switch using links in the "up" direction, a multicast packet is forwarded to those "down"-ward output ports whose reachability string has one or more '1' bits in the same position as in the packet header. To simplify decoding, the reachability strings are set up so that only one of them has a '1' bit corresponding to a given node. This restricts the amount of adaptivity, because although more than one output port may lead to a given destination, a multicast packet is forwarded to a fixed output port to cover the destination. For irregular networks, the bit string header must furthermore be modified so that the copy of the packet forwarded through a given port only has the set of destinations being covered through that port in its bit string header. Details of the logic to do this and the method for setting up the reachability strings are presented in [35].

Figure 6 illustrates the tree-based multidestination scheme using bit-string encoding for a sample multicast. Figure 6(a) shows the BFS graph of a sample irregular network. Let us assume that there is at least one destination node connected to each of the switches that are highlighted, and that the source node is

connected to switch 10. Figure 6(b) shows the tree-based multidestination worm that covers all the destinations of the multicast. Finally, Fig. 6(c) shows a bit-string encoded header and the reachability strings at a 4-port switch in an 8-node system. The worm with the header shown in Fig. 6(c) has to be replicated to all the output ports of the switch.

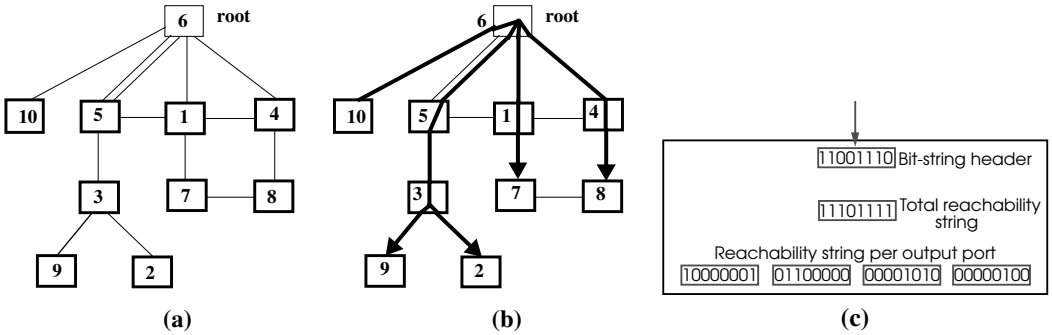

Figure 6: Illustrating the single-phase tree-based multicasting scheme: a) sample BFS graph of an irregular network with participating switches in the multicast highlighted, b) the tree-based multidestination worm, and c) an example of a bit-string encoded header and the various reachability strings in a 4-port switch.

3.4.4 Path-based Multicasting using Multidestination Worms

Under the path-based multicasting scheme [15], multidestination worms use almost exactly the same path followed by a unicast worm from a source to one of its destinations (as allowed under the base routing algorithm). However, at every switch along its path, the multidestination worm is allowed to replicate and cover all the destinations of the multicast that are connected to the switch. At every switch, the worm is allowed to replicate to multiple output ports that are connected to destination nodes and (if needed) to exactly one other output port that is connected to a switch. Such multidestination worms are called *multi-drop path-based* worms. Figure 7(a) shows an example of a multi-drop path-based multidestination worm. Since there may be no single path from the source which covers all of the destinations, path-based multicasting requires multiple multidestination worms to cover an arbitrary multicast destination set. Furthermore, to arrive at a relatively small number of multidestination worms that cover a given destination set, an algorithm is needed that chooses appropriate paths so that as many destinations as possible are covered by each of the worms. Multiple algorithms have been proposed for this [15]. In addition, a multi-phase algorithm is required to cover the destination set using the multidestination worms so that contention is reduced and multicast can be performed efficiently. In this paper, we use the Multi-Drop Path-based Less Greedy (MDP-LG) algorithm that finds a small number of multidestination worms to cover the destination set and decides how to send these worms in multiple-phases so as to reduce contention and enhance performance. Details of this algorithm can be found in [15], where this algorithm was shown to perform best among the proposed algorithms.

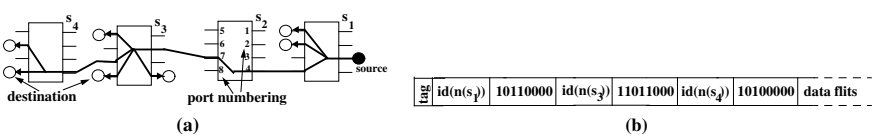

Figure 7: The Multi-drop mechanism: (a) example of a multidestination worm and (b) the header encoding.

The encoding used for the header of each of the multidestination worms under the path-based scheme is as follows. The header of a path-based multidestination worm consists of $2r$ fields, where r is the number of switches on the worm's path where it replicates. The header consists of alternating fields consisting of a node ID field followed by a p-bit string field (where p is the number of input ports/output ports in each switch). Figure 7(b) shows the header encoding of the worm shown in Fig. 7(a). The expression $id(n(s_i))$ is the ID of any arbitrary destination node which is connected to the switch s_i. The node ID field is used to look up the tables for unicast message routing and to route the message towards the switch that connects the specified node. When that switch is reached, the node ID field is stripped and the p-bit string field is examined. The p-bit string has '1' bits corresponding to the output ports to which the worm has to be forwarded. As mentioned above, at most one of these ports leads to another switch. The other ports correspond to destination nodes where the message is absorbed. The p-bit string field is stripped before the worm is forwarded to the next switch (if any) where once again the node ID field at the head of message is used to look up the unicast routing tables and to forward the message to an appropriate switch. When the message reaches the last switch on its path, the node ID field is stripped away and the message is forwarded to the output ports that correspond to the '1' bits in the last (p-bit string) field of the header.

3.5 Comparison of Enhanced Multicasting Schemes

We now qualitatively compare the three enhanced multicasting schemes described above. The NI-based multicasting scheme requires the computation of the entire k-binomial multicast tree. It may also require additional memory at the network interfaces to buffer multicast packets. This is because each packet is forwarded to multiple destinations, and has to be buffered till the NI-processor has injected all required copies into the network. However, the NI-based multicasting scheme requires no additional support at the switches than what is required for traditional point-to-point message communication.

