Architectural Support for Efficient Multicasting in Irregular Networks*

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

*IBM Power Parallel Systems
522 South Road, P963
Poughkeepsie, NY 12601
Phone: (914) 433-2913
Email: rsivaram@us.ibm.com

†Dept. of Computer and Information Science
The Ohio State University
Columbus, OH 43210
Phone: (614) 292-5199, Fax: (614) 292-2911
Email: {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

Abstract

Parallel computing on networks of workstations is fast becoming a cost-effective high-performance computing alternative to MPPs. Such a computing environment typically consists of processing nodes interconnected through a switch-based irregular network. Many of the problems that were solved for regular networks have to be solved anew for these systems. One such problem is that of efficient multicast communication. In this paper, we propose two broad categories of schemes for efficient multicasting in such irregular networks: *network interface-based* (NI-based) and *switch-based*. The NI-based multicasting schemes use the network interface of intermediate destinations for absorbing and retransmitting messages to other destinations in the multicast tree. In contrast, the switch-based multicasting schemes use hardware support for packet replication at the switches of the network and a concept known as *multidestination routing* to convey a multicast message from one source to multiple destinations.

We first present alternative schemes for efficient multi-packet forwarding at the NI, and derive an optimal k-binomial multicast tree for multi-packet NI-based multicast. We then propose a spectrum of switch-based multicasting schemes that differ in the power of the encoding scheme, and the complexity of the decoding logic at the switches. These multicasting schemes range from using multidestination worms that can only cover the nodes of a single switch, to path-based multidestination worms that can cover all nodes connected to switches along a valid unicast path, to tree-based multidestination worms that can cover entire destination sets in a single phase using one worm. For each scheme, we describe the associated header encoding and decoding operation, the method for deriving multidestination worms that cover arbitrary multicast destination sets, and the multicasting scheme using the derived multidestination worms. We then compare the NI-based multicasting scheme to the switch-based multicasting schemes with path-based and tree-based multidestination worms using simulation to determine the system parameters that affect each of the schemes, and the range of system parameters for which each scheme performs best.

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. A preliminary version of this paper has been presented at ICPP '98 [40].

Contents

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. Switch-based networks of commodity workstations present an attractive alternative platform for cost-effective high-performance computing. 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 [14], torus [7], hypercube [13], multistage interconnection network (MIN) [26, 47], 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 [9]. 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 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, 36, 38]) 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 (2 or more) processes [24, 30]. Examples of collective operations include multicast [28], barrier synchronization [44], reduction, etc. The importance of these operations is underlined by the inclusion of several primitives for collective communication in the Message Passing Interface (MPI) standard [27]. Some collective operations are also used for system level operations in distributed shared memory systems, such as for cache invalidations, acknowledgment collection, and synchronization [8].

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 that use a multi-phase approach have been proposed for systems with regular [25, 52] and irregular topologies [16, 21]. In this approach, first, the source sends out a multicast message to one of its destinations. In subsequent phases, the source and the destinations that have received the message act as secondary sources and forward the message to other destinations. Such an approach 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. However, even the best multicast algorithms using this approach need at least $\lceil log_2(n+1) \rceil$ steps to complete. This is due to the fact that unicast communication delivers a message to only a single destination in a step. Thus, techniques which allow the delivery of a message to multiple destinations in a single step can provide substantial performance improvement.

One such technique can be proposed by augmenting the firmware support at the *network interface* (NI). Each node in modern switch-based systems has a NI plugged into the I/O bus, which connects to a port in a switch. The NI has a processor, some memory, DMA engines, and most importantly, firmware which executes on the NI processor. Modern networks allow the firmware to be modified and downloaded at system initialization [4]. This modification of the firmware can be done in an intelligent fashion so that the NI can recognize multicast packets and handle the forwarding of packets from intermediate destinations without involving the host processor [49]. Using such a technique, the source node can initiate a multicast and the NIs of the participating nodes can cooperate to get the message delivered to all members of an arbitrary destination set. Thus, the significant software overhead at each intermediate destination host processor can eliminated. However, there are several issues that must be addressed when proposing enhanced multicasting using such support. For example, the choice of the forwarding mechanism at each intermediate node, the buffering requirement at the NIs, etc., are issues associated with the implementation of this technique. Furthermore, most current day systems break up messages longer than a given maximum packet size into multiple packets. It is not clear what multicast trees will be optimal for efficient multicast of such multi-packet messages [6] using support at the NIs of the participating nodes, and deriving such an optimal tree for multi-packet multicast is a challenging problem.

Another technique to allow delivery of a message to multiple destinations with reduced software overhead can be proposed by augmenting the hardware support at the switch. Prior work on traditional regular topology parallel systems has shown how a message can be sent to multiple destinations in a single step (called a *multidestination message*) [22, 32, 33]. Some modern cut-through switches possess the capability of simultaneous replication of an incoming message to multiple output ports [37]. Although replication of messages in cut-through switches is deadlock-prone, deadlock-free replication methods have been proposed [43, 48]. This replication capability can be utilized to deliver a message to multiple destinations while incurring a single software overhead for sending the message. Using such a technique, the source node can initiate a multicast message and the switches in the network can deliver the message to multiple destinations. However, there are several problems associated with proposing enhanced multicasting using such support. For example, efficient header encoding/decoding schemes need to be designed keeping in mind that the new messages must conform to the base (unicast) routing supported by the system. (Such a requirement prevents multicast messages from introducing additional deadlock scenarios [32].) Furthermore, the tradeoffs associated with the cost and complexity of the encoding/decoding schemes need to be evaluated against the ability of the corresponding multicast messages to cover arbitrary multicast destination sets. For encoding schemes that are unable to capture arbitrary destination sets within a single worm header, alternative multi-phase algorithms need to be proposed to cover arbitrary multicast destination sets efficiently.

In addition, enhanced multicasting schemes which use the above mentioned techniques need to be evaluated. In particular, there is a need to compare the effect of improved support for multicast at the switches to support for multicast at the NI to decide which of these schemes results in better performance and when. A number of questions need to be answered, such as the following: How quickly can multicast be performed with support at the NI and low overhead messaging layer support? How does multicast with switch/router support (alone) compare with multicast implemented with NI support? For what range of system parameters does it make sense to use one over the other?

The goal of this paper is to propose and evaluate enhanced multicasting schemes to further improve multicast performance. As described above, the proposed schemes can be classified into two main categories: (a) *network interface-based* or *NI-based multicasting schemes* and (b) *switch-based multicasting schemes*. First, we describe the system model, traditional multicasting support, and the above classification of the proposed schemes for improved multicasting. Then, we propose and evaluate *smart NI support*, and two implementations of smart NI support: First-Child-First-Served (FCFS) and First-Packet-First-Served (FPFS). It is shown that the FPFS NI support is more practical and efficient in terms of buffer requirement and ease of implementation. We then consider multicasting with FPFS NI support and show that the binomial tree is not optimal for an arbitrary multicast set size and an arbitrary number of packets. To provide optimal multicast with FPFS NI support we develop a new concept called k-binomial trees and show them to be optimal for multi-packet multicast. We also present a method to construct contention-free k-binomial trees.

Next, switch-based multicasting schemes are examined in detail. We extend the concept of multidestination messaging to irregular switch-based networks and briefly examine related issues such as header encoding/decoding, replication mechanisms, and routing. We present two subclasses of switch-based multicasting schemes: multi-phase [19, 42] and single-phase [42]. The multi-phase 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 [19]. Two multi-phase multicasting schemes using multidestination routing are described which differ in the type of encoding used for their multidestination headers, and which require more than one multidestination worm to cover arbitrary multicast destination sets. We present the concept of path-based multidestination routing on which one of these schemes is based and develop a number of the associated theoretical issues. We also present algorithms to organize these multidestination worms into multiple phases to perform efficient multicast. In the single-phase multicasting scheme, multicast can be performed in a single phase using one multidestination worm from the source node to all the destinations [42, 48]. We present a multicasting scheme that uses a single bit-string encoded multidestination worm.

We then consider three schemes for comparing the relative performance of the enhanced multicasting schemes: the NI-based multicasting scheme using the k-binomial tree, the multi-phase multicasting scheme using path-based multidestination worms, and the multicasting scheme that uses a single multidestination worm. 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. It is clear that a multicasting scheme with enhanced support at the NI *and* the switches will perform better than a scheme that makes use of support at *either* the NI *or* the switches. Therefore, to clearly compare NI-based schemes with switch-based schemes, we assume no smart NI support for the switch-based multicasting schemes in our simulation experiments, i.e., every communication phase under the tree-based and path-based schemes incurs software overhead at the host and NI of the source and (intermediate) destinations.

Our analysis and simulations show that the single-phase switch-based multicasting scheme studied in this paper is the most powerful 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 multi-phase switch-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. Section 2 presents the network model and the underlying routing algorithm assumed in this paper. Then, Sec. 3 presents the communication support in a NOW environment and the classification of the multicasting schemes proposed in this paper. Smart NI support and the proposed NI-based scheme is discussed in Sec. 4. Issues in switch support for multidestination routing and various switch-based multicasting schemes are described in Sec. 5. An qualitative and an exhaustive simulation based comparison of the three schemes is presented 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 [4, 37]. 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.

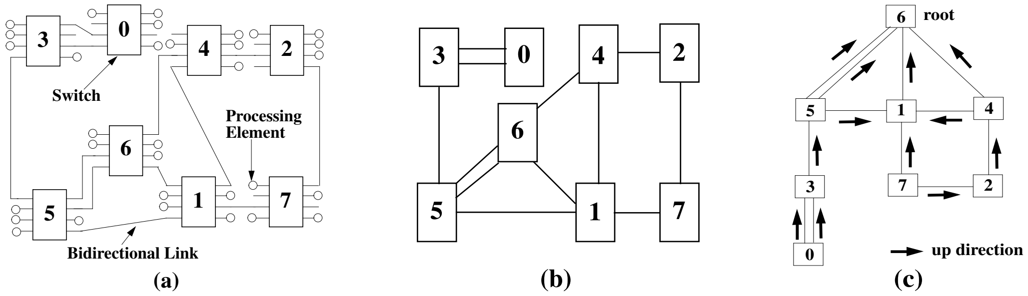

Figure 1: a) An example system with switch-based interconnect and irregular topology; b) corresponding interconnection graph G; c) corresponding BFS spanning tree rooted at node 6.

2.2 Routing Issues

Several deadlock-free routing schemes have been proposed in the literature for irregular networks [4, 12, 36, 37]. Without loss of generality, in this paper we assume the routing scheme for our irregular network to be similar to that used in Autonet [37] due to its simplicity and its commercial implementation. The Autonet routing allows adaptivity, and is deadlock-free. However, the multicasting solutions proposed in this paper can be applied to irregular switch-based networks with other routing schemes.

