Multicasting in Irregular Networks with Cut-Through Switches using Tree-Based Multidestination Worms*

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

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

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

Contact Author: Dhabaleswar K. Panda

Abstract: Multidestination message passing has been proposed as a mechanism to achieve efficient multicast in regular direct and indirect networks. The application of this technique to parallel systems based on irregular networks has, however, not been studied. In this paper we propose two schemes for performing multicast using multidestination worms on irregular networks and examine the extent to which multidestination message passing can improve multicast performance. For each of the schemes we propose solutions for the associated problems such as, methods for encoding and decoding multidestination headers, alterations to the setup algorithm run by the switches, logic to perform header manipulation, etc. We perform extensive simulations to evaluate our schemes under a variety of changing parameters: software startup overhead per message, system size, switch size, message length and degree of connectivity. Our results establish that even a very naive multicasting algorithm using multidestination message passing can result in considerable performance gains over the best unicast based multicasting schemes.

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

*This research is supported in part by NSF Grant MIP-9309627, NSF Career Award MIP-9502294 and an IBM Cooperative Fellowship.

Contents

1 Introduction

Parallel computing on networks of workstations is fast emerging as a cost-effective alternative to MPPs. These systems are typically interconnected by switches in an irregular topology. Because of this irregular topology, many of the problems that have been solved for regular topologies have to be solved anew for these systems.

To perform communication with low latency these systems typically use cut-through switching (routing) due to its inherent advantages like low latency and reduced communication hardware overhead. A lot of the recent research has focussed on the problem of deadlock free cut-through routing in irregular networks of workstations[1, 10, 11].

It has been established that efficient multicast is crucial to the operation of a parallel system. This is because multicast, in addition to being used in other collective operations, is also used in system level operations such as for cache-invalidations in a Distributed Shared Memory system [4]. Both systems using the Distributed Memory, and systems using the Distributed Shared Memory programming paradigms, make use of multicast/broadcast for various application and system level data distribution functions.

A lot of research has therefore focussed on methods for efficient multicast in parallel systems. These methods have predominantly been proposed for regular networks and fall into two broad categories: unicast-based and multidestination-based multicast. Unicast based multicast typically follows a multi-phase binomial tree based approach [16]. A unicast message is sent by the source to a destination in the first phase of this multicast. In every successive phase, the source and the multicast destinations covered up to the beginning of that phase send one message each to one of the remaining destinations. Such a multicast requires at least $\log_2\lceil(d+1)\rceil$ phases to cover d destinations. Since each of the phases already involves the high overhead of a software communication startup, the various software multicasting schemes focus on reducing or eliminating contention among the worms in a phase of the algorithm. Such unicast based multicasting schemes have also been recently proposed for irregular networks [5].

Multidestination routing has been proposed for parallel systems based on regular networks. Originally proposed for direct (router-based) networks [2, 6, 8], this scheme has been recently extended to regular switch-based parallel systems [13, 15]. However, the problem of how multidestination routing can be extended to irregular networks has not yet been addressed. Similarly, the extent to which such a multidestination message passing based multicast will benefit multicast performance in an irregular network has also not been evaluated.

In this paper we study the problem of performing multicast in irregular networks with cut-through routing using multidestination worms. We propose two schemes that are at the opposite ends of the spectrum of multidestination worm based multicasting. The first scheme, called the MPM-SSR scheme, uses a predominantly software based multi-phase approach to perform multicast to arbitrary destination sets along with the ability of a switch to replicate worms to multiple output ports. The second multicasting scheme, called the OPM scheme, carries out multicast in a single phase and is predominantly hardware based.

We then perform simulations to evaluate the *extent* of performance gain that can be achieved by using the two proposed schemes. While it is obvious that hardware multicasting should help improve multicast performance in any type of network, the influence of network contention can have considerable impact on multi-phase multicasting approaches. It is important therefore to study the impact of various system parameters on the performance of such schemes by simulation. Our simulations isolate the effect of increasing software startup times, system size, switch size, message length and degree of connectivity on multicast performance of both schemes, by varying one of these parameters at a time. The performance of our schemes is compared with a best possible unicast based multicast implementation—an extremely optimistic estimation of the performance achievable using unicast based multicasting schemes.