The tree-based multicasting scheme only requires a single multidestination worm to perform multicast. Furthermore, encoding the multidestination worm header is fairly simple: start with an N bit string and set the bits in the positions corresponding to the destinations to '1'. However, decoding the multidestination header is more complex. First, the switches need to be equipped with reachability strings, which means space is required at the switches for this. Second, there is a one time cost of setting up the values in the strings (which can be done at the time of network startup or reconfiguration). Finally, the N bit string has to be compared with the reachability strings of each of the N output ports and the copy of the worm forwarded through a given output port should carry a modified header. Depending on the size of the bit string (which in turn depends on the system size), the cost of such logic may be significant. Finally, support for deadlock-free replication is required at the switches.

The path-based multicasting scheme also requires support for replication at the switches. Furthermore, encoding the worm header is relatively harder. The source needs to know the network topology and needs to run an algorithm to decide on paths that can cover the destination set using very few multidestination worms. Once the paths have been formed, the header must be formed in the format described above. However, decoding the multidestination header may be relatively easy. First, there is no necessity for maintaining reachability strings at each of the switches. Second, the decoding operation at the switches is relatively simple: it is the same table lookup operation required for unicast messages or it is the simple processing of a k-bit string. Finally, the cost of decoding logic does not increase with increase in system size.

We have provided the various trade-offs relating to the cost and complexity of each of three multicasting

schemes above. In the next two sections we compare the performance of the three schemes analytically and using simulation experiments.

4 Analytical Evaluation

Having described the three different efficient multicasting schemes that we are comparing in this paper, we now analytically compare their performance. Let us assume the parameters shown in Table 1.

Table 1: VARIOUS PARAMETERS THAT AFFECT MULTICAST LATENCY UNDER EACH OF THE THREE SCHEMES.

t_{hs}	communication software overhead per message at the host processor on the sender side
t_{hr}	communication software overhead per message at the host processor on the receiver side
t_{ns}	overhead per packet at the NI processor on the sender side
t_{nr}	overhead per packet at the NI processor on the receiver side
t_g	time between injecting the last flit of a packet and the first flit of the next packet of the same message at the NI link interface on the sender side
t_p	time for propagation of a flit over a physical link
t_{sw}	time for propagation of a flit through a switch (from input buffer to output buffer)
t_{ru}	time to route a unicast packet through a switch
t_{rp}	time to route a path-based multidestination packet through a switch
t_{rt}	time to route a tree-based multidestination packet through a switch
l_p	number of flits in a packet
n_p	number of packets in a message
d	average number of hops for a packet from source to destination
n_{ph}	number of phases in multicast
$n_D - 1$	number of destinations in multicast

Using these parameters, let us first estimate the best case latency (with no contention) for a single packet message to be delivered from one host node to another. An overhead of t_{hs} at the host processor and an overhead of t_{ns} at the NI processor is incurred by the message on the source node. The time taken for the header flit of the packet to be routed across d hops to the destination NI is $d(t_{ru}+t_{sw}+t_p)+t_p$, and the time taken for the remaining (l_p-1) flits to reach the destination NI is $(l_p-1)max(t_{sw},t_p)$. Subsequently, an overhead of t_{nr} at the NI processor and an overhead of t_{hr} at the host processor is incurred by the message on the destination node. Thus, the latency τ_1 for a single packet message to be delivered from one host node to another is given by:

$$\tau_1 = t_{hs} + t_{ns} + d(t_{ru} + t_{sw} + t_p) + t_p + (l_p - 1)max(t_{sw}, t_p) + t_{nr} + t_{hr} \quad (1)$$

Now, let us estimate the latency for a n_p-packet message to be delivered from one host node to another. Let the time at which the last flit of the ith packet of a message leaves the NI at the source node be T. Then, the time T_1 at which the ith packet completely reaches the host processor at the destination is given by, $T_1 = T + d(t_{sw}+t_p)+t_p+t_{nr}$, which is the time for the last flit to traverse d hops added to the overhead at the destination NI processor. Here we make a realistic assumption that $d < l_p$, which means that the header

flit of the ith packet has reached the destination NI before the time T. The time at which the header flit of the $(i+1)$th packet of the same message reaches the destination NI is $T_2 = T + t_g + d(t_{ru} + t_{sw} + t_p) + t_p$, and therefore the time at which the the $(i+1)$th packet completely reaches the host processor at the destination node is $T_3 = T_2 + (l_p - 1)max(t_{sw}, t_p) + t_{nr}$. Therefore, the time interval between the ith packet reaching the host processor and the $(i+1)$th packet reaching the host processor on the destination node is given by $T_3 - T_1$. Therefore, the latency for a n_p-packet message to be delivered from one host node to another is given by:

$$\tau_{n_p} = \tau_1 + (n_p - 1)(T_3 - T_1) \tag{2}$$

Let us define the following values: $A = t_{hs} + t_{hr} + (n_p - 1)t_g$, and $B = t_{ns} + t_{nr} + d(t_{sw} + t_p) + t_p + (l_p - 1)max(t_{sw}, t_p)$. Therefore, the above equation can be rewritten as:

$$\tau_{n_p} = A + B + dt_{ru} + (n_p - 1)(dt_{ru} + (l_p - 1)max(t_{sw}, t_p)) \tag{3}$$

Using the above equations, let us estimate the multicast latencies for the following schemes assuming the size of the multicast set (i.e., the set containing the source and the multicast destinations) to be n_D. We will assume the ideal case of a single multicast in the system and zero contention between messages of the same multicast.