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 [37]. This routing is also referred to as *up*/down** routing or UD routing.

3 Multicasting Approaches

In this section, we present an overview of the various 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.

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 NICs at the hosts. The software enables the sending of messages through this hardware, and implements the desired communication protocols.

Let us first consider the switches and the NIs which comprise the hardware. 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 [4, 37, 45, 46, 47].

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 NI. This allows sophisticated processing of network transactions to be off-loaded to the NI processor, thus freeing up the host processor for computation. The NI also contains some amount of memory, up to three 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. One DMA engine is 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.

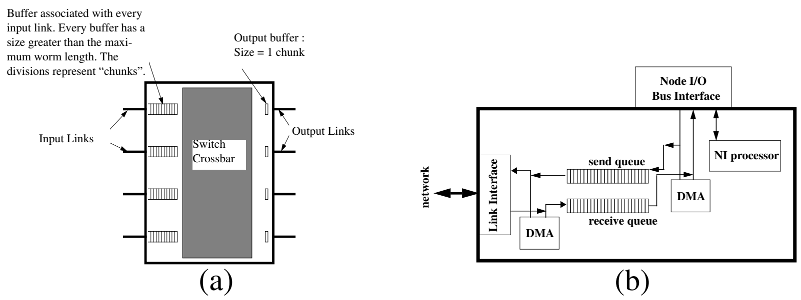

Figure 2: Generic diagrams of (a) a typical cut-through switch and (b) a typical network interface at a workstation.

Now, let us consider the software portion of the communication support. This software portion 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, to support efficient buffer utilization in the network and the NIs, and to minimize the effect of long messages on short message latency. 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 NI. Finally, the software is often responsible for ensuring that messages are reliably delivered, although current day network hardware also guarantee a high degree of reliability [4, 11].

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 actual latency of the message through the physical network. This has led to a considerable amount of research devoted to the design and development of lightweight communication software.

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 NI memory into 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. A 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 [23, 51], U-Net [50], Fast Messages (FM) [29], and SHRIMP [3, 10].

Let us examine how these lightweight messaging systems achieve message transfer. An application is typically linked to a communication library, 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 into the send queue of the NI using programmed I/O [29]. Alternatively, the host processor copies data into the host DMA memory, writes the message pointers to memory-mapped NI memory, and the NI processor uses DMA to copy the packets to the send queue [50]. The DMA setup overhead makes the programmed I/O method more efficient for small message sizes, but DMA may prove 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 NI. The NI 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 for transfer of data between host memory and NI 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 [16, 24, 25, 35] 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. It can be easily observed that $\lceil log_2(n+1) \rceil$ communication steps are required for such a binomial tree based hierarchical multicast to be completed [25]. A communication step is the time required for a message to be 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.

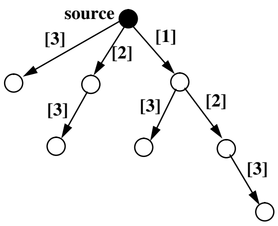

Figure 3: Example of a hierarchical multicast algorithm on a destination set size of 7.

Much recent research has concentrated on improving multicast mechanisms. A simple multi-phase approach to perform multicast in *wormhole-routed* irregular networks with reduced/minimized contention among the constituent messages of the multicast has been proposed by Kesavan et al. [16]. Verstoep et al. [49] have proposed (and implemented) a method for performing multicast in switch-based systems using NI support for absorbing and retransmitting messages. However, the scheme uses only a heuristically derived binary multicast tree. The scheme makes no optimizations for the case where multiple packets may be transmitted in a message and where each of the packets has to be absorbed and retransmitted into the network.

In this paper, we focus on two main directions for improving multicast mechanisms.

3.4 Enhanced Multicasting Schemes

The two major directions in which enhancements can be proposed for improving multicast latency are: (a) increasing the functionality of the software/firmware running at the NI processor (henceforth called NI software), and (b) supporting hardware multicast at the switch.

3.4.1 Multicasting using Smart Network Interface Support

The NI software controls the sending/receiving of packets into/from the network. A packet received from the network is copied from the receive queue of the NI to host memory using DMA. A packet to be sent is copied from the host memory to the send queue in the NI using DMA, and 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 NI 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 NI, 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.

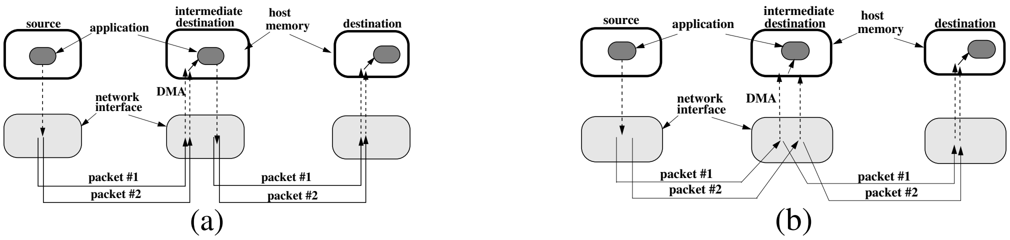

Figure 4: Forwarding of a 2-packet long multicast message by intermediate node using (a) conventional NI support, and (b) smart NI support.

Smart NI support can reduce this overhead for multicast, especially for multi-packet messages. Current lean messaging layers allow modification of the NI software. This software can be modified to allow it to identify a multicast packet [2, 15, 20, 49]. 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 an example of such forwarding with smart NI support. An example of the use of such a smart NI has been described in [49]. A detailed discussion on the use of such smart NI support, the related implementation and architectural issues, and optimal algorithms for multicast is given in Section 4.

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 [22, 33]. 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. The deadlock-free replication method has been proposed in [39, 43, 48] that depends on the guarantee that the multicast packet can eventually be completely buffered at the switch. 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 each input buffer uses a single FIFO queue to store packets (as in the switches of [11]).

Given such support for replication at the switches, a number of schemes can be proposed for carrying out multicast in switch-based irregular networks. These 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 UD 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 [32]. A detailed discussion of these schemes, the related implementation issues, and efficient algorithms for performing multicast are given in Section 5.

4 Optimal Multicasting using Smart Network Interface Support

The concept of smart NI support has been described in Section 3.4.1. In this section, we present a detailed discussion on building optimal multicast trees using such support. We first estimate the latency of a multicast operation using smart NI. Next, we discuss two implementations of smart NI support and compare them. Finally, we present an optimal multicast algorithm using the better of the two implementations.

4.1 Multicasting over Smart Network Interface

Let us estimate the latency of a multicast operation using smart NI 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 at the host processor are large, they are independent of the choice of the multicast tree. However, the overhead incurred at the NIs 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 NIs 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.

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 NI support. Figures 5(a) and 5(b) show the multicast over conventional and smart NI, respectively. It can be easily observed that the multicast latencies using conventional and smart NIs are $2(t_{hs} + t_{step} + t_{hr})$ and $(t_{hs} + 2t_{step} + t_{hr})$, respectively. For an arbitrary multicast set size of n nodes, these values will be $\lceil log_2 n \rceil (t_{hs} + t_{step} + t_{hr})$ and $(t_{hs} + \lceil log_2 n \rceil t_{step} + t_{hr})$, respectively. Therefore, multicast latency can be lowered significantly by using smart NI support.

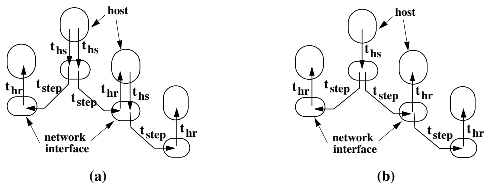

Figure 5: Performance benefits of the smart network interface: (a) binomial single-packet multicast tree over the conventional NI and (b) binomial single-packet multicast tree over the smart NI.

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. We consider alternative implementations of the NI support for packet forwarding in the next section.

4.2 Implementations of Smart Network Interface Support for Packet Forwarding

There are two possible implementations of the smart NI support: First-Child-First-Served (FCFS) and First-Packet-First-Served (FPFS). Here we describe and compare both implementations, and show why the FPFS implementation is more efficient and practical.

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

In this implementation, the NI 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 NI 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 NI, 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 NI forwards the message to the second child, followed by the third child, and so on. Figure 6(a) formally expresses this implementation in a pseudo-code format.

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

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. Figure 6(b) formally expresses this implementation in a pseudo-code format.

(a)

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

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

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

(b)

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

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

Figure 6: Pseudo-code description of (a) the FCFS and (b) the FPFS implementations of the smart NI for multicast.

4.2.3 Comparison of FCFS and FPFS Implementations

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

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 NI at an intermediate node. To implement FCFS, the NI 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 NI processor handles the forwarding of the multicast message on a per-packet basis. When the NI processor reads the header of a multicast packet from the receive queue, it forwards the packet to all its children in the multicast tree. The NI processor does not have to maintain a counter for each incoming multicast message. Therefore, the FPFS is an easier implementation than the FCFS.

Buffer Requirement at NI: It can be quantitatively 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 NI memory to the network. Let us consider the time interval starting from when the NI processor reads 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 NI 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. Therefore, $T_c = (p-i+1)t_{ns} + (k-2)pt_{ns} + it_{ns}$. In the FPFS implementation, a packet only needs to be buffered at the NI 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. Therefore, $T_p = kt_{ns}$. 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. Next, we develop optimal multicast trees for systems with FPFS support.

4.3 Optimal Multicast with FPFS

In this section, we first describe the problem of optimal multicast. We then propose an optimal multicast tree on a system with FPFS NI support and discuss related implementation details.

4.3.1 Non-optimality of Binomial Tree

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 NI support [25]. However, it is not clear whether this is true for systems with smart NI support. Let us consider an example multicast of a 3-packet message to three destinations on a system with smart network interface support. Figures 7(a) and 7(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 NI support.

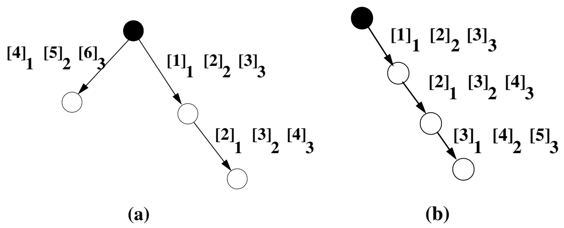

Figure 7: The number of steps to complete multicast of a 3-packet long message to 3 destinations using: (a) a binomial tree and (b) a linear tree.