Our results show that even a naive multicasting algorithm in conjunction with minimal hardware support for multidestination routing can result in considerable gain in performance over the best unicast based solutions with performance gains by factors of up to 2.6 being realizable. Our results also confirm that a single phase multicast in irregular networks scales across destination set sizes, system sizes, switch sizes, message lengths and system connectivities, providing almost constant latency multicast with variation in all these parameters. While such a scheme requires additional hardware for decoding and modifying multidestination headers and an addition to the setup algorithm at the switches, the gain in performance (by factors of up to 7 over the best unicast based multicast) due to such schemes may very well justify this cost.

The remaining portion of this paper is organized as follows. In Sec. 2 we present our assumptions about the system and review a method for deadlock free unicast message passing. Section 3 reviews the various factors that affect tree based multidestination routing in switch based parallel systems. We then present the SSR scheme in Sec. 4 and the OPM scheme in Sec. 5. Our simulation results are presented in Sec. 6 and our conclusions are presented in Sec. 7.

2 Irregular Networks

2.1 Topology and System Model

In an irregular network processing nodes are connected to switches which are interconnected arbitrarily. An example of such a topology is shown in Fig. 1(a). The figure shows a 32 node system consisting of eight 8 port switches. The processing elements are typically complete workstations. The methods developed in this paper are general and applicable to any network, although enhancements can be made to improve performance in networks where the topology is regular (such as unidirectional and bidirectional MINs).

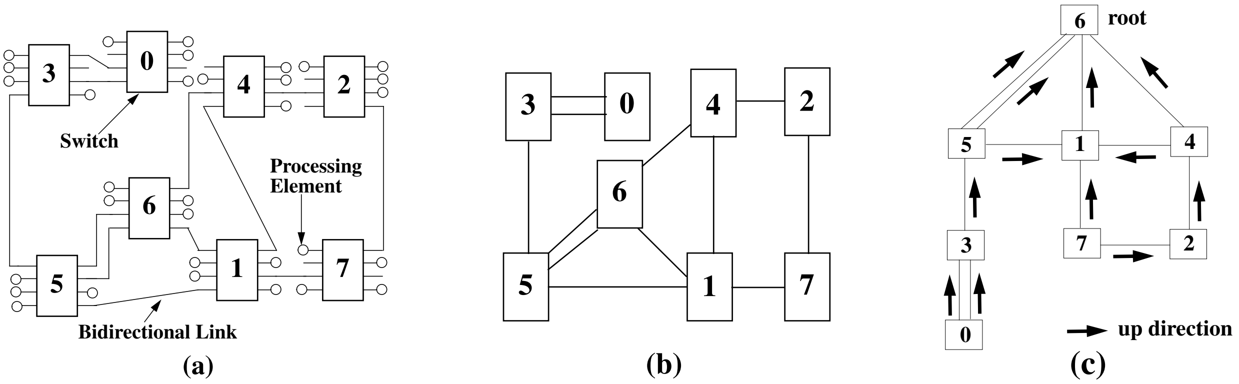

Figure 1: (a) An example of an irregular network, (b) a graphical representation of the network, and (c) the direction of links after running the setup algorithm for up*/down* routing.

The switches in the network are interconnected using bidirectional links, i.e. the ith input port and the ith output port of a given switch are connected to the jth output port and jth input port respectively of another switch. The network is assumed to support cut-through routing. To achieve cut-through routing, switches are equipped with buffers. We assume an input buffer based architecture as described in [13, 15].

For the purposes of our experimental study, we assume that half of a switch's ports are connected to processing nodes. This is not a limitation of the methods proposed in this paper but is merely an assumption made for our performance evaluation. The other ports are either connected to other switches or left unconnected. The number of ports that are left unconnected is varied to obtain networks that are increasingly better connected.

2.2 Unicast Routing in Irregular Networks

Recently, a number of approaches have been proposed for routing packets in irregular networks [1, 10, 11, 12]. An important issue in networks with cut-through routing is the avoidance of deadlock due to cyclic waiting among packets. Each of the proposed routing solutions tries to address this deadlock problem either by eliminating the possibility of cyclic wait [1, 10, 11] or by allowing cycles but by providing paths for messages to escape from cyclic wait scenarios [10, 12]. For the purposes of this paper however, we will adopt the approach outlined in [11]. This approach, called up*/down* routing, is described next.