4.1 Multicast using Unicast Messages

First, let us estimate the multicast latency using the a binomial tree based hierarchical algorithm using unicast-based messaging. It has been well documented that given a multicast set of size n_D such an algorithm takes $\lceil log_2 n_D \rceil$ steps to complete [21]. Each step involves the delivery of a message from one node to another. Therefore, the multicast latency for an n_p-packet message using such a hierarchical unicast-based multicasting algorithm is given by:

$$\tau_{um} = \lceil log_2 n_D \rceil \times \tau_{n_p}$$

$$\tau_{um} = \lceil log_2 n_D \rceil (A + B + dt_{ru} + (n_p - 1)(dt_{ru} + (l_p - 1)max(t_{sw}, t_p))) \tag{4}$$

4.2 Multicast with Smart Network Interface Support

Let us now estimate the multicast latency when using the NI-based scheme described in Sec. 3.4.1. It has been shown in [16] that given a multicast set of size n_D and an n_p-packet message, the time taken by NI-based scheme is $\alpha + (n_p - 1)k\beta$, where α is the time for the first packet to reach all the destinations, β is the time for a node to inject a packet into the network, and k represents the maximum degree of the vertices in the k-binomial tree. It can be shown that the value of α is $t_{hs} + t_{hr} + f(n_p, n_D)(B + dt_{ru})$, where the function $f(n_p, n_D)$ returns the number of steps for the first packet to reach all the destinations. It can also be shown that the value of β is $t_g + l_p t_p$. (A detailed description of this derivation can be found in [16].) Thus, the multicast latency for an n_p-packet message using the NI-based scheme is given by:

$$\tau_{ni} = A' + f(n_p, n_D)(B + dt_{ru}) + (n_p - 1)kl_p t_p \tag{5}$$

where $A' = t_{hs} + t_{hr} + (n_p - 1)kt_g$

4.3 Multicast with Switch Support

Finally, let us estimate the multicast latency when using tree-based and path-based schemes with switch-level hardware multicast support. It can be easily shown that the multicast latency for an n_p-packet message using the tree-based scheme described in Sec. 3.4.3 is given by:

$$\tau_{tree} = A + B + dt_{rt} + (n_p - 1)(dt_{rt} + (l_p - 1)max(t_{sw}, t_p)) \quad (6)$$

The path-based scheme described in Sec. 3.4.4 with hardware switch-level support typically takes a few phases to complete. Let n_{ph} denote the number of phases. Then, it can be shown that the multicast latency for an n_p-packet message using the path-based scheme is given by:

$$\tau_{path} = n_{ph}(A + B + dt_{rp} + (n_p - 1)(dt_{rp} + (l_p - 1)max(t_{sw}, t_p)) \quad (7)$$

It must be noted that the value of n_{ph} is always smaller than the $\lceil log_2 n_D \rceil$ phases taken by the binomial tree based hierarchical algorithm described in Sec. 3.3, i.e. $n_{ph} \leq \lceil log_2 n_D \rceil$.

4.4 Analytical Comparison

It can be observed from the above equations that multicast with either switch support or smart NI support will perform much better than the traditional unicast-based Umesh algorithm. This is primarily due to the term $\lceil log_2 n_D \rceil (t_{hs} + t_{hr})$, which is a dominant part of the Umesh latency, being larger than the corresponding terms in the multicast latencies with switch support, which are $n_{ph}(t_{hs} + t_{hr})$ and $t_{hs} + t_{hr}$ for the path-based and tree-based schemes, respectively.

The performance of the tree-based scheme, which completes in a single phase, is clearly better than that of the path-based scheme and the NI-based scheme. However, the performance of the path-based scheme is comparable to that of the NI-based scheme and is effected by the relative values of n_{ph} and $f(n_p, n_D)$. The NI-based scheme performs better than the path-based scheme if

$$n_{ph} > \frac{A' + f(n_p, n_D)(B + dt_{ru}) + (n_p - 1)kl_p t_p}{A + B + dt_{rp} + (n_p - 1)(dt_{rp} + (l_p - 1)max(t_{sw}, t_p))} \quad (8)$$

5 Simulation-based Evaluation

The relative performance of the three multicasting schemes described in the previous sections may depend on a number of parameters. To study the factors that may affect multicast performance in the presence of network contention, we performed simulation experiments varying a number of system parameters one at a time. In the following subsection we present the experiments we performed and the parameters that we varied. We then describe our method for generating different irregular topologies. The results of our experiments with single multicast are presented next, followed by the results for our experiments to measure the impact of increasing applied load on multicast latency.

5.1 Experiments and Performance Measures

We used a C++/CSIM based simulation testbed [27] for our experiments. The simulation testbed is capable of modeling a large number of topologies, and can model a variety of flow control techniques ranging from wormhole routing to virtual cut-through routing. In this paper, we assume cut-through routing as the flow control technique and choose an input buffer size of 640 flits. While this value is greater than the largest

packet that we assume (viz. 128 flits), the flow control technique places weaker requirements than virtual cut-through: a packet is allowed entry into the input buffers even if there isn't space for the entire packet to be buffered in the switch. As specified before, for an input buffered scheme a buffer larger than the largest packet ensures that a multicast can eventually be completely buffered and this is sufficient to prevent deadlock.

For each of our experiments, we assume the following default parameters. We assume that the I/O Bus at every host has a bandwidth of 264 MB/s. The current PCI Bus standard calls for a bandwidth of 132 MB/s: our assumptions reflect the belief that I/O bus bandwidths will increase in the future. We use the term R to denote the ratio of t_{hs} to t_{ns}, and we assume a default value of $R = 1$, i.e., in our default case, $t_{hs} = t_{ns}$. This reflects our belief that messaging layers will become thin and efficient, and that while most work relating to initiating a message may still be done on the host processor, the relatively low speed of the NI processor makes the value of t_{ns} comparable to t_{hs}. We assume a default cycle time of 5 ns, and a default value of 1000 cycles for t_{hs} (this time includes the time for initiating the DMA transfers and any other operations that the messaging layer at the host may perform). This value corresponds to the software overhead incurred at the host using many of the current-day lightweight messaging layers (such as FM [25]). For all our experiments, we assume that the (portions of the) overhead for receiving a message at the host (t_{hr}) and at the NI (t_{nr}), are equal to their counterparts for the sending overhead (i.e. $t_{hr} = t_{hs}$ and $t_{nr} = t_{ns}$).