4.3.2 A Pipelined Model for FPFS Multicast

The discussion in Section 4.1 clearly shows that multicast latency for a single packet on a system with smart NI support can be written as $(t_{hs} + num\_steps * t_{step} + t_{hr})$. The same formula can be extended to multi-packet multicast latency where t_{hs} (t_{hr}) denote the send (receive) overhead at the host processor to transfer the message to (from) the NI. In this section, we analyze multicast latency in terms of *steps* occurring at the NI layer, as discussed in Section 4.1.

Let us first model the multicast latency at the NI 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. We generalize this basic concept in the following discussion to accurately model FPFS multicast latency.

In the following discussion, we develop the theory to prove that multicast of m packets can be modeled as m pipelined multicasts of 1 packet each. First, the delay between the arrival of consecutive packets of a multicast at a node is calculated. Then, it is shown that this interval is dependent only on the number of

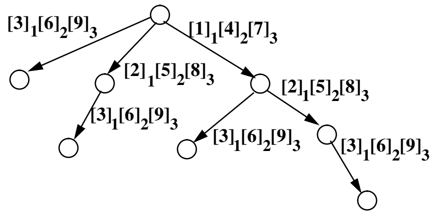

Figure 8: The break up of a 3-packet multicast over 7 destinations using a binomial multicast tree.

children the root (of the multicast tree) has. Thus, a pipelined model of an m-packet multicast is developed. When designing multicast trees, it is a desirable feature that a vertex in a multicast tree does not have a larger number of children than any of its ancestors. This is obvious since vertices higher up in the tree get the message earlier, and can therefore cover more destinations. We limit the following discussion to such trees where the root has the maximum number of children.

Let $T = (V, E)$ be a multicast tree, and let r be the root vertex of T. Let the function $\delta : V \rightarrow N$ be such that $\delta(v)$ be the number of children v has in T. Starting at time zero, let L_i denote the time at which the multicast of ith packet is completed, i.e. the time at which the ith packet has been received by the NIs of each destinations. Let the set $Ch(v)$ be defined as the set of vertices which are children of the node v in T, and let the set $Lv(T)$ be the set of leaves of T. We now define and quantify the delay between arrival of consecutive packets of a multicast at a node.

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

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

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

Lemma 2 *The function $delay_i(v)$ for any vertex v $(v \neq r)$ in T is equal to $\delta(r)$.*

Proof: Since r is the root of T, and the source of the multicast $delay_i(r) = 0$. It can be easily observed that $\forall x \in Ch(r) delay_i(x) = \delta(r)$. Similarly, if the $delay_i$ value is calculated for all vertices of T by iteratively applying the equation in Lemma 1, it can easily be shown that $\forall v \in V \; delay_i(v) = \delta(r)$. ∎

Now, we try to derive a relationship between the *delay* function and the interval between the completion of multicasts of consecutive packets, i.e. the interval $(L_{i+1} - L_i)$.

Lemma 3 *The time interval $(L_{i+1} - L_i)$, i.e. the time between the completions of multicast of any two successive packets of a multicast, is equal to the function $delay_i(v_l)$ where v_l is the is the last vertex in T which receives the multicast packets.*

Proof: Since each packet of the multicast follows the same pattern as the previous one, there is one fixed vertex, v_l, of T which is the last vertex to receive each multicast packet. The time at which the multicast of the ith packet is completed, L_i, is determined by the time at which v_l receives the ith packet. The value of $delay_i(v_l)$ determines the interval between the arrivals of the ith and the $(i+1)$th packet at v_l. Therefore, $delay_i(v_l)$ determines the time interval $(L_{i+1} - L_i)$. ■

The results from Lemmas 1, 2, and 3, lead to the following theorem.

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

Proof: Follows from the above discussion. ■

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 $\delta(r)$ steps after the completion of the previous one. We will refer to $\delta(r)$ as δ_r in the remainder of this discussion. 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 the m pipelined single-packet multicasts is $L_1 + (m-1)\delta_r$ steps.*

Proof: Follows from the above discussion. ■

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 $3 + (3-1) * 3$.

4.3.3 Optimal k-binomial Trees

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), $\delta_r = 1$ 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 [25] 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 2 *A k-binomial tree is defined as a recursively doubling tree where each vertex has at most 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. Let $N(s,k)$ denote the number of nodes covered in s steps by a k-binomial multicast tree. For the boundary condition $k = 0$, we fix $N(s,k) = 1$ which denotes that the source node already has the message to be multicast.

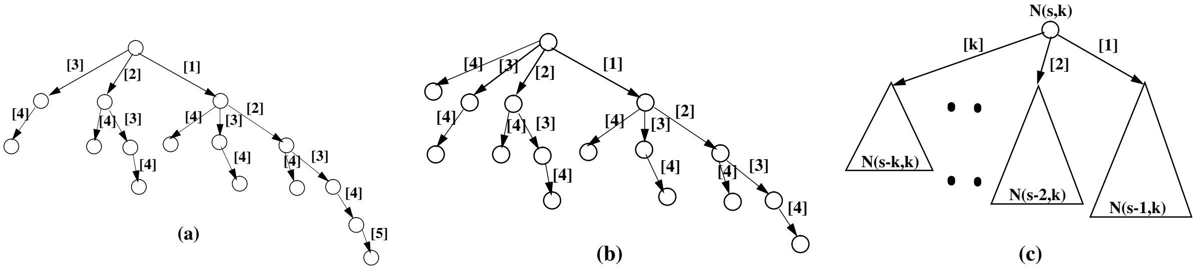

Figure 9: Examples of k-binomial trees on a multicast set size of 16: (a) the 3-binomial tree, and (b) the 4-binomial tree. (c) The number of nodes covered by a k-binomial tree in s steps when $s > k$. The number of nodes in the ith subtree from right is given by $N(s-i,k)$

Lemma 4 *The value of $N(s,k)$ is given by*

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

Proof: If $s \le k$, the k-binomial tree is like a binomial tree, so $N(s,k) = 2^s$. For the case of $s > k$, Fig. 9(c) 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 $(s-1)$. Similarly, 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). ∎

Thus, for a given δ_r and n, the value of L_1 is the minimum value of s such that $N(s,\delta_r) \ge n$. 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 optimal multicast tree is that k-binomial tree which produces the minimum value of $L_1 + (m-1)k$, where $1 \le k \le \lceil log_2 n \rceil$, and L_1 is equal to the minimum value of s for which $N(s,k) \ge n$.*

4.3.4 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.

Precomputation of Optimal k: For given n and m, it can be shown using Theorem 3 and Lemma 4 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. 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 [20]. Thus, this table requires less than $O(mn)$ memory. Therefore the precomputation of the optimal value of k for a given range of n and m is a feasible implementation.

Contention-free k-binomial Trees: For optimal multi-packet multicast performance, the multicast tree should be *depth contention-free* [25]. 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 [25]. 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, even for the boundary condition $x = y$. Without loss of generality, let us assume that the source of the multicast is the first node in the ordering. Figure 10 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 using Lemma 4 and 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.

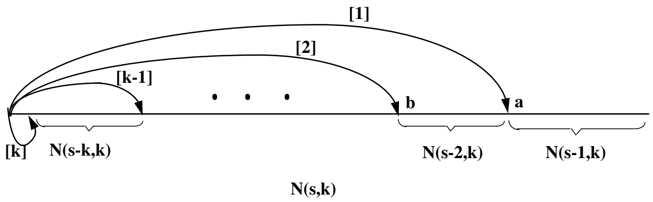

Figure 10: The construction of contention-free k-binomial tree from a given contention-free ordering of participating nodes.

This construction can be applied to different types of systems. For k-ary n-cubes, the dimension-ordered chain [25] 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 [16]. A concept of Partial Ordered Chain (POC) has been proposed to create an ordering with minimal contention on these networks. Such an ordering can be used to construct k-binomial trees with minimal contention on irregular switch-based networks.

5 Efficient Multicasting using Switch Architecture Enhancements

In the previous section we have described a method to improve multicast performance using enhancements to smart NIs to support reception and forwarding of messages. We now examine methods to improve multicast performance by making use of architectural enhancements at the switches to support a concept known as multidestination routing. We first present the basic concepts behind multidestination routing. We then present a spectrum of multicasting schemes using multidestination routing which differ in the complexity of the encoding/decoding operations that need to be supported and in the number of phases required to perform multicast to arbitrary destination sets.

5.1 Multidestination Routing: Background

Multidestination message passing is a technique that has recently been proposed for routing messages from a single source to multiple destinations. Proposed originally in the context of parallel systems based on direct (router-based) networks [22, 33], this work has been extended recently to switch based networks [41, 48]. A multidestination worm covers multiple destinations in a switch-based network by replicating at the switches on its path. The copies of the multidestination worm formed by replication at a switch are forwarded to other switches or nodes depending on whether the output ports to which they are replicated are connected to switches or nodes. There are three main issues w.r.t. implementing multidestination worm based multicast in a switch based parallel system: multidestination header encoding/decoding, the replication mechanism adopted at the switches of the network, and the routing algorithm used for the multidestination messages.

5.1.1 Encoding and Decoding Multidestination Worm Headers

A multidestination worm header encodes information that helps the switches on the worm's path decide the set of output ports to which the message should be forwarded. This set of ports may be determined either *statically* at the source or *dynamically* by the switches on the worm's path. In the former case, the header encodes a route to the set of destinations that it covers. The switches on the worm's path use this route information (possibly after some decoding) to decide the set of output ports to which the multidestination message must be forwarded. Route table information is stored at the processing nodes: the switches in the network just require the ability to decode the header. In the latter case, the multidestination worm header typically encodes the set of destinations to which the worm is directed. Under this scheme the switches in the network must store enough routing information to decide the output ports that must receive a copy for the worm to reach all its destinations.

5.1.2 Replication Mechanism

Multidestination routing in switch based parallel systems requires an efficient replication mechanism at the switches [28]. In this paper, we assume that switches adopt the deadlock-free input-buffer-based asynchronous replication mechanism of [48]. Under such a replication mechanism, an arriving multidestination worm at an input buffer of a switch is read by multiple output ports. Furthermore, the various output ports proceed independently of each other so that blocked branches do not inhibit other non-blocked branches. A count value associated with packet units known as *chunks*, is initialized to the number of output ports to which the multidestination worm is to be forwarded. Every output port decrements the count value associated with a chunk as soon as it is read: a chunk is deleted once it has been read by all destination output ports.