In the up*/down* routing scheme, the switches of the network first run a setup algorithm. In this algorithm, a common (breadth first) spanning tree is formed among the switches of the network from a commonly agreed upon root switch. The spanning tree is then used to assign directions to each of the unidirectional links in the network as one of 'up' or 'down'. A link is assigned an 'up' direction if the switch at which it ends is higher up in the tree (closer to the root switch) when compared to the switch from which it originates or if the switch at which it ends is at the same level of the tree but has a lower ID. Links are marked as 'down' links otherwise. In addition to assigning directions to links, the setup procedure also results in every switch being aware of the entire network topology after which routing tables are set up at every switch. The up*/down* scheme used to perform deadlock-free routing can be summarized as follows: a message is routed on zero or more 'up' links followed by zero or more 'down' links to reach its destination. Once a message takes a 'down' link it is prohibited from taking an 'up' link again. Figure 1(b) shows a graphical representation of the irregular network of Fig. 1(a). The directions assigned to the links of the network after running the setup algorithm for up*/down* routing are shown in Fig. 1(c). The arrows indicate the 'up' links in each of the bidirectional links: the 'down' links are in the opposite direction.

The methods described in this paper make use of the information that is gathered by the setup procedure described above, adding to this procedure to set up information at the switch so as to facilitate the routing of multidestination messages. Before we describe those methods however, we briefly review some of the concepts relating to multidestination routing in switch based parallel systems [15].

3 Tree-Based Multidestination Message Passing

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[6, 8], this work has been extended recently to switch based networks in [13, 15]. A tree-based 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 tree-based 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.

3.1 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.

3.2 Replication Mechanism

Multidestination routing in switch based parallel systems requires an efficient replication mechanism at the switches[7]. In this paper, we assume that switches adopt the deadlock-free input-buffer-based asynchronous replication mechanism of [15]. 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.

3.3 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[8]. By restricting tree-based multidestination worms to such paths deadlock problems can be avoided without making special rules for multidestination messages. In this paper we assume that multidestination worms conform to the up*/down* routing in the following sense: no branch (copy) of the worm takes an 'up' link after it has taken a 'down' link.

4 Multiphase Multicast using Single Switch Replication (MPM-SSR)

A simple method for multicasting can use the replication mechanism described above at a *single* switch. In this method, 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 multinomial tree based approach is used for multicast[13]. 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 2(a) shows the irregular network of Fig. 1 with the multicast destinations highlighted and a multidestination worm that replicates at a single switch. A multinomial tree based approach to cover the destination sets is depicted in Fig. 2(b).