In the network, we assume that links are 1 byte wide and that this is equal to the flit size. The link propagation time for a flit across a physical link (t_p) is assumed to be 1 cycle, as is the time to propagate through a switch crossbar from the input to the output buffer of the switch (t_{sw}). We assume a uniform routing overhead of 1 cycle for all three schemes (t_{ru} for the nic-based scheme, t_{rp} for the path-based scheme, and t_{rt} for the tree-based scheme). This reflects our assumption that while the *cost* of the logic involved for decoding and routing a header under the different schemes may vary, the approximate *latency* for doing these operations is likely to be kept within a cycle in most switch implementations. We assume a default packet size of 128 flits, and a default message size of 1 packet. Finally, we assume a default system of 32 nodes that are interconnected by eight 8-port switches in an irregular topology. Our method for generating different irregular topologies is described in the next subsection.

We use two types of experiments to measure the performance of the three multicasting schemes. In the first type of experiments, we measure the latency of single multicasts for each of the three schemes and study the effect of different parameters on the relative latencies of the three schemes. We assume that exactly one multicast occurs in the system at any given time and that there is no other network traffic. This gives us an estimate of the best possible performance of each of the three schemes in isolation. Furthermore, we can isolate the effect of the various network parameters on the performance of each of the three schemes. For our study, we vary each of the following parameters one at a time: the ratio R of t_{hs} to t_{ns}, the software startup overhead (t_{hs}) at the host, system size, switch size, message length, number of switches, and packet size.

In a real parallel system however, it is unlikely that at any given moment the only traffic in the network is due to a single multicast. A more likely traffic scenario consists of multiple concurrent multicasts in the system. We use such traffic for our second type of experiments. We apply an increasing load consisting of multicast traffic alone and examine the load at which the network saturates with each of the three multicasting schemes under the influence of the various parameters. As in [39], we use effective applied load as a measure of our stimulus. For a multicast of degree m and a load of B_i, the effective applied load is mB_i. We study the performance of two different degrees of multicasts over the range of loads till saturation.

We also study the impact on this type of traffic of three network parameters: the ratio R, the number of switches, and the message length.

5.2 Generating Irregular Topologies

We now describe the method that we adopted for generating different irregular topologies for our experiments. To generate a topology with s k-port switches and p nodes, we reduce the problem to that of generating interconnections among $s \times k - p$ switch ports so that the graph with the switches as vertices remains connected. We assume that all ports of a switch are full duplex (i.e. we use the term 'port' for an input port-output port pair) and that at most one full duplex link runs between a pair of switch ports. Links are not allowed between ports of the same switch.

Depending on a parameter which we call the *percentage connectivity*, we allow a certain number of switch ports to remain unconnected (i.e. they have no attached links). For an interconnection with 100% connectivity, we have a total of $s \times k - p$ ports available, each of which are connected to the port of another switch via a bidirectional link. On the other hand, for a percentage connectivity of 80% we have $(s \times k - p) \times 0.8$ switch ports which are connected to other switches: $(s \times k - p) \times 0.2$ of the switch ports remain unconnected.

We use a random number generator to generate the port and switch to which a given switch port should be connected or to decide if the port should be connected to a processing node. Using this methodology, we generate 10 different topologies for a default connectivity of 80%.

5.3 Single Multicast Performance

We now present our results for the effect of single multicasts on the three different multicasting schemes. One by one, we examine the effect of each parameter on the performance of the schemes. As mentioned in Sec. 1, it must be kept in mind that the multi-phase path-based multicasting scheme can also make use of support at the NI to further enhance multicast performance. However, since we are evaluating the effect of support at the NI versus support at the switches, we assume that under the path-based scheme, every intermediate destination receives the incoming message completely at the host and then retransmits the message to the NI and then onto the network.

5.3.1 Effect of R

We first examine the effect of variation in the ratio R (t_{hs}/t_{ns}) on the performance of the three multicasting schemes. Given our default value of $t_{hs} = 1000$ cycles we vary the value of t_{ns} to take on the values 2000, 1000 (default), 500, and 200 cycles (to generate the following values of R: 0.5, 1, 2, and 5, respectively). Figure 8 presents the results of our experiments. We find here (as well as in the other single multicast experiments to follow) that the tree-based multicasting scheme performs extremely well, since it requires only one message and therefore only one communication phase. The interplay between the NI-based and path-based schemes is more interesting. As the ratio increases (i.e. t_{ns} shrinks relative to t_{hs}), the NI-based multicasting scheme begins to outperform the path-based scheme. This is because although the NI-based scheme involves more communication phases, every communication phase incurs a receive overhead of t_{nr} and a send overhead of t_{ns}. On the other hand, the fewer phases of the path-based scheme each incur a receive overhead of $t_{nr} + t_{hr}$ and a sending overhead of $t_{hs} + t_{ns}$.

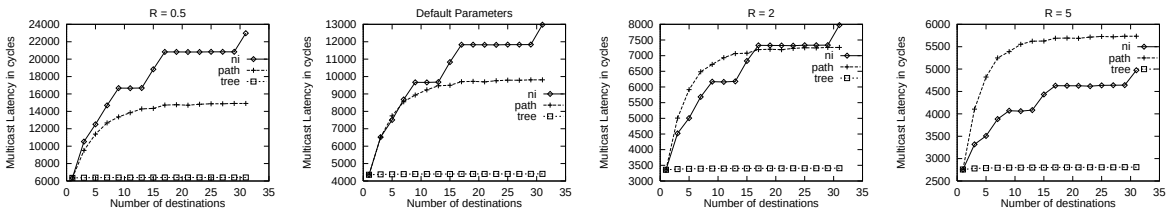

Figure 8: Effect of R on single multicast performance.

5.3.2 Effect of Software Overhead at Host Processor

We also studied the performance of varying the startup overhead at the hosts (t_{hs}) while keeping R at a constant value of 1 (i.e. t_{ns} was changed proportionately with t_{hs}). As can be seen from Fig. 9, we find that the relative performance of the three schemes remains largely unaltered by the change in the value of t_{hs}. This is perhaps intuitive: given that the number of phases for multicast under the three schemes remains unchanged because of the change in the software overhead at the host processor, and that the t_{ns} changes with t_{hs}, the relative performance of the three multicasting schemes should remain unchanged. We therefore conclude that the relative performance of the path-based and NI-based schemes is more sensitive to the ratio R than to the value of t_{hs} (or t_{ns}) alone.