5.1.3 Routing

Although many static encoding schemes encode the entire path of a multidestination worm, dynamic encoding schemes do not restrict the path of a multidestination worm in any way (except in requiring that the multidestination worm's destinations be eventually covered). The Base Routing Conformed Path (BRCP) model restricts multidestination worms to paths that are valid for unicast worms in the system [32]. By restricting multidestination worms to such paths deadlock problems can be avoided without making special rules for multidestination messages. A 'parent' of a particular branch (or copy) of a multidestination worm is the multidestination worm copy that replicates at a switch to give rise to the said branch. An 'ancestor' branch is similarly (and inductively) defined. In this paper we assume that multidestination worms conform to the UD routing in the following sense: no worm or branch (copy) of a worm takes an 'up' link after it or any one of its ancestors has taken a 'down' link.

5.2 Multi-phase Multicast using Multidestination Worms

The number of destinations that can be covered by a single multidestination worm depends on the scheme used for encoding multidestination messages. The scheme used for encoding multidestination worms in turn affects the cost and complexity of the decoding logic used at the switches. Encoding schemes that require very simple decoding logic may be unable to encompass arbitrary multicast destination sets into a single header. Conversely, some encoding schemes may allow arbitrary multicast destination sets to be captured in a single header, but may require more complex decoding logic at the switches.

In this section, we discuss two methods for multi-phase multicast that differ primarily in the complexity of finding multidestination worms to cover arbitrary multicast sets. Both schemes use headers that are simple to decode, but which are unable to capture arbitrary multicast destination sets in the same header, thereby requiring multiple multidestination worms for multicast to arbitrary destination sets. We present the header format for each scheme and the method used to derive the set of multidestination worms that are needed to cover any arbitrary multicast destination set. We then present two algorithms to divide the derived multidestination worms into multiple phases so that the source sends out a multidestination worm to a subset of the destinations in the first phase, and in each of the succeeding phases, the source and some of the destinations that have been covered act as secondary sources and send out multidestination worms to the remaining destinations.

5.2.1 Single Switch Replication (SSR) Multidestination Worms

A simple method for multicasting can use the replication mechanism described above at a *single* switch. In this method, called *single switch replication* (SSR), a multidestination worm covers a set of destinations that are all connected to the same switch. If the multicast destination set spans multiple switches, a multi-phase approach is used for multicast [41]. Under this approach multicast proceeds in multiple phases with the recipients of a multicast in any given phase acting as secondary sources in the next phase of the multicast. Figure 11(a) shows an example of a multidestination worm that replicates at a single switch.

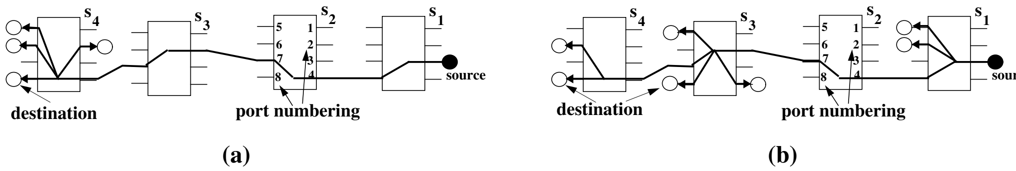

Figure 11: An example of (a) a multidestination worm using single switch replication, and (b) a path-based multidestination worm.

5.2.2 Path-based Multidestination Worms

Path-based multidestination worms cover destinations connected to any number of switches, provided that the switches lie on a valid unicast path from the source to a destination. In other words, a path-based multidestination worm uses the capability at switches to replicate and to forward copies to several output ports simultaneously, but the replication is restricted. A path-based multidestination worm is defined as follows.

Definition 3 *A worm is defined to be a path-based multidestination worm if a) it can replicate to multiple outgoing ports on a traversed switch, b) no more than one of these ports is connected to another switch, and c) the remaining outgoing ports are connected to processors (destinations).*

Such a multidestination worm is also called a *multi-drop worm* [19]. Path-based multidestination worms allow easy construction of paths (of switches) in an irregular network. Obviously it can potentially cover more destinations than an SSR multidestination worm. However, the complexity of identifying valid paths to create worms to cover an arbitrary multicast is higher in the case of path-based worms. This will be discussed further in Sec. 5.2.4. Figure 11(b) shows an example of a path-based multidestination worm. The route of a path-based multidestination worm is defined by the set of inter-switch links taken by the worm. From this point on, we refer to a path as a sequence of inter-switch links.

5.2.3 Encoding and Decoding Multidestination Headers

We now present a simple method for encoding and decoding the headers of the multidestination worms described above. Let us first consider SSR multidestination worms. Since replication is only allowed at a single switch, a simple method of encoding the multidestination header is to store information that the intermediate switches can use to route to the particular switch whose nodes are the destinations of the multidestination worm. In addition, the header must also encode the list of the switch's nodes that are the destinations of the multicast.

Figure 12(a) shows a generic representation of the header encoding for SSR multidestination worms. The first portion of the header consists of a node or switch ID along with a 1 bit tag to denote whether the message is a unicast or multidestination message. The choice of whether a node or a switch ID is used can be made depending on whether or not the system allows messages to be directed to switches (as allowed in Autonet [37]). If a node ID is used, this field may carry the ID of any one of the worm's destinations attached to the destination switch. The second field consists of a p bit string where p is the number of output ports in a switch: the ith bit in this string is set to '1' if output port i of the destination switch leads to a multicast destination. Figure 12(b) shows the header encoding for the sample SSR multidestination worm shown in Fig. 11(a). The switch ID is used in this example.

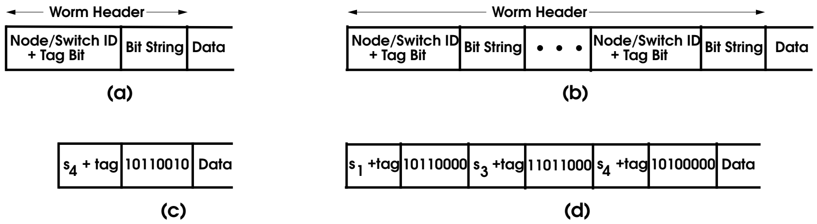

Figure 12: Header encodings: (a) Generic representation for SSR worms, (b) generic representation for path-based worms, (c) example encoding for SSR worm shown in Fig. 11(a), and (d) example encoding for path-based worm shown in Fig. 11(b).

The *multi-drop* header encoding scheme for path-based multidestination worms is an extension of the above proposed encoding scheme. Figure 12(c) shows a generic representation of the header encoding for a path-based worm. It consists of a sequence of node or switch ID (depending on whether or not the system allows messages to be directed to switches) and p bit string pairs. Each pair encodes a set of destinations connected to a switch which lies in the path of the worm. It should be noted that the encoding does not have to specify the complete route taken by the worm—intermediate switches with no destinations can be excluded. Figure 12(d) shows the header encoding for the sample path-based worm shown in Fig. 11(b). Like in the example shown in Fig. 12(b) the switch ID is used. Since switch s_1 is the first switch with destinations on it, the encoding begins with the switch ID s_1. The following 8 bits encode the information about which output ports the worm replicates to in switch s_1. Similarly, information about switches s_3 and s_4 (the switches with destinations connected to them) is encoded in the header.

For decoding the multidestination header, the information set up for routing unicast messages is sufficient. Let us consider decoding of the SSR header encoding. The decoding operation for the multi-drop header at a switch is identical to that for the SSR worm header, and we, therefore, restrict this discussion to the decoding of the SSR header. Let us assume the case where the first field of the multidestination header encodes a node ID. A switch on the worm's path examines the first field of the header and the tag bit. If the tag bit is off (a unicast packet), the worm looks up its routing table to decide the port to which the worm is to be forwarded. If the tag bit is on (a multidestination packet), the switch first examines if the destination is one of the nodes connected to it. If so, the second field is read to determine the output ports that should receive copies of the packet. Otherwise, the worm is forwarded to an appropriate output port as determined from the switch's routing tables just as if it were a unicast message. Since the routing table lookup and the determination of whether a destination is connected to a given switch can potentially be performed concurrently, the multidestination header causes no additional decoding overhead (in terms of latency) at the intermediate switches. In the case when the first field of the header carries a switch ID, decoding can become even simpler. An intermediate switch examines the switch ID field with its own ID and forwards the message to an appropriate output port in the case of a mismatch. If the switch ID field matches its ID, the tag field is used to determine whether the packet must be consumed by the switch or whether it is a multidestination packet intended for nodes connected to the switch. In the latter case, the second field of the header is examined and the worm is replicated accordingly to the ports which correspond to '1' bits in the string. Decoding of the multi-drop header encoding is an easy extension of the above. It must be noted that the header encoding used in all cases is *source independent*, i.e. the set of destinations covered by a multidestination worm depends only on the contents of the header and not on the source from which it originates.

It is obvious that a path-based worm can cover more destinations an SSR worm. This is because SSR worms are path-based worms which have no destinations in the intermediate switches. The header encoding and decoding complexity is quite similar in both kinds of worms because the header encoding for path-based worms is a simple extension of the encoding for SSR worms. Although path-based worms can cover more destinations than SSR worms, the complexity of finding worms to cover an arbitrary multicast destination set is much less with SSR worms. The following section discusses this aspect in more detail.

5.2.4 Finding the Multidestination Worms for Arbitrary Multicasts

An SSR multidestination worm can only cover a set of destinations connected to the same switch. Therefore, given a set of destinations connected to a switch it is a simple operation to construct an SSR worm from any possible source to cover the destination set. Also, to perform multicast to arbitrary destination sets, we need as many multidestination worms as the number of switches that have destination nodes. Deriving a set of headers (one for each multidestination worm) is a simple operation. Given a mapping between nodes, switches, and the ports at which they are attached, subsets of the destination set can be found that are all attached to the same switch. From each subset a header for a corresponding multidestination worm can be derived by choosing one of the nodes in the subset for the node ID field of the header. The second field in the header can be obtained by looking up the mapping between the destinations in each subset and the switch ports to which they are attached.

However, construction of path-based worms to cover a given a set of destinations is a much more difficult problem. Given an arbitrary destination set, it may not always be possible to find a path from the source (conforming to UD routing) to any one of the destinations, such that all the destinations are connected to the switches that lie on that path. In the remaining part of this section, we examine the problem of defining valid path-based worms to cover a given destination set.

Let us define the list of participating switches, $L_{sw}(w)$, of a path-based multidestination worm w as follows. $L_{sw}(w)$ consists of switches directly connected to the destinations of w, in the order in which the corresponding destinations will be covered. Let us denote the jth processor on the ith switch as $p_{i,j}$. If w starts from source processor $p_{s,1}$ and covers the processors $p_{1,1}, p_{1,2}, \ldots, p_{2,1}, p_{2,2}, \ldots, p_{n,m}$, in that order, then $L_{sw}(w)$ is given by the switches $< s_s, s_1, s_2, \ldots, s_n >$. Therefore, generating a valid path-based multidestination path for w on an arbitrary switch-based irregular network N is equivalent to generating a valid path which covers the switches in $L_{sw}(w)$ (in that order) on the interconnection graph G of that network.