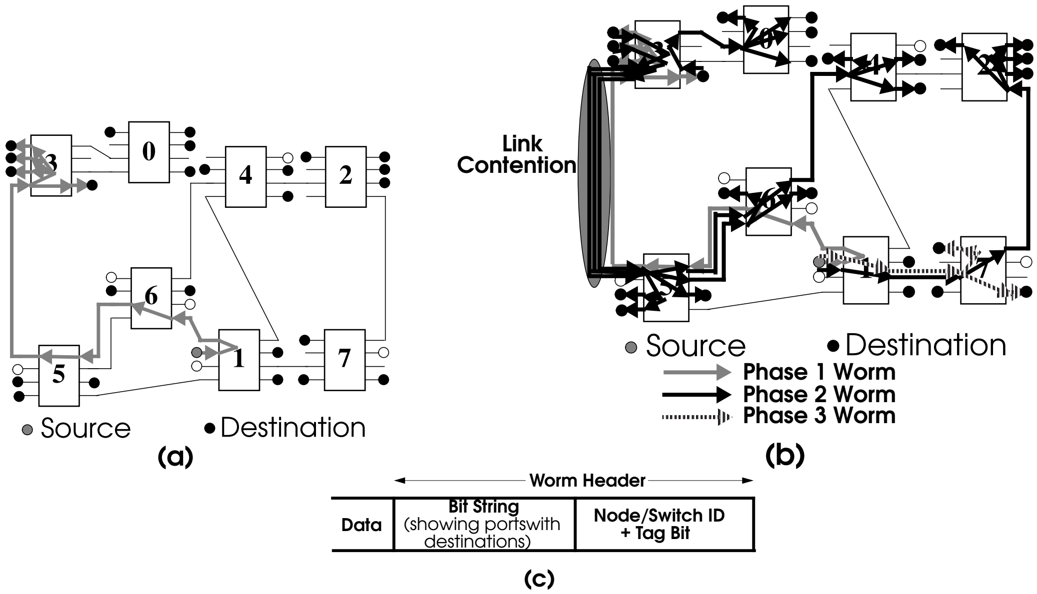

Figure 2: (a) The path followed by a sample worm, (b) the paths followed by the worms of a multinomial tree based multicast, and (b) the format of a multidestination packet header under the MPM-SSR scheme.

4.1 Encoding and Decoding Multidestination Headers

Before presenting the multinomial tree based algorithm for performing multicast to arbitrary destination sets, we first present a simple method for encoding and decoding the headers of the multidestination worms of the MPM-SSR scheme. 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 it 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.

A representation of such a header is shown in Fig. 2(c). 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 [11]). 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.

For decoding the multidestination header, the information set up for routing unicast messages is sufficient. Let us first consider 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.

If 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.

4.2 Multinomial Tree Based Multicast

The multinomial tree based multicast proceeds in multiple phases. The multicast performance depends on the number of phases in the multicast and the amount of contention within each of these phases. Given a multicast destination set and an irregular network, the destinations covered by each of the multidestination worms in the MPM-SSR scheme can be easily inferred. However, because of contention, differing distributions of the multidestination worms into the various phases of the multicast can result in differing multicast performance. Given a multicast destination set (and hence the set of multidestination worms), a distribution of the worms into various phases so that multicast latency is minimized, is an open problem. In this paper however, we adopt a simple, greedy method that focuses exclusively on the reduction of the number of phases. This method is described next.

Given a set of destinations, determine their associated switches. Arrange these switches in descending order of size according to the number of destinations in each switch. Since a worm can cover all the destinations associated with a single switch, we can associate one worm with each of these switches. Given this list of multidestination worms, in the first phase the (a) largest multidestination worm is sent from the source. In the second phase of the multicast the source and the destinations covered in the first phase send out one multidestination worm each from the ordered list of worms. At any given phase the destinations covered upto the end of the previous phase and the source send out one worm each from the ordered list of (remaining) worms. This process continues till all worms have been sent.

As can be inferred from the above description, if on the average p of the multicast destinations belong to the same switch, then multicast to d destinations using this approach requires $\lceil \log_{p+1}(d+1) \rceil$ phases. Such an approach will require fewer phases to cover a given multicast destination set than the number of phases required by the best known software multicasting algorithm[5] if $p \geq 2$. If $p = 1$ then the number of phases required for multicast is equal to the number of phases in the best known software multicasting algorithm. Since in current day parallel systems, especially those based on networks of workstations, the startup overhead incurred by the messages dominates message passing cost, a reduction in the number of phases required for multicast directly translates to better multicast performance. However, because of the presence of network contention, evaluation of the extent of this gain in performance is not straightforward: Section 6 evaluates the performance of the multicast algorithm described above.

We have described a simple multi-phase approach for multicast in irregular networks which uses extremely simple decoding logic and the ability of a switch to replicate an arriving multidestination worm to multiple output ports. In the next section, we describe a method that accomplishes multicast in a single phase, but which requires enhancements to the information stored and the decoding logic used at every switch in network.

5 One Phase Multicast using Bit-String Encoding (OPM)

Multicast using bit-string encoded multidestination worms has been proposed in [3, 9, 15]. An efficient, single phase multicasting scheme using such a multidestination worm with a bit-string encoded header has been proposed in [9, 15]. 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 network 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 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 up*/down* routing algorithm in irregular networks. Another contribution is the presentation of simple header modifying logic 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.

5.1 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 i th 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 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*[9], decoding a bit string header reduces to a simple logical operation.

5.2 Setting up Reachability Information

The setup procedure used for the up*/down* 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. 1(c) with a bidirectional link between switches 4 and 7. Two of switch 6's 'down' ports lead to the 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 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. 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. 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 and provide a possible implementation for this in the next section. 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 OPM scheme. 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 up*/down* routing) is shown in Fig. 3.