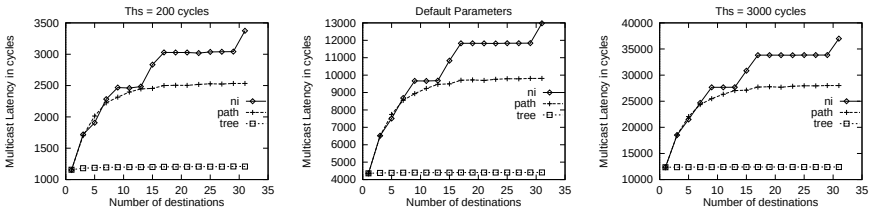

Figure 9: Effect of Software Overhead at Host Processor.

5.3.3 Effect of System Size

To study how multicast performance scales with system size we performed experiments for 4 different system sizes: 32 nodes (default), 64 nodes, 128 nodes, and 256 nodes. With increase in system size we also increased the number of switches so that the percentage connectivity remained 80% and the percentage of switch ports connected to processing nodes remained unchanged. For example, we used eight 8-port switches (64 switch ports in all) for our experiments with a 32 node system, sixteen 8-port switches (128 switch ports in all) for our 64 node experiments, and so on.

Figure 10 presents our results. We find that the tree-based multicast again remains almost unaffected by change in system size. The path-based multicasting scheme is adversely affected by increase in system size. As the system size increases, the number of phases required for the path-based multicasting scheme also increases. Note that there is an increase in the number of phases required to perform multicast to the *same* number of destinations in a larger system than in a smaller system under the path-based scheme. This is because our random choice of multicast destinations spreads the destination set among the larger number of switches present in larger systems. The number of path-based multidestination worms required to cover a given destination set (and hence the number of phases required in the multicast) depends inversely on the average number of multicast destinations per switch. Again, the fact that the path-based scheme incurs send

and receive overhead at both the host and network interface causes its performance to worsen more rapidly than the NI-based multicast, which actually requires more communication phases for performing multicast.

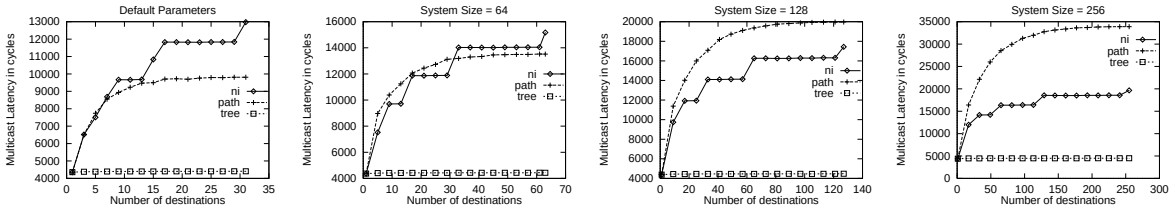

Figure 10: Effect of System Size.

5.3.4 Effect of Switch Size

We studied the effect of increasing switch size on multicast performance by keeping the system size constant (at the default value of 32 nodes), and by increasing the switch size from 8 ports to 32 ports. We decreased the number of switches so that the total number of switch ports remain the same. We therefore have four switches in our experiments with 16-port switches and two switches in our experiments with 32-port switches.

Our results are shown in Fig. 11. We find that increasing switch size favors the path-based scheme since the number of path-based multidestination worms (and the number of phases in the multicast) reduces with increasing switch size. For our experiments with 32-port switches, the path-based and tree-based multicasts perform identically since they require exactly one multidestination worm and a single phase to perform multicast. The performance of the NI-based and tree-based schemes remain unaffected by increase in switch size: the number of phases for both schemes is independent of switch size.

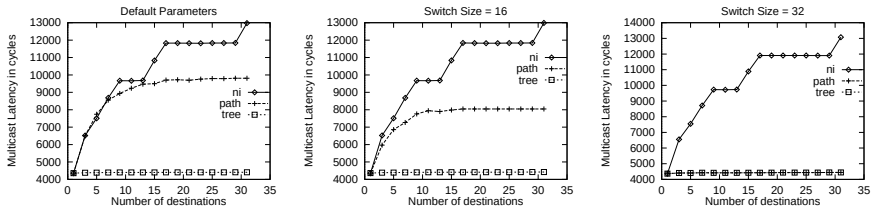

Figure 11: Effect of Switch Size.

5.3.5 Effect of Number of Switches

To see the effect of increase in number of switches on multicast performance we increased the number of switches used while keeping the system size constant. However, all switches had 8 ports. Therefore, unlike the previous experiment, the total number of switch ports are *not* kept constant: they increase with increase in the number of switches.

As the number of switches for a given system size increase, the average number of nodes per switch decreases as does the average number of multicast destinations per switch. The results of Fig. 12 corroborate the results of Sec. 5.3.3, viz., the number of multidestination worms, the number of phases for multicast, and the multicast latency for the path-based scheme increase with decrease in the average number of destinations per switch. The other two multicasting schemes remain largely unaffected by this increase in the number of switches.

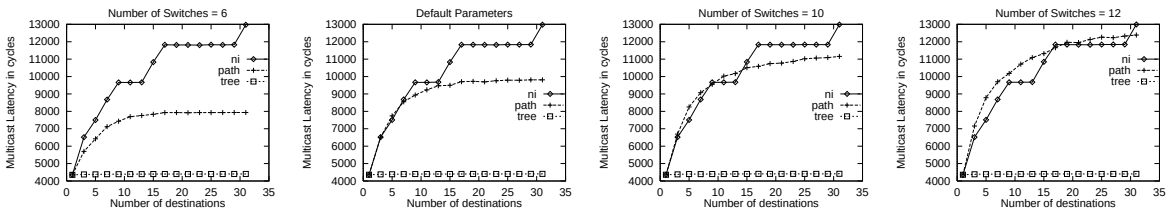

Figure 12: Effect of Number of Switches

It is to be noted that for the NI-based scheme, the average path length changes in each of the last three sections. However, this affects only the propagation time of the worms. Furthermore, since we are using cut-through routing (which is almost "distance independent") this increase in propagation time is negligible.