Definition 4 *Let P_{sd} and P_{sd}^m be defined as the sets of all paths allowed between a source switch s to a destination switch d under the UD routing and UD minimal routing, respectively.*

Section 2.2 discussed the minimal UD routing support at the routing tables on each switch. We now try and construct valid UD paths for supporting multidestination worms. It should be noted that although the routing tables support only minimal UD, our construction might result in non-minimal UD paths. However, minimal UD and non-minimal UD paths can coexist without deadlock because UD routing is deadlock-free.

In a separate work [16], we defined the concept of a *Partial Ordered Chain* (POC), with the specific purpose of constructing an ordering among switches in an irregular network to reduce link contention between two messages of the same multicast using *unicast* message passing. In this paper, we use this concept to construct valid path-based multidestination paths.

Definition 5 *A partial ordered chain (POC) is any ordered list of switches $< s_1, s_2, \ldots, s_n >$, where s_i is connected to s_{i+1} by a "down" tree link (from the BFS spanning tree) or a "down" cross link connecting switches at different levels of the BFS spanning tree.*

Definition 6 *The path set of a path-based multidestination worm w, denoted as $PS^m(w)$ is defined as the set of all paths that can be taken by w when routed in a piece-wise manner using the routing tables which support UD minimal routing.*

Lemma 5 *The path set of a path-based multidestination worm w, $PS^m(w)$, is given by $P^m_{s_s s_1} \times P^m_{s_1 s_2} \times \ldots \times P^m_{s_{n-1} s_n}$ where $L_{sw}(w) =< s_s, s_1, s_2, \ldots, s_n >$.*

Proof: The worm w starts from the switch s_s and covers $s_1, s_2, \ldots, s_n$, in that order. Since w is routed in a piece-wise manner using UD minimal routing, it first takes one of the paths from $P^m_{s_s s_1}$. Then, it takes one of the paths from $P^m_{s_1 s_2}$, and so on. Thus, $PS^m(w) = P^m_{s_s s_1} \times P^m_{s_1 s_2} \times \ldots \times P^m_{s_{n-1} s_n}$. ■

We now show that if some conditions hold, each element of the set $PS^m(w)$ also belongs to the set of all non-minimal UD paths from s_s to s_n. In that case, any path taken by piece-wise minimal UD routing of w is a UD path (possibly non-minimal), and therefore will not result in deadlock.

Theorem 4 *If $L_{sw}(w) =< s_s, s_1, s_2, \ldots, s_n >$ and if the switches $s_1, s_2, \ldots, s_n$ lie on some POC, P, in that particular order, then $PS^m(w) \subseteq P_{s_s s_n}$.*

Proof: It can be easily observed that all paths in $P^m_{s_s s_1}$ are minimal UD paths. If it can be shown that $\forall i, (1 \leq i < n)$, all paths in $P^m_{s_i s_{i+1}}$ are *down** paths, then using the result from Lemma 5 it can be seen that all paths in $PS^m(w)$ are UD paths. Let P be given by the ordered list of switches $< sw_1, sw_2, \ldots, sw_m >$. Then $\forall i, (1 < i < n)$, $s_i \in P$. Let us use the symbol $<_{poc}$ to denote the order in the above list P. Therefore, $\forall i, (1 \leq i < m)$, $sw_i <_{poc} sw_{i+1}$, and $\forall i, (1 \leq i < n)$, $s_i <_{poc} s_{i+1}$. Let $E =< e_1, e_2, \ldots, e_{n-1} >$ denote the list of "down" links such that switch sw_i is connected to the switch sw_{i+1} by the "down" link e_i, where sw_i and $sw_{i+1} \in P$. Let us consider two arbitrary switches s_a and s_{a+1} such that $1 \leq a < n$, $s_a = sw_i$, and $s_{a+1} = sw_j$. Therefore, $sw_i <_{poc} sw_j$. It can be easily observed that the path described by list of links $p_{i,j} =< e_i, e_{i+1}, \ldots, e_{j-1} >$ is one of the valid minimal paths between sw_i and sw_j. Also, each link in this path is a down link, and the path is $j - i$ links long. Any path from sw_i to sw_j that contains an up link is definitely longer than $p_{i,j}$ because an up link takes the worm higher up the BFS routing tree, whereas sw_j is exactly $j - i$ levels lower than sw_i in the BFS tree. Since a routed worm from s_a to s_{a+1} takes only down links, all paths in $P^m_{s_a s_{a+1}}$ are *down** paths. Therefore, piece-wise routing from s_1 to s_n results in a *down** path only. Therefore, all paths in $PS^m(w)$ are UD paths. ∎

Theorem 5 *A path-based multidestination worm w, where $L_{sw}(w) =< s_s, s_1, s_2, \ldots, s_n >$ and the switches $s_1, s_2, \ldots, s_n$ lie on some POC, P, in that particular order, always follows a valid UD path and introduces no deadlocks.*

Proof: This can be easily derived from Theorem 4, and the fact that non-minimal UD routing is deadlock-free, and non-minimal and minimal UD paths can coexist without deadlock. ∎

Given the above background, we can see that finding a set of POCs that cover an arbitrary multicast destination set is equivalent to finding a set of path-based multidestination worms that cover the same multicast set. Furthermore, instead of just finding a set of path-based multidestination worms that cover a given destination set, we would like to find a minimal set of the path-based multidestination worms that cover the multicast set. To do this, we adopt a greedy heuristic. We first try to find POCs that have maximal number of destinations (called the *longest POCs*). After eliminating the destinations covered by these POCs, we then repeat the procedure of trying to find POCs with maximal number of destinations among the remaining nodes. This procedure is repeated till a set of POCs is obtained that cover the given destination set. The path-based multidestination worms corresponding to these POCs then represent a near-optimal number of worms for covering the given destination set.

A formal specification of the algorithm to find POCs is given in Fig. 13 as a five-step approach. In the first step, a depth-first-search (DFS) is applied on the BFS spanning tree discussed in Section 2.2, starting with the root node and considering only the "down" links. This is to facilitate the construction of the longest POCs. The step results in a DAG, T. Figure 14(a) shows the BFS graph of a sample irregular network. Let us assume there are two processors connected to each switch of the irregular graph, and let the processors on the switch numbered i be labelled as $p_{i,1}$ and $p_{i,2}$. Let us consider a multicast with the source as $p_{10,1}$ and two destinations on each switch except switches 10 and 6. Figure 14(b) shows the DAG, T, which is created when the above DFS is applied on the BFS tree in Fig. 14(a). A participating switch is defined as one with at least one participating processor connected to it. In the third step, the resultant DAG, T, from the DFS is reduced to a DAG, T', which contains only the participating switches. Figure 14(c) shows the T' created when the T from Fig. 14(b) is reduced.

In order to determine the longest POCs and form an set of such POCs, we carry out a weighted descendents approach. As indicated in step 4, each switch is given an appropriate weight according to the number of participating processors connected to it and to all its descendent switches. Figure 14(c) shows the corresponding weights. The child with the largest weight indicates how to proceed while building the longest

Finding a set of POCs

Inputs: G: irregular graph, B: a breadth first spanning tree on the switches of G, D: set of destinations, n_s: the source; $c(n, s)$: node n is connected to switch s.

Output: A set of POCs, L.

Procedure:

1. Apply the DFS on B from the root node r of B. Only *down* tree links belonging to B, and *down* cross links which connect switches in different levels of B are used. Let the resulting DAG be T.
2. $S \leftarrow \{s \mid s$ is a switch in G, and $\exists d, d \in D \cup \{n_s\}$ and $c(d, s)\}$. S is the set of participating switches.
3. Reduce T to a DAG T' which contains all switches in S and no other switches. Reduction is done by removing all non-participating switches from T. Every time a switch s and all links incident on s are removed, directed *down* links are drawn from each parent of s to each child of s.
4. Calculate the weight wt of each switch in T' as follows:
   $wt(s) \leftarrow \sum_{s_i \in Desc(s)} \rho(s_i)$ where $Desc(s)$ is the set containing s and all the descendents of s in T', and $\rho(s)$ is the number of participating processors connected to the switch s.
5. $L \leftarrow null$; $i \leftarrow 0$
   while T' is not empty
   (a) Let p_1 be a switch with maximum weight in T'.
   $l_i \leftarrow < p_1, p_2, \ldots >$ such that p_{j+1} is the switch with the maximum weight among all children switches of p_j.
   (b) $T' \leftarrow T' - \{p \mid p \in l_i\}$
   (c) $l'_i \leftarrow < n_{1,1}, n_{1,2}, \ldots, n_{2,1}, n_{2,2}, \ldots >$ where $\{n_{j,1}, n_{j,2}, \ldots\}$ are the participating processors connected to switch p_j in l_i.
   (d) $L \leftarrow L \cup \{l'_i\}$; $i \leftarrow i + 1$
   end while

Figure 13: Finding a set of POCs.

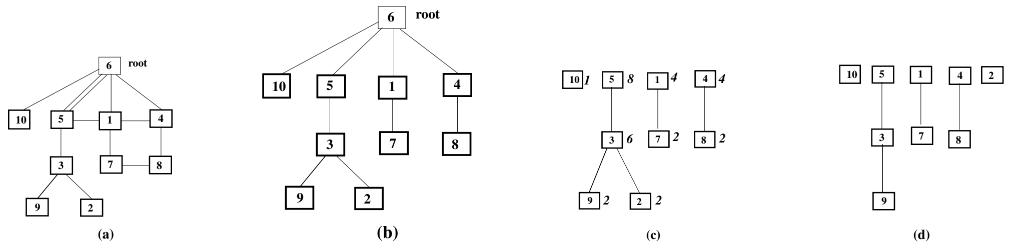

Figure 14: Illustrating steps for construction of POCs: (a) sample BFS graph of an irregular network, (b) DFS T created by step 1, (c) DAG T' created according to step 3 and the weights for switches computed according to step 4, and (d) the set L of chains.

POC from the parent. After the weights have been calculated, chains of switches are stripped off from T' according to their weights in step 5. In other words, the heaviest chain gets stripped first from T', and the lightest the last. The chains of switches stripped off from the T' are shown in Fig. 14(d) in chronological order from left to right (except the chain with switch 10).