[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

```
int level = max_level;
while  (level >= 0) {
     for (i=num_switches-1; i >= 0; i--) {
          if (switches[i].BFSlevel == level) {
               /* find reachability of each of the down links */
               for (j=0; j < num_ports; j++) {
                    if (switches[i].ports[j].out_direction == DOWN) {
                         if (switches[i].ports[j].conx_type == PROCESSOR) {
                             tmp = switches[i].getProcessorId(j);
                             switches[i].ports[j].reachability_string |= tmp;
                             switches[i].total_reachability |= tmp;
                             /* Bitwise inclusive OR */
                         }
                         else if (switches[i].ports[j].conx_type == SWITCH) {
                             tmp = switches[i].getEndSwitchId(j);
                             switches[i].ports[j].reachability_string |=
                                          switches[tmp].total_reachability;
                             switches[i].total_reachability |= switches[tmp].total_reachability;
                         }
                         /* if unconnected do nothing */
                    } /* if DOWN */
               } /* for j (each port) */
          } /* if level */
     } /* for i (each switch) */
     level--;
} /* while level > 0 */

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

for (i=0; i < num_switches; i++) {
     j=0;
     while (j < num_ports) {
          if (switches[i].ports[j].out_direction == DOWN) {
               for (k= j+1; k < num_ports; k++) {
                    if (switches[i].ports[k].out_direction == DOWN) {
                         switches[i].ports[j].reachability_string ^=
                              switches[i].ports[k].reachability_string;
                         /* Bitwise exclusive OR */
                    } /* if k port dirn DOWN */
               } /* for k */
          } /* if j port dirn DOWN */
          j++;
     } /* for j */
} /* for i */
```

Figure 3: Enhancements to the setup algorithm to find the restricted reachability strings and the total reachability at a given switch in the network. The procedure `getProcessorId()` returns an N bit string with a 1 bit in the ith position if the port passed to it as argument leads to node i.

5.3 Decoding Multidestination Headers

5.3.1 Basic Idea

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 as described above. 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 choice may be made adaptively depending on local traffic conditions or may be made arbitrarily (for eg., round robin among the output ports that leads 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 up*/down* routing algorithm and the method followed for assigning 'up' and 'down' directions to the links (the set-up procedure).

5.3.2 Header Decoding and Modification Logic

As described above, to prevent multiple copies of a worm from arriving at the multicast destinations we have decided to alter the bit-string header that is forwarded to each output port (along the 'down'-ward path). An implementation that achieves this function in a switch with 4 output ports is shown in Fig. 4. The implementation consists of *reachability registers* that store the restricted reachability strings that are set up as described above. A reachability register is associated with every one of the switch's output ports (although the restricted reachability strings are only stored for those output ports that lead to 'down' links—the other registers are initialized to zero). This is required because a switch can be used in any arbitrary topology and the output ports that lead to 'down' links are different for different topologies. An additional register (the *header register*) stores the bit-string encoded header. The modified header for each of the output ports is obtained by a simple bit-wise AND of the reachability registers and the header register. This modified header is stored in a set of *result registers*: if the result in any of the registers is non-zero, a request is made for the corresponding output port. If the output port is free it begins forwarding the worm starting with the modified header in the corresponding result register. The bit-wise AND of the header register and each of the reachability registers can be carried out in parallel and completed in a single cycle. An additional cycle may be incurred for checking whether the result register is all zero to decide if the worm is to be forwarded to the corresponding port. Note that these two operations ((i) bit-wise AND of header and reachability string and (ii) checking if result is zero) may be pipelined. This logic may be replicated for every input port. Alternatively, it may be more cost-effective to have a single logic for all input ports. This has two implications. The number of result registers must be increased by a factor of k, where k is the number of input ports (output

will have an all 0 restricted reachability string in this case.

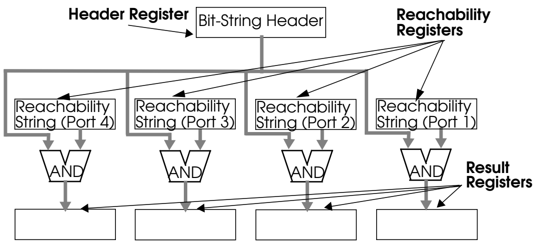

Figure 4: An example implementation for modifying the multidestination headers.

ports) per switch so that the result of one header modification does not overwrite the result of the next header modification (which may still be needed because the requested output port may not have been available). Thus k^2 registers are required in all.