5.3.6 Effect of Message Length

Figure 13 shows the effect of increasing message length on multicast performance. Here too the path-based scheme begins to perform worse than the NI-based multicast beyond a message length of 512 flits. However, the reason for this decrease in the relative performance of the path-based scheme can be attributed to the increase in the latency of each of its phases: the number of phases remains unchanged. We assume a packet size of 128 flits. Messages larger than this size are split into multiple packets. Under the path-based scheme, a phase finishes only when all the packets of the message arrive at an intermediate destination: only then can the node initiate the path-based multidestination worm of the next phase. On the other hand, under the NI-based scheme (and following the FPFS discipline outlined earlier), a packet can be forwarded to the each of the recipients of the next phase as soon as it arrives at the network interface of an intermediate destination node. The NI-based scheme therefore begins to gain in performance as the number of packets in a message increases.

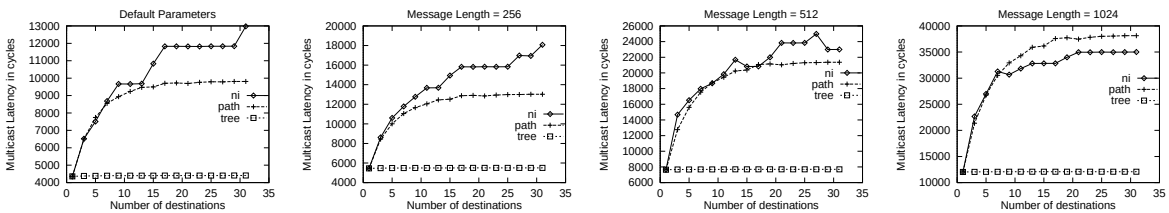

Figure 13: Effect of Message Length

5.3.7 Effect of Packet Size

The results of this section underline the importance of the *number* of packets on the relative performance of the NI-based and path-based schemes. In the experiments of this section, we keep the message length constant at 128 flits but vary the packet size from 32 flits to 128 flits. Figure 14 shows the results. We see that as the number of packets increases, the NI-based scheme begins to perform comparably to the path-based scheme. The graphs for multicast with four packets (a packet size of 32 flits in this section, and the graph for a message length of 512 flits in the previous section) are almost identical in terms of the relative performances of the schemes. We therefore conclude that the *number* of packets in a multicast has greater bearing than the actual message length on the relative performance of the NI-based and path-based multicasts.

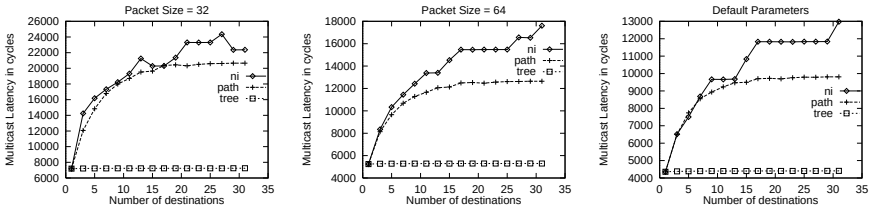

Figure 14: Effect of Packet Size

5.4 Latency versus Applied Load for Multicast

We now present our results for multicast latency under an increasing multicast load for each of the three schemes. We used two different multicast degrees in our experiments: 3-way multicasts (i.e., multicasts with 3 destinations) and 15-way multicasts. For each of our experiments, our simulations were run for at least one million cycles, with measurements beginning after a cold-start time of 500,000 cycles. It is worth keeping in mind that for each of the networks, the maximum unicast throughput (assuming no software overheads and no contention for the I/O bus) was observed to be less than 0.18 using up*/down* routing. Also, each of the plots in this section show multicast latency against *effective applied load* as discussed in Sec. 5.1.

5.4.1 Effect of R

Figure 15 shows the results of our experiments under variation of R. In general, for a value of R less than or equal to 1.0 the NI-based scheme performs worst followed by the path-based scheme. The tree-based scheme performs best for such values of R. However, when R becomes greater than 1.0, we note an interesting trend. Now the NI-based scheme performs comparably to the tree-based scheme and much better than the path-based scheme. A possible reason for this is that the tree-based scheme causes an almost simultaneous reception in all its recipients causing an increase in the contention for resources at the recipient nodes. On the other hand, NI-based scheme "spreads" the receive times among the recipients of the multicasts, causing the performance improvement.

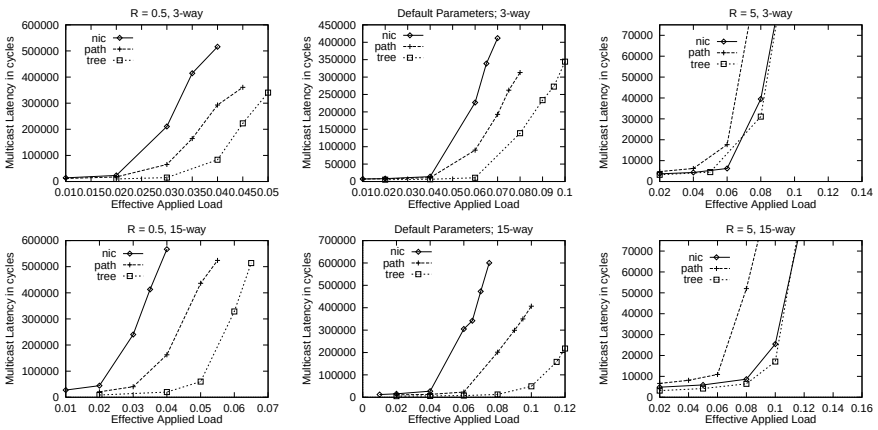

Figure 15: The effect of parameter R on the latency of multicasts under increasing multicast load for 3-way and 15-way multicasts.

5.4.2 Effect of Number of Switches

Our experiments with single multicasts in systems with increasing number of switches has shown that the path-based scheme begins to perform worse as the number of switches in the system increases. We observe a similar trend for our results with multiple multicast traffic (shown in Fig. 16). As the number of switches increases, the saturation load for the path-based scheme approaches that of the NI-based scheme. However, the NI-based scheme results in a greater amount of traffic and higher contention in the network. The tree-based multicast performs almost uniformly well with increase in the number of switches and saturates much later than the other two schemes.