5.2.5 Algorithms for Performing Multi-Phase Multicast

In this section we propose two multicast algorithms using the multidestination worms that were derived in the previous sections to cover given (arbitrary) multicast destination sets. Under the Multi-phase multicast with SSR worms (MPM-SSR) scheme, each of these multidestination worms cover only destinations that are all connected to the same switch. On the other hand, while using path-based multidestination worms, each worm can cover a destinations connected to switches along a valid (unicast) path from the source that conforms to the base routing scheme. The algorithms presented differ in the number of phases required to perform the multicast and in the amount of contention that is caused among the constituent multidestination worms of the multicast.

The Greedy (G) Algorithm

This algorithm uses a greedy approach on the set of multidestination worms that were derived in the above subsections. The first step of the algorithm is to find a set of multidestination worms to cover the given set of multicast destinations and to order these worms in decreasing order of the number of destinations covered (this set is obtained as a result of the previous subsections). In the next step of the algorithm, the source sends a multidestination worm that covers the largest subset of the destinations. In every succeeding step, the source and destinations covered up to the beginning of that step act as secondary sources and send out multidestination worms from the ordered list to some of the remaining destinations. This process continues till all the multidestination worms in the set have been sent out i.e. until the multicast is complete. Figure 15(a) shows these steps using SSR multidestination worms for the example of Fig. 14(a). In the first step, an SSR worm from the source on switch 10 covers both destinations on switch 5. In the second step, the source and each destination on switch 5 cover all destinations on switches 3, 9, and 1, respectively, and so on. The multicast completes in 3 steps. Figure 15(b) shows the steps for the same example multicast using path-based multidestination worms: In the first step, the source on switch 10 covers all the destinations on switches 5, 3, and 9 with a single path-based multidestination worm. In the second step, the source covers all destinations on switches 1 and 7 using a multidestination worm, one destination on switch 5 covers all destinations on switches 4 and 8, and the other destination on switch 5 covers all destinations on switch 2. Therefore, the multicast completes in two steps.

Since at each step of the algorithm, the maximum possible destinations are being covered (under each of the multi-phase schemes), this algorithm takes minimum number of steps to execute. However, it has the drawback that the number of multidestination worms starting from a single switch might be large, leading to link contention.

The Less Greedy (LG) Algorithm

We propose the LG algorithm to address the above drawback. Like the Greedy algorithm, this algorithm also finds a set of multidestination worms to cover the given destination set in the first step, and sends out a multidestination worm that covers the largest subset of the given multicast destination set in the next

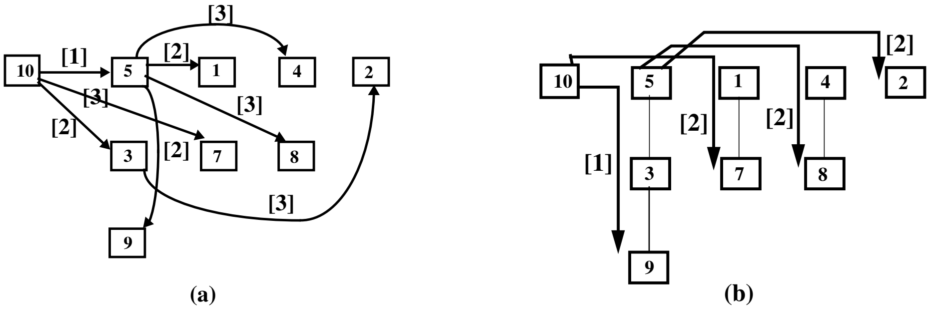

Figure 15: Illustrating the Greedy multicast algorithm for: a) the MPM-SSR scheme and b) the Multi-drop Path-based scheme. The numbers in brackets denote step numbers.

step. However, following this step, only one destination from each switch that has been covered sends out multidestination worms from the set of remaining multidestination worms. Since all destinations covered in the particular step do not send out worms, this scheme is given the name Less Greedy.

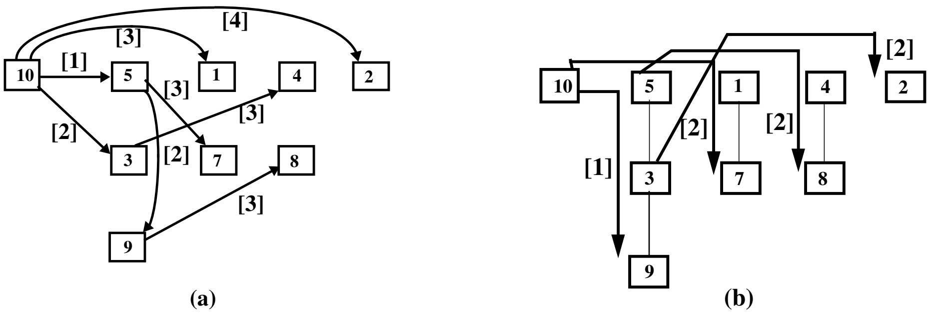

Figure 16: Illustrating the Less Greedy multicast algorithm for: a) the MPM-SSR scheme and b) the Multi-drop Path-based scheme. The numbers in brackets denote step numbers.

Figure 16(a) shows these steps using SSR multidestination worms for the example of Fig. 14(a). In the first step, an SSR worm from the source on switch 10 covers both destinations on switch 5. In the second step, the source and one destination on switch 5 cover all destinations on switches 3 and 9, respectively. In the third step, the source covers both destinations on switch 1, one destination on switch 5 covers both destinations on switch 7, and so on. The multicast completes in 4 steps. Figure 16(b) shows the steps for the same example multicast using path-based multidestination worms: In the first step, the source on switch 10 covers all the destinations on switches 5, 3, and 9 with a single path-based multidestination worm. In the second step, the source covers all destinations on switches 1 and 7 using a multidestination worm, one destination on switch 5 covers all destinations on switches 4 and 8, and one destination on switch 3 covers all destinations on switch 2. This algorithm may take more steps to complete, but it reduces link contention by making sure that only one multidestination worm starts from a switch during one step.

5.2.6 Comparison of Multi-phase Multicasting Schemes

We now compare the two multi-phase multicasting schemes that we have proposed in the above subsection. It is fairly obvious that the each of the multidestination worms under the MPM-SSR scheme is also a valid path-based multidestination worm, since it replicates at only one switch and at that switch it replicates to one or no other switch (actually it only replicates to those ports that lead to other destinations). However, each of the path-based multidestination worms can cover at least the same number of destinations as a multidestination worm under the MPM-SSR scheme. Thus the number of worms required for multicast using path-based multidestination worms is always less than or equal to the number of worms required under the MPM-SSR scheme. Thus, multicasting using path-based multidestination worms will always perform as well or better than the MPM-SSR scheme.

However, the MPM-SSR scheme has some advantages over the path-based scheme in that deriving the set of multidestination worms for multicast is a simpler operation under the MPM-SSR scheme. Furthermore, the node need not really be equipped with the complete topology of the network: it suffices for the nodes to know the mapping from the nodes to the switch IDs and the port numbers of the switches to which they are connected. However, as described above, the path-based scheme will almost always outperform the MPM-SSR scheme. We therefore do not consider the MPM-SSR scheme for the rest of this paper.

Under the path-based multicasting scheme, the actual multicast latency may differ depending on the algorithm used for multicast and the multicast destination set given as input. Our previous study [19] has shown that the Less Greedy algorithm performs better than the Greedy algorithm on the average because it reduces the contention for network links among the constituent multidestination worms of the multicast. We therefore only consider the path-based multicasting using the Less Greedy algorithm as a representative of the multi-phase multicasting schemes using multidestination worms in the rest of this paper.

5.3 One-phase Multicast using Tree-based Multidestination Worms

We have presented methods for multicasting using multiple phases of multidestination worms in the previous section. We now propose a method to perform multicast with a single multidestination worm using a method of multidestination header encoding called *bit string encoding*.

Multicast using bit-string encoded multidestination worms has been proposed in the context of regular networks [5, 34, 48]. An efficient, single phase multicasting scheme using such a multidestination worm with a bit-string encoded header has been proposed in [34, 48]. Figure 17(a) shows a sample tree-based multidestination worm which completes the sample multicast shown in Fig. 14(a) in a single phase. Key to the efficient implementation of this multicast is the presence of reachability strings at each of the network switches. Although such reachability strings are fairly easy to determine in regular networks, determining such reachability strings in irregular networks is harder. Furthermore, multiple paths exist to a given destination from a given switch, a property not present in many regular networks such as many of the unidirectional and bidirectional MINs. One important contribution of this paper is to outline a method for setting up these reachability strings as an enhancement of the setup procedure used for the UD routing algorithm in irregular networks. Another contribution is the presentation of simple header modifying procedure that prevents multiple copies of a worm from being forwarded to a destination along different paths. Before examining these however, we briefly review the strategy adopted for encoding such multidestination worms.

5.3.1 Encoding Multidestination Headers

Bit string encoding is an example of a dynamic encoding scheme. In this form of encoding the header carries an N-bit string where N is the number of nodes in the system. The ith bit of this string is set to 1 if node i is a multicast destination; otherwise the bit is set to 0. In addition, a bit (called the *up-down bit*) may be added to the header to indicate whether the worm is currently traveling along the 'up' or 'down' direction. This bit is initially set to '1' to indicate that the worm is traveling in the 'up' direction: it is reset when the

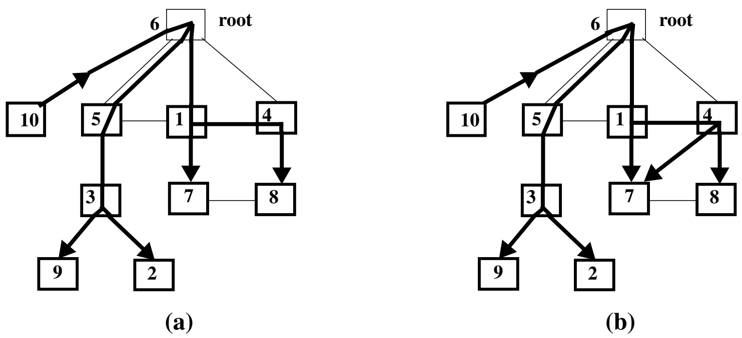

Figure 17: (a) A tree-based multidestination worm which completes the sample multicast of Fig. 14(a) in a single phase. (b) A sample existence of multiple paths to a destination on switch 7.

worm begins traveling 'down'-wards.

To decode such a header efficiently, the switches of the system must possess information about the nodes that are reachable from each of their output ports. This information can be efficiently captured using a bit string similar to the one described above. Bits that are '1' in this bit string represent nodes that are reachable through a particular 'down' output port. In addition, every switch maintains a bit string that captures the set of nodes reachable through *all* of its 'down' output ports. Given these *reachability strings* [34], decoding a bit string header reduces to a simple logical operation. Before examining this logic however, we first examine the procedure adopted for setting up these reachability strings.