The second implication is that header modification/decoding of multidestination worms that arrive simultaneously at their corresponding input buffers has to be sequentialized. This may result in an additional delay of up to $k-1$ cycles to the worm whose header is modified/decoded last. However, since the probability of multiple worms arriving at their corresponding input buffers is likely to be low, such overhead may be rarely incurred making the scheme extremely cost-effective.

In this paper however, we assume that the logical unit shown in Fig. 4 is replicated for each input port. We conservatively assume that the above header modification takes 4 cycles (40.0 nanoseconds) for our experiments, as discussed in the next section. It is to be noted that the number of bits in each of the registers specified above is identical. This register size determines the size of the bit-string allowed in the system and hence the maximum system size for which multicast can be done under the OPM scheme.

6 Performance Evaluation

We performed simulation experiments using a C++/CSIM based simulator to measure the performance of the schemes proposed in the previous section. The flit-level simulator is flexible enough to model different regular and irregular topologies. We studied the impact of the following system parameters on single multicast performance: message startup time, system size, switch size, message length and the percentage connectivity among the switches of the network. We assumed that half the number of switch ports are connected to processors/workstations and generated networks that were better connected by leaving fewer of the remaining switch ports unconnected. The interconnection between switches was randomly generated. As in [15], the switches were assumed to have input buffers deep enough to buffer the largest packet, a requirement for asynchronous deadlock free replication. When a worm header is blocked, as many as possible of the packet flits are stored in the input buffer: the flits of a blocked packet may be spread over multiple switches. Packets at an input buffer are processed in FIFO order.

In each of the experiments, we measured multicast latency over a range of destination set sizes varying one parameter at a time to isolate its impact. Every point in the graphs that follow represents the average performance for the corresponding multicast set size over 20 repetitions each for 10 different randomly generated networks (i.e. an average over 200 multicasts). The default parameters assumed for the experiments are as follows: software startup overhead (t_s) = 10.0 μsecs., link propagation time for a single flit (t_p) = 10.0 nanoseconds, multidestination header decoding/modification time (t_{route}) = 40.0 nanoseconds, switch propagation time (t_{sw}) = 10.0 nanoseconds, and injection channel and consumption channel propagation time per flit (t_{inj}

and t_{cons}) = 10.0 nanoseconds. Our default system consisted of 64 nodes interconnected by 8 port switches with 75% network connectivity.

To estimate the impact of our proposed schemes we compare them with the currently prevalent software point to point messaging based mechanisms. We present a *lower bound* estimate of the performance of such a multicast. This estimate was obtained as follows. A software multicast using point to point communication primitives requires $\lceil \log_2(d+1) \rceil$ phases. Ignoring contention, each of these phases incurs the software overhead for a single communication and the propagation time for the entire message over *at least* the injection channel, consumption channel and one switch. The estimate we used for a multicast to d destinations is given by: $((t_s + t_{inj} + t_{route}^{unicast} + t_{cons} + l \times t_{cons}) \times \lceil \log_2(d+1) \rceil)$ μsecs, where l is the message length in flits, and $t_{route}^{unicast}$ is the routing time for unicast messages (assumed to be 10.0 nanoseconds in our experiments). As shown in [5], contention free unicast based multicast is not always possible in irregular networks. Furthermore, messages may pass through multiple switches and links, so that the propagation cost of these messages may be higher. Actual multicast latency using unicast messages will therefore be higher than our lower bound estimate.

6.1 Effect of Startup Time

We first study the impact of startup overhead on the multicast performance. As shown in Fig. 5 the performance of the MPM-SSR scheme shows some variation with startup overhead. For startup times of 10.0 and 20.0 μsecs., the MPM-SSR scheme shows up to a factor of 2 improvement over the best unicast based solution. For a low startup time of 5 microseconds, the performance gain of the multicast under the MPM-SSR scheme reduces to a factor of 1.4. This is due to increased contention among the worms of the multicast at low values of startup. The single phase OPM multicast has the same factor of gain over the best unicast based multicast across the range of startup times. The main component of the multicast under this scheme is the single startup overhead.