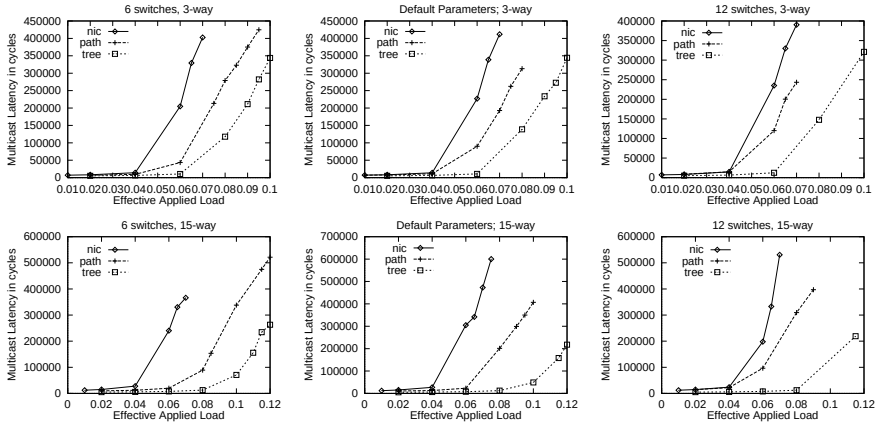

Figure 16: The effect of the number of switches on the latency of multicasts under increasing multicast load for 3-way and 15-way multicasts.

5.4.3 Effect of Message Length

Our results for multicast performance under increasing message length are shown in Fig. 17. The results show that the tree-based multicasting scheme performs best for all message lengths. Furthermore, as was noted for single multicasts, the performance of the NI-based and path-based schemes become comparable as the message lengths increase. However, for a single multicast with a message length of 1024 flits we observed that the NI-based scheme performs better than the path-based scheme. Under multiple multicast traffic, the NI-based scheme performs worse (has a lower saturation point and higher latencies) than the path-based scheme for this value of message length, especially for large multicast degrees. This is because the NI-based scheme involves more communication phases and results in more traffic than the path-based scheme, thereby increasing the contention in the network.

6 Related Work

A lot of recent research has concentrated on proposing enhanced multicast mechanisms. However, very little of this work has concentrated on irregular networks.

A simple multi-phase approach to perform multicast in *wormhole-routed* irregular networks so that contention among the constituent messages of the multicast is reduced/minimized has been proposed by Kesavan et al. [13]. Verstoep et al. [41] have proposed (and implemented) a better method for performing multicast in switch-based systems using network interface support for absorbing and retransmitting messages. However,

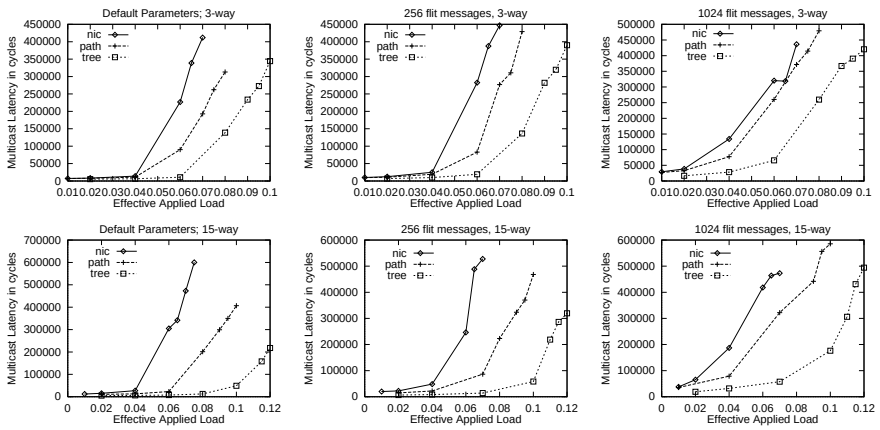

Figure 17: The effect of the message length on the latency of multicasts under increasing multicast load for 3-way and 15-way multicasts.

the scheme uses only a heuristically derived binary tree to order the destinations into various phases of the multicast. The scheme makes no optimizations for the case where multiple packets may be transmitted in a message and where each of the packets have to be absorbed and retransmitted into the network.

Multicast using switch support for replication has been proposed in the context of regular networks. Chiang and Ni [4] proposed a method for performing multicast in wormhole routed MINs using tree-based *multi-head* worms with *synchronous* replication. Similarly, path-based multicast was originally proposed in the context of mesh networks by Lin and Ni [17], and was enhanced to a base routing conformed path model by Panda et al. [28]. Extensions of the latter method to support multiple multicast more efficiently have been proposed by Kesavan and Panda [14].

None of these papers have evaluated the alternative enhanced multicasting schemes in comparison to one another in the context of irregular networks and in the presence of multiple multicast traffic.

7 Conclusion

In this paper, we have compared three schemes for performing multicast efficiently in switch-based irregular networks. After describing the three schemes, we compared them analytically and using simulation experiments.

We find that the tree-based multicasting scheme performs better than the path-based and NI-based schemes. The relative performance of the path and NI-based schemes is sensitive to a number of parameters. The most important of these parameters is the ratio R of overhead at the host to the overhead at the NI. We find that the path-based scheme performs better than the NI-based scheme for values of R less than 1, smaller system sizes, larger switch sizes, fewer switches for a given system size, and for multicasts with fewer packets. In all other cases, the NI-based scheme outperforms the path-based scheme.

Since a wealth of research has focussed on more efficient network interfaces, the value of R is likely to rise in the future. It is also important that the performance of multicast scale with increasing system size and with increase in the number of switches. We therefore conclude that support for multicast at the NI is an important first step to improving multicast performance. However, there is still considerable gain that can be achieved by supporting hardware multicast in switches. In particular, unlike with the NI-based schemes, the performance of the switch-based multicasting schemes is able to scale with the trend of increasing switch

size. Finally, while supporting such hardware multicast, it is better to support schemes that can achieve multicast in one phase even at a (perhaps) additional cost.

Additional Information: A number of related papers and technical reports can be obtained from http://www.cis.ohio-state.edu/~panda/pac.html.