5.3.2 Setting up Reachability Information

The setup procedure used for the UD routing algorithm assigns a direction as one of 'up' or 'down' to every link in the system. From the root switch, the 'down' links along with the other switches form a Directed Acyclic Graph (DAG). This implies that more than one of the switch's 'down' links may lead to a given set of destinations. Furthermore, two different switches may lead to the same set of destinations (following 'down' links alone). If reachability strings captured all the destinations reachable through a given switch's ('down'-ward) output ports, we may end up forwarding a multidestination worm to more than one output port to cover the same set of destinations—this set of destinations will receive multiple copies of the message. To prevent this we can restrict the reachability strings at a switch so that only one 'down' port leads to a given destination. We call these reachability strings *restricted reachability strings.*

However, even this does not prevent multiple copies of the worm from reaching some of the destinations. For example, consider a variation of Fig. 17(a) with a bidirectional link between switches 4 and 7. Two of switch 6's 'down' ports lead to switch 7, one through switch 1 and the other through switch 4. Even if the reachability strings at switch 6 were made mutually exclusive, a multidestination worm intended for destinations in switches 4 and 7 will result in two copies of the message being forwarded to switch 7. This is because that switch is reachable through the 'down' links of switches 1 and 4. Switch 1 forwards copies of the worm[1] along its 'down' ports to switches 4 and 7. However switch 4 will end up forwarding the arriving

[1]Note that switch 4's nodes will be included in the reachability strings of two of switch 6's ports: the port that leads to switch 1 and the port that leads directly to switch 4. However, the reachability string at switch 1 also includes the nodes of switch 4 (and those of switch 2) since these are reachable through its down links. We are assuming here that when the restricted reachability strings are calculated, the nodes of switch 4 (and switch 2) will be included in the restricted reachability string of the port that leads to switch 1 from switch 6. The port that leads to switch 4 will have an all 0 restricted reachability string in this case.

worm to switch 7 because: (i) switch 7 is reachable through its down links and (ii) the header of the worm still encodes the destinations in switches 4 and 7. This situation is shown in Fig. 17(b). There are two methods by which such forwarding can be prevented. The first method is to restrict multicasts so that they are adhere to a strict (common) BFS tree, i.e. to ignore cross-links. In a strict BFS tree the link between switches 4 and 7 would not be usable, preventing the situation described above.

A second alternative is to modify the headers of multidestination worms so that the worm copies carry only the subset of the multicast destinations that are reachable through the restricted reachability strings. In this case, only one of switch 1's ports will have a restricted reachability string that includes the nodes of switch 4. Depending on how the restricted reachability strings are set up, this string may or may not include the nodes of switch 7. If it does include the ports of switch 7, no other ports of switch 1 will have restricted reachability strings that include nodes of switch 7. Therefore, one copy of the worm will be forwarded to switch 4, which will in turn forward a copy to switch 7. If, on the other hand, the port whose restricted reachability string includes the nodes of switch 4 *does not* include the nodes of switch 7, then some other port of switch 1 will have these nodes in its restricted reachability string. Switch 1 will therefore forward two copies of the worm to switches 4 and 7 respectively. However, because of the header modification, the copy that is forwarded to switch 4 will not include the destination of switch 7 in its header, preventing a copy from being forwarded for switch 4 to switch 7.

If we decide to modify multidestination worm headers for each of the replicated copies, additional logic is required to perform this header modification. However the method has the important advantage that multicast need not be restricted to one common tree. We therefore choose to modify multidestination headers. It is to be noted that with header modification, we can even relax the requirement for restricted reachability strings (i.e. the original reachability strings can be used instead). However the header modification logic becomes considerably more complex without the restricted reachability strings. We therefore use such restricted reachability strings for the scheme discussed in this paper. A procedure to set up the restricted reachability strings at the switches given the network topology (found as a result of the setup algorithm for UD routing) is shown in Fig. 18. The notation used in this procedure is as follows: B_l is the set of switches at level l of the BFS tree B, r_i is the total reachability string of switch sw_i, and r_{ij} is the restricted reachability string of port p_j of switch sw_i.

5.3.3 Decoding Multidestination Headers

For the purposes of this paper, the decoding method adopted is as follows. If an arriving multidestination worm has its up-down bit set to '1', the worm's header is compared with the total reachability string associated with the switch (a bit-wise AND is performed of the two strings). If all the worm's destinations are reachable from the switch's 'down' output ports (the fore-mentioned bit-wise AND operation results in a string that is identical to the bit-string header), the up-down bit is set to '0' and the header is compared with the (restricted) reachability strings of each of the switch's output ports that lead to 'down' links (another bit-wise AND). If the output port leads to any of the worm's destinations (the result of the bit-wise AND is non-zero), the worm is replicated to the specified output port. Before being forwarded to the specified output port, the worm header has to be modified. If a multidestination worm arrives at a switch with its up-down bit set to '0', the worm's header is directly compared with each of the 'down'-ward ports' restricted reachability strings.

If not all the multidestination worm's destinations are reachable from the switch's output ports that lead to 'down' links, the worm is routed to any one of the switch's output ports that lead to 'up' links. The

Calculating Reachability Information

Inputs: B: the BFS tree derived for the UD routing algorithm
Output: Set of restricted reachability strings for each switch
Procedure:

1. **for** $l = maxlevel$ downto 0 /* root switch is at level 0 */
   foreach switch $sw_i \in B_l$
   for $j = 1$ to num_ports
   if port p_j connects to node n_k via a *down* link
   set kth bit of r_{ij} and r_i
   elseif port p_j connects to switch sw_k via a *down* link
   r_{ij} = bit-or(r_{ij}, r_k); r_i = bit-or(r_i, r_k);

/* *for each switch, make sure only one of the down ports leads to a given destination* */

2. **foreach** switch $sw_i \in B$
   for $j = 1$ to $num\_ports - 1$
   for $k = j + 1$ to num_ports
   if ports p_j and p_k outbound to *down* links
   r_{ij} = bit-and(r_{ij}, bit-not(r_{ik}))

Figure 18: The setup algorithm to find the restricted reachability strings and the total reachability string for each switch in the network.

choice may be made adaptively depending on local traffic conditions or may be made arbitrarily (for eg., round robin among the output ports that lead to 'up' links). The worm should eventually arrive at a switch from which all its destinations are reachable. Note that this is a property that exists because of the UD routing algorithm and the method followed for assigning 'up' and 'down' directions to the links (the set-up procedure). A sample implementation that performs this function has been presented in [42]. We assume such an implementation for the rest of this paper.

6 Comparative Evaluation

Having described the three different efficient multicasting schemes that we are comparing in this paper, we now compare their performance qualitatively and by using detailed simulation study.

6.1 Qualitative Evaluation

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 until 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.

The relative performance of the three multicasting schemes 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.

6.2 Experiments and Performance Measures

We used a C++/CSIM based simulation test-bed [31] for our experiments. The simulation test-bed 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. Keeping in mind the total amount of buffer space in current-day switches such as the IBM SP, and accounting for an increase in this amount with advances in technology, we choose an input buffer size of 640 flits for the simulated switches. 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 266 MB/s. The current PCI Bus standard calls for a bandwidth of 133 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 [29]). 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 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. We assume a uniform routing overhead of 1 cycle for all three schemes. 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 [18, 17]. 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 [48], 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.

6.3 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 $(sk - p)$ switch ports so that the graph with the switches as vertices remains connected. We assume that all ports of a switch are full duplex 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%.

6.4 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.

6.4.1 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 19 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 $t_{hs} + t_{ns}$.

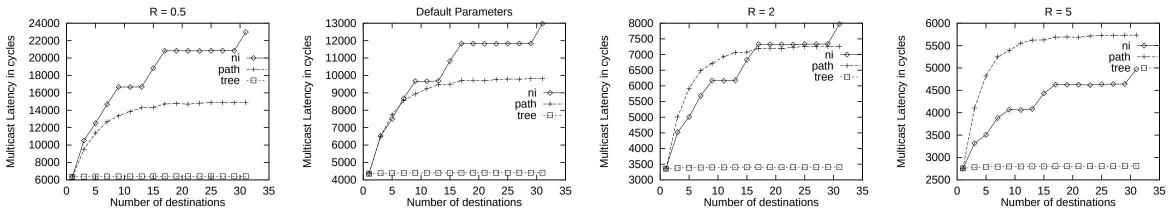

Figure 19: Effect of R on single multicast performance.

6.4.2 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. 20, 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.

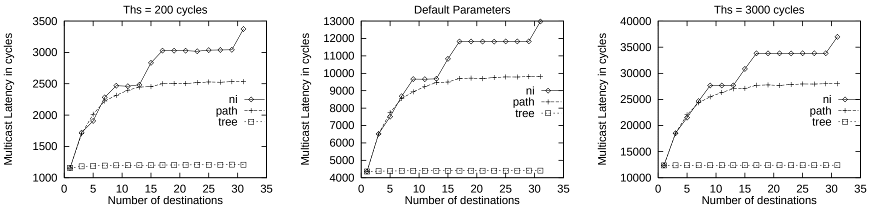

Figure 20: Effect of Software Overhead at Host Processor.

6.4.3 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 21 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.

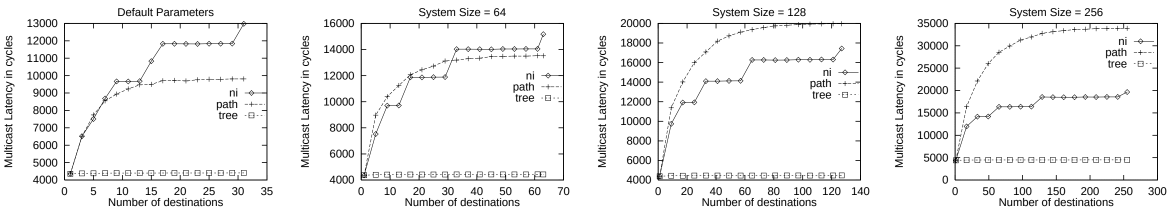

Figure 21: Effect of System Size.

6.4.4 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. 22. 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.

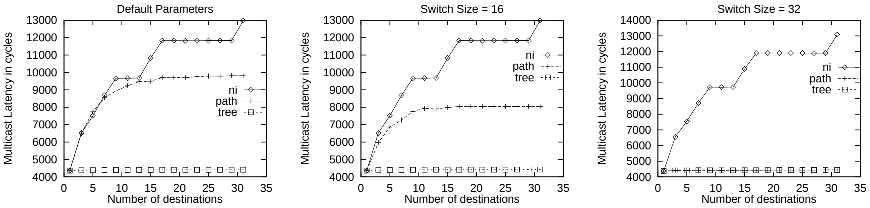

Figure 22: Effect of Switch Size.