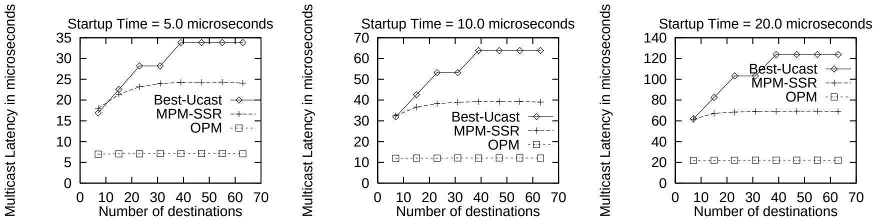

Figure 5: Multicast latency vs. number of destinations for three different communication startup times: 5.0, 10.0, and 20.0 microseconds.

6.2 Effect of System Size

The performance of the OPM scheme is dominated by the single startup overhead. As shown in Fig. 6 it therefore performs uniformly well across a range of destination set and system sizes, incurring approximately constant multicast latency. The performance *gain* due to the OPM multicast therefore increases with system size, from a factor of up to 5 for a 32 node system to a factor of up to 7 for a 128 node system over the best unicast based multicast scheme (a factor of up to 3–5 over the MPM-SSR scheme). Under the MPM-SSR scheme the amount of network contention increases as the multicast set size increases lowering the performance gain from a factor of up to 1.7 for a 32 node system to a factor of up to 1.5 for a 128 node system.

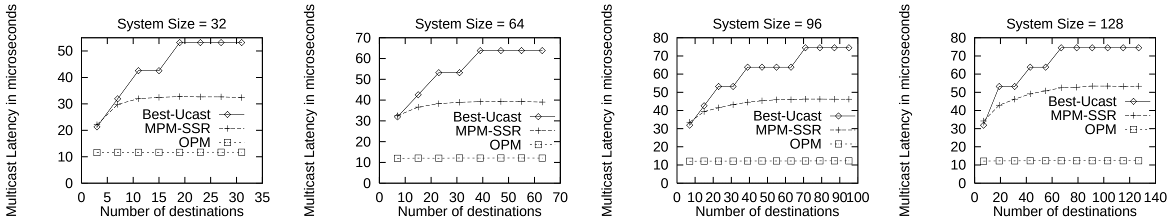

Figure 6: Multicast latency vs. number of destinations for four different system sizes: 32, 64, 96, and 128 nodes.

6.3 Effect of Switch Size

Since the multidestination worms under the MPM-SSR scheme can cover the destinations attached to a single switch, the number of destinations covered by each of them varies directly with the switch size. Therefore, for a given system size, the number of phases required for multicast under the MPM-SSR scheme reduces directly with increasing switch size and multicast performance therefore improves (as shown in Fig. 7). Compared to the best unicast based multicast performance, multicast using the MPM-SSR scheme results in gains by factors of up to 1.6 for a 64 node system with 8 port switches and in gains by factors of upto 2.6 for a similar system with 32 port switches. Again, OPM multicast incurs approximately constant latency across the range of destination sets and switch sizes.

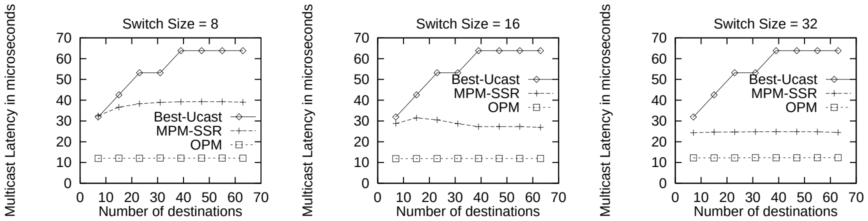

Figure 7: Multicast latency vs. number of destinations for three different switch sizes: 8, 16, and 32 ports.

6.4 Effect of Message Length

As shown in Fig. 8, increasing message length adversely affects the performance of multicast under the MPM-SSR scheme because the amount of network contention increases with increasing message length. However, even for a message length of 256 flits, MPM-SSR multicast performs better than the best unicast based multicast (by a factor of 1.15). Although increasing message lengths increase the propagation time for OPM multicast, the difference in performance is very small because of the dominance of the single startup overhead.