References

[1] B. Abali. A Deadlock Avoidance Method for Computer Networks. In *Proceedings of the First International Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC '97).* , pages 61–72, February 1997. Available as `Lecture Notes in Computer Science #1199, Springer-Verlag`.

[2] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 96–107, April 1991.

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

[4] C. M. Chiang and L. M. Ni. Deadlock-Free Multi-Head Wormhole Routing. In *Proceedings of the First High Performance Computing-Asia*, 1995.

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

[6] D. Dai and D. K. Panda. Reducing Cache Invalidation Overheads in Wormhole DSMs Using Multidestination Message Passing. In *International Conference on Parallel Processing*, pages I:138–145, Chicago, IL, Aug 1996.

[7] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach.* The IEEE Computer Society Press, 1997.

[8] E. W. Felten, R. A. Alpert, A. Bilas, M. A. Blumrich, D. W. Clark, S. N. Damianakis, C. Dubnicki, L. Iftode, and K. Li. Early Experience with Message-Passing on the SHRIMP Multicomputer. In *International Symposium on Computer Architecture (ISCA)*, pages 296–307, 1996.

[9] David Garcia and William Watson. Servernet II. In *Proceedings of the 2nd Parallel Computer Routing and Communication Workshop*, June 1997.

[10] R. Horst. ServerNet Deadlock Avoidance and Fractahedral Topologies. In *Proceedings of the International Parallel Processing Symposium*, pages 274–280, 1996.

[11] Intel Corporation. *Intel iPSC System Overview*, 1986.

[12] Intel Corporation. *Paragon XP/S Product Overview*, 1991.

[13] R. Kesavan, K. Bondalapati, and D. K. Panda. Multicast on Irregular Switch-based Networks with Wormhole Routing. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA-3)*, pages 48–57, February 1997.

[14] 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.

[15] R. Kesavan and D. K. Panda. Multicasting on Switch-based Irregular Networks using Multi-drop Path-based Multidestination Worms. In *Proceedings of the 2nd Workshop on Parallel Computer Routing and Communication (PCRCW '97)*, pages 179–192, June 1997.

[16] R. Kesavan and D. K. Panda. Optimal Multicast with Packetization and Network Interface Support. In *Proceedings of International Conference on Parallel Processing*, pages 370–377, Aug 1997.

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

[18] 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.

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

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

[21] 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.

[22] Meiko Limited. *Meiko CS-2 System Overview*, 1994.

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

[24] Lionel Ni. Should Scalable Parallel Computers Support Efficient Hardware Multicasting? In *ICPP Workshop on Challenges for Parallel Processing*, pages 2–7, 1995.

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

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

[27] D. K. Panda, D. Basak, D. Dai, R. Kesavan, R. Sivaram, M. Banikazemi, and V. Moorthy. Simulation of Modern Parallel Systems: A CSIM-based approach. In *Proceedings of the 1997 Winter Simulation Conference (WSC'97)*, pages 1013–1020, December 1997.

[28] D. K. Panda, S. Singal, and R. Kesavan. Multidestination Message Passing in Wormhole k-ary n-cube Networks with Base Routing Conformed Paths. Technical report, The Ohio State University, December 1995. *IEEE Transactions on Parallel and Distributed Systems*, to appear.

[29] D. K. Panda, S. Singal, and P. Prabhakaran. Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 131–145, 1994. Available as `Lecture Notes in Computer Science #853, Springer-Verlag`.

[30] 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.

[31] W. Qiao and L. M. Ni. Adaptive Routing in Irregular Networks Using Cut-Through Switches. In *Proceedings of the International Conference on Parallel Processing*, pages I:52–60, Chicago, IL, Aug 1996.

[32] M. D. Schroeder et al. Autonet: A High-speed, Self-configuring Local Area Network Using Point-to-point Links. Technical Report SRC research report 59, DEC, Apr 1990.

[33] F. Silla, M. P. Malumbres, A. Robles, P. Lopez, and J. Duato. Efficient Adaptive Routing in Networks of Workstations with Irregular Topology. In *Proceedings of the First International Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC '97)*, pages 46–60, February 1997. Available as `Lecture Notes in Computer Science #1199, Springer-Verlag`.

[34] R. Sivaram, D. K. Panda, and C. B. Stunkel. Efficient Broadcast and Multicast on Multistage Interconnection Networks using Multiport Encoding. *IEEE Transactions on Parallel and Distributed Systems.* In Press.

[35] R. Sivaram, D. K. Panda, and C. B. Stunkel. Multicasting in Irregular Networks with Cut-Through Switches using Tree-Based Multidestination Worms. In *Proceedings of the 2nd Parallel Computer Routing and Communication Workshop (PCRCW '97)*, pages 35–48, June 1997.

[36] R. Sivaram, C. B. Stunkel, and D. K. Panda. A Reliable Hardware Barrier Synchronization Scheme. In *Proceedings of the 11th IEEE International Parallel Processing Symposium*, pages 274–280, April 1997.

[37] R. Sivaram, C. B. Stunkel, and D. K. Panda. HIPIQS: A High Performance Switch Architecture using Input Queuing. In *Proceedings of the 12th International Parallel Processing Symposium*, April 1998. To appear.

[38] C. B. Stunkel, D. Shea, D. G. Grice, P. H. Hochschild, and M. Tsao. The SP1 High Performance Switch. In *Scalable High Performance Computing Conference*, pages 150–157, 1994.

[39] C. B. Stunkel, R. Sivaram, and D. K. Panda. Implementing Multidestination Worms in Switch-Based Parallel Systems: Architectural Alternatives and their Impact. In *Proceedings of the 24th IEEE/ACM Annual International Symposium on Computer Architecture (ISCA-24)*, pages 50–61, June 1997.

[40] Craig B. Stunkel, D. G. Shea, B. Abali, et al. The SP2 High-Performance Switch. *IBM System Journal*, 34(2):185–204, 1995.

[41] 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.

[42] T. von 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.

[43] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *International Symposium on Computer Architecture*, pages 256–266, 1992.

[44] H. Xu, Y.-D. Gui, and L. M. Ni. Optimal Software Multicast in Wormhole-Routed Multistage Networks. In *Proceedings of the Supercomputing Conference*, pages 703–712, 1994.