6.4.5 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. 23 corroborate the results of Sec. 6.4.3, 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.

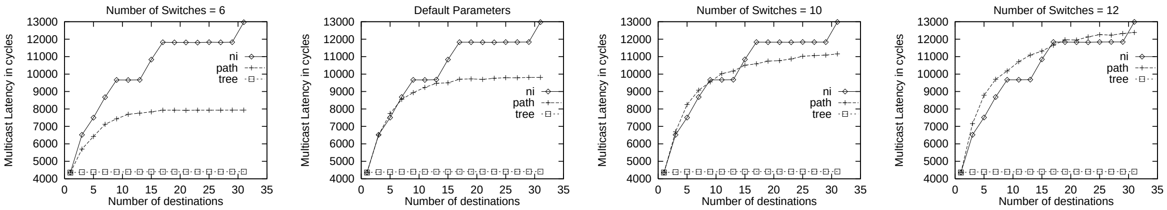

Figure 23: Effect of Number of Switches

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.

6.4.6 Effect of Message Length

Figure 24 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.

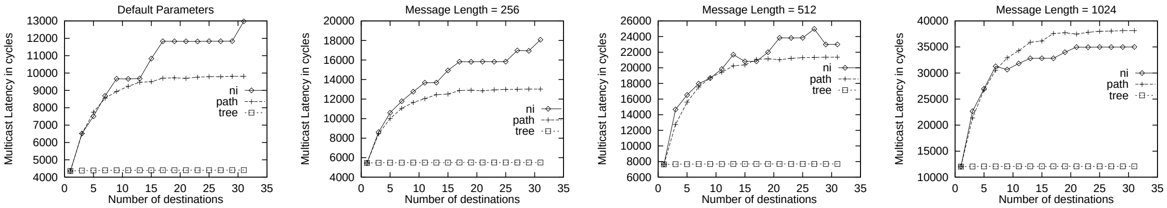

Figure 24: Effect of Message Length

6.4.7 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 25 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.

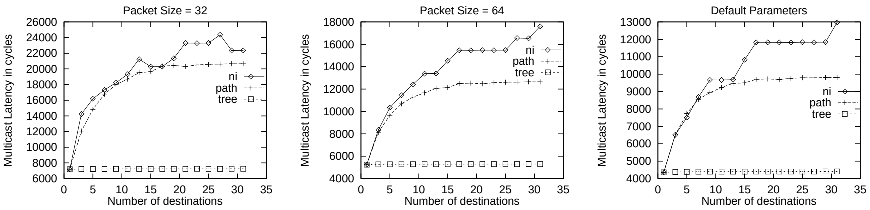

Figure 25: Effect of Packet Size

6.5 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 noting 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. 6.2.

6.5.1 Effect of R

Figure 26 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.

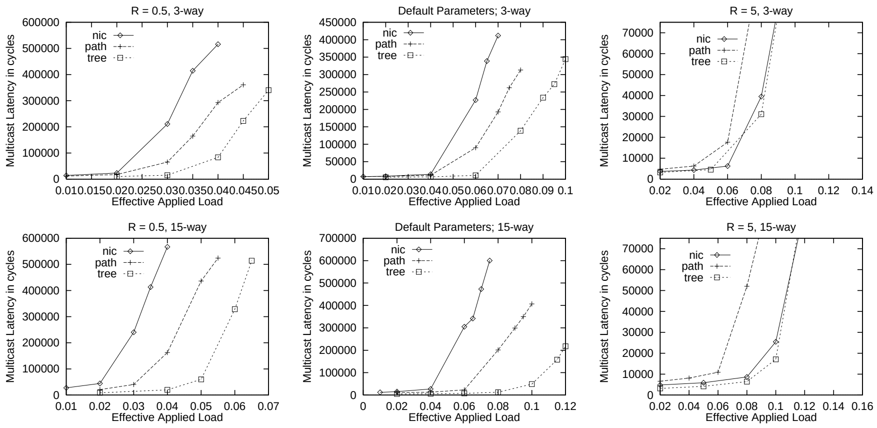

Figure 26: The effect of parameter R on the latency of multicasts under increasing multicast load for 3-way and 15-way multicasts.

6.5.2 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. 27). 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.

6.5.3 Effect of Message Length

Our results for multicast performance under increasing message length are shown in Fig. 28. 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

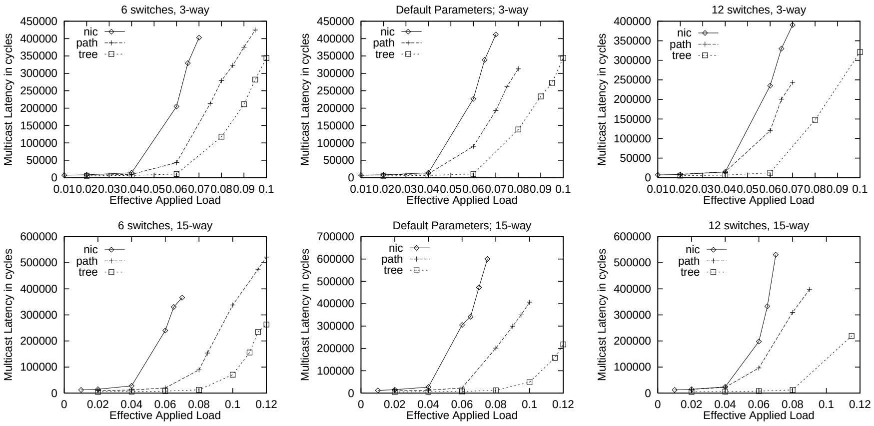

Figure 27: The effect of the number of switches on the latency of multicasts under increasing multicast load for 3-way and 15-way multicasts.

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.

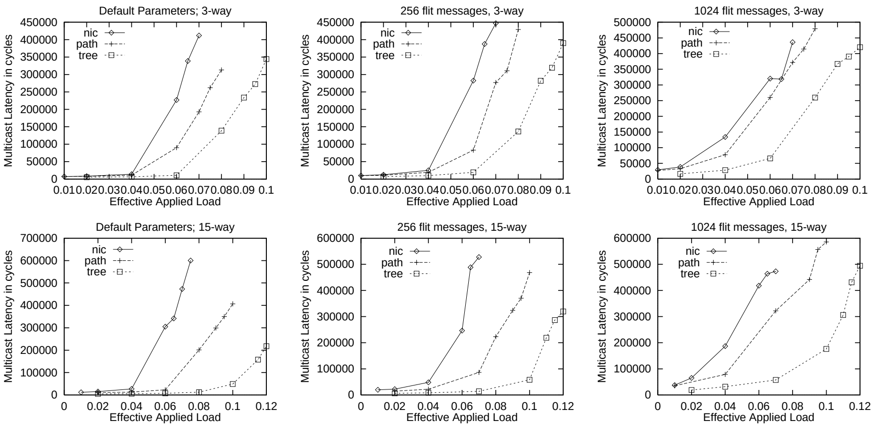

Figure 28: The effect of the message length on the latency of multicasts under increasing multicast load for 3-way and 15-way multicasts.

7 Conclusion

In this paper, we have proposed and compared various schemes for performing multicast efficiently in switch-based irregular networks. After describing the network and routing models, we discussed in detail multicasting schemes from two categories of schemes: NI-based and switch-based. We then selected the optimal NI-based scheme, the best multi-phase switch-based scheme, and the single-phase switch-based scheme and compared them qualitatively and using simulation experiments.

We find that the single-phase switch-based multicasting scheme performs better than the multi-phase switch-based and NI-based schemes. The relative performance of the multi-phase 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 multi-phase 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 multi-phase 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] R. A. F. Bhoedjang, T. Ruhl, and H. E. Bal. Efficient Multicast on Myrinet Using Link-Level Flow Control. In *Proceedings of the 27th International Conference on Parallel Processing (ICPP '98)*, pages 381–390, August 1998.

[3] 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.

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

[5] C. M. Chiang and L. M. Ni. Multi-Address Encoding for Multicast. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 146–160, May 1994.

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

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

[8] 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.

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

[10] 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.

[11] D. Garcia and W. Watson. Servernet II. In *Proceedings of the 2nd Parallel Computer Routing and Communication Workshop*, pages 119–135, June 1997. Available as `Lecture Notes in Computer Science #1417, Springer Verlag`.

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

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

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

[15] R. Kesavan. Communication Mechanisms and Algorithms for Supporting Scalable Collective Communication on Parallel Systems. PhD Thesis, The Ohio State University, October 1998.

[16] 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.

[17] R. Kesavan and D. K. Panda. Multiple Multicast with Minimized Node Contention on Wormhole k-ary n-cube Networks. *IEEE Transactions on Parallel and Distributed Systems.* accepted for publication.

[18] 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.

[19] 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), Lecture Notes in Computer Science # 1417*, pages 217–230, June 1997.

[20] 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.

[21] R. Libeskind-Hadas, D. Mazzoni, and R. Rajagopalan. Optimal Contention-Free Unicast-Based Multicasting in Switch-Based Networks of Workstations. In *Proceedings of the Merged 12th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing*, pages 358–364, April 1998.

[22] 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.

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

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

[25] 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.

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

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

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

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

[30] 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.

[31] 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.

[32] 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 OSU-CISRC-12/95-TR54, The Ohio State University, December 1995. *IEEE Transactions on Parallel and Distributed Systems*. In Press.

[33] 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`.

[34] D. K. Panda and R. Sivaram. Fast Broadcast and Multicast in Wormhole Multistage Networks with Multidestination Worms. Technical Report OSU-CISRC-4/95-TR21, Dept. of Computer and Information Science, The Ohio State University, April 1995.

[35] 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.

[36] 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.

[37] 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.

[38] 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`.

[39] R. Sivaram. Architectural Support for Efficient Communication in Scalable Parallel Systems. PhD Thesis, The Ohio State University, August 1998.

[40] R. Sivaram, R. Kesavan, D. K. Panda, and C. B. Stunkel. Where to Provide Support for Efficient Multicasting in Irregular Networks: Network Interface or Switch? In *Proceedings of the 27th International Conference on Parallel Processing (ICPP '98)*, pages 452–459, August 1998.

[41] R. Sivaram, D. K. Panda, and C. B. Stunkel. Efficient Broadcast and Multicast on Multistage Interconnection Networks using Multiport Encoding. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, pages 36–45, Oct 1996.

[42] 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), Lecture Notes in Computer Science # 1417*, pages 39–52, June 1997.

[43] 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*, 9(10), October 1998.

[44] 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.

[45] 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*, pages 134–143, April 1998.

[46] 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.

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

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.