6.5 Effect of Percentage Connectivity

The effect of increased network connectivity on single multicast performance under both the MPM-SSR and OPM schemes is very small (almost imperceptible in Fig. 9). Increased network connectivity in general improves the performance of both MPM-SSR and OPM multicast, although by a very small amount. The reason why increase in network adaptivity does not have a more pronounced effect on our performance figures is that we did not restrict the worms to minimal paths and the

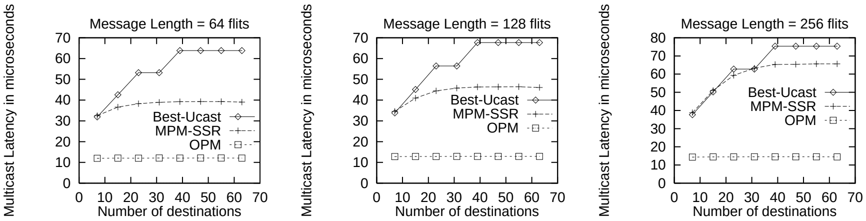

Figure 8: Multicast latency vs. number of destinations for three different message lengths (payload sizes): 64, 128, and 256 flits.

fact that the up*/down* routing does not provide as much adaptivity as some of the more recent schemes ([1, 10]). Because of this, the increased network connectivity could not be better made use of.

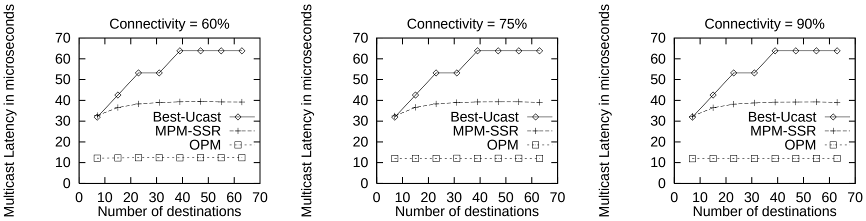

Figure 9: Multicast latency vs. number of destinations for three different degrees of connectivity: 60, 75, and 90%.

7 Conclusion

We have presented two schemes for multicast using multidestination worms. The MPM-SSR scheme is predominantly software based and just uses the ability of a switch to replicate an arriving worm to multiple output ports. Under this scheme, multicast to arbitrary destination sets proceeds in multiple phases with destinations receiving the message in one phase acting as sources in succeeding phases. We have presented a header encoding and decoding method for the multidestination worms of this scheme that requires no additional information at the switch than what is already present for routing unicast messages. The second scheme, called the OPM scheme, is a predominantly hardware based and carries out multicast efficiently in a single phase. However it requires additions to the setup algorithm and modifications to the switch to support header modification and decoding. We have developed this addition to the setup algorithm as well as an implementation for carrying out header modification [14].

Our simulation results have shown that even the naive multicasting algorithm used in the MPM-SSR scheme can result in considerable performance gains over a unicast based multicasting scheme. Furthermore, our results confirm that a hardware multicasting scheme, such as the OPM scheme, can result in scalable multicast; multicast latency under this scheme is almost constant across a range of destination set sizes, system sizes, switch sizes, message lengths and network connectivities. These schemes therefore show considerable promise for being applied to parallel systems based on networks of workstations to achieve efficient multicast. We are currently studying enhancements to the naive multicasting algorithm used in the MPM-SSR scheme that work towards reducing network contention. We are also examining whether these schemes can make better use of higher network connectivities when used in conjunction with other routing algorithms (such as those of [1, 10, 12]).

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

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.

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

[3] 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.

[4] 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.

[5] 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, 1997.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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. Available from http://www.cis.ohio-state.edu/~panda/paper.html.

[14] R. Sivaram, D. K. Panda, and C. B. Stunkel. Multicasting in Irregular Networks with Cut-Through Switches using Tree-based Multidestination Worms. Technical report, The Ohio State Univeristy, March 1997.

[15] 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 ACM Annual International Symposium on Computer Architecture (ISCA-24)*, June 1997. Available from http://www.cis.ohio-state.edu/~panda/paper.html.

[16] 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.