# Optimal Software Multicast in Wormhole-Routed Multistage Networks *

*Hong Xu*

Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292-6695

*Ya-Dong Gui and Lionel M. Ni*

Department of Computer Science
Michigan State University
East Lansing, MI 48824-1027

**Abstract**

Multistage interconnection networks are a popular class of interconnection architecture for constructing scalable parallel computers (SPCs). The focus of this paper is on wormhole routed multistage networks supporting turnaround routing. Existing machines characterized by such a system model include the IBM SP-1, TMC CM-5, and Meiko CS-2.

Efficient collective communication among processor nodes is critical to the performance of SPCs. A system-level multicast service, in which the same message is delivered from a source node to an arbitrary number of destination nodes, is fundamental in supporting collective communication primitives including the application-level broadcast, reduction, and barrier synchronization. This paper addresses how to efficiently implement multicast services in wormhole-routed multistage networks, in the absence of hardware multicast support, by exploiting the properties of the switching technology. An optimal multicast algorithm is proposed. The results of implementations on a 64-node SP-1 show that the proposed algorithm significantly outperforms the application-level broadcast primitives provided by currently existing collective communication libraries including the public domain MPI.

## 1 Introduction

Multistage interconnection networks are a popular class of interconnection architecture for constructing scalable parallel computers (SPCs), such as the BBN TC-2000, IBM SP-1, and NEC Cenju-3. In such systems, processor nodes are interconnected through multistage interconnection networks. Each processor node has its own processor, local memory, and other supporting devices. As the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system scales up as well.

Efficient data communication among processor nodes is critical to the performance of message-based SPCs. Generally data communication can be classified into point-to-point communication and collective communication [1]. While point-to-point communication deals with the basic *send* and *receive* operations between two nodes, collective communication deals with communication that involves a group of nodes. Multiple collective communication groups may space-share a SPC in a way that processor nodes in the system can be partitioned into several disjoint subsets each of which is dedicated to a distinct group. As a result, a system-level multicast service, in which the same message is delivered from a source node to an arbitrary number of destination nodes, is fundamental in supporting collective communication primitives including the application-level broadcast, reduction, and barrier synchronization [2, 3].

However, most existing message-based SPCs support only unicast communication, single-destination message passing, in hardware. In these environments, multicast must be implemented in software by sending unicast messages. One way to implement multicast in such systems is *separate addressing* in which a separate copy of the message is sent from the source to every destination. As the number of destinations increases, separate addressing may require excessive time because many systems allow a local processor to send only one or few messages at a time. An alternative approach is *multicast tree* in which the source sends the message to only a subset of the destinations. Each recipient of the message forwards it to some subset of the destinations that have not yet received it. The focus of this paper is on multicast tree implementation, also known as *unicast-based* or *software* multi-

*This work was supported in part by NSF grants CDA-9121641 and MIP-9204066, and DOE grant DE-FG02-93ER25167.



1063-9535/94 $4.00 © 1994 IEEE

Ex.1161 / IPR2025-00913 / Page 1 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

cast implementation.

Issues of software multicast implementation in direct (point-to-point) networks have been addressed in [4]. In [4], an optimal unicast-based multicast algorithm is developed for direct networks characterized by multi-dimensional mesh topology and one-port communication architecture. Recently, several algorithms [5, 6, 7] are proposed for efficient unicast-based implementation of collective communication services in direct networks which support all-port communication. Among them, an optimal unicast-based broadcast algorithm [7] is achieved for hypercube networks. Unlike previous work, this paper addresses more efficient software implementation of multicast services in wormhole-routed multistage networks with turnaround routing. By exploiting recent advances in switching technology, an optimal software multicast algorithm is proposed for multistage interconnection networks which support one-port communication. Significant performance improvement has been attained by the proposed algorithm against the currently existing collective communication libraries including the public domain MPI [8].

The remainder of the paper is organized as follows. Section 2 describes the system model under consideration which includes switch architecture, network topology, switching techniques, and the number of communication ports. Section 3 presents a deadlock-free routing algorithm for unicast communication. Section 4 defines the problem addressed in terms of the system model and the unicast routing algorithm. Section 5 presents an optimal software multicast algorithm for multistage networks with turnaround routing. The results of performance comparison with other collective communication libraries on a 64-node IBM SP-1 are given in Section 6. Section 7 concludes the paper and mentions several areas of future research.

# 2 System Model

Switches are the basic building blocks of multistage cube networks. In a $k \times k$ switch with turnaround connection (Figure 1), a left-hand side port is labeled $\ell_i$ and a right-hand side port is labeled $r_i$ $(0 \leq i \leq k-1)$. It is assumed that each port is bidirectional and incident with two unidirectional channels (in opposite directions) between neighboring switches. This implies that two messages can be transmitted simultaneously in opposite directions between neighboring switches. In addition, multiple messages can be relayed simultaneously within a switch, provided that each incoming message requires a unique outgoing channel. For ease of explanation, it is assumed that processor nodes are on the left side of the network, as shown in Figure 2.

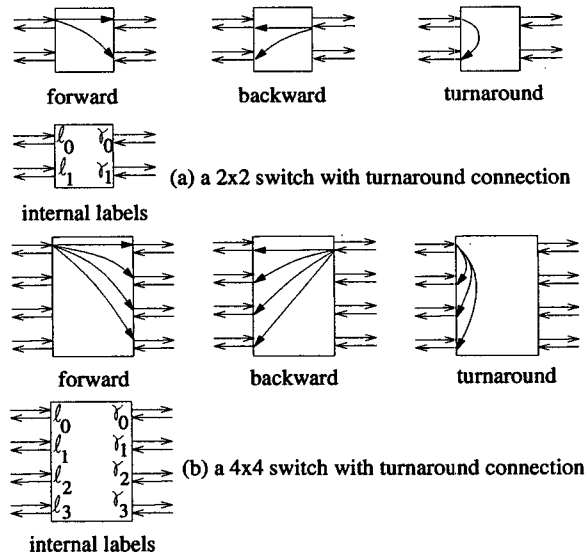


Figure 1. Examples of switches with turnaround connection

A switch supports three types of connections: *forward*, *backward*, and *turnaround*. For simplicity, the input device on port $\ell_i$ $(r_j)$ is denoted as *input port* $\ell_i$ $(r_j)$, and the output device on port $\ell_i$ $(r_j)$ is denoted as *output port* $\ell_i$ $(r_j)$. In forward connection, input port $\ell_i$ is connected to output port $r_j$, where $0 \leq i, j \leq k-1$. In backward connection, input port $r_i$ is connected to output port $\ell_j$, where $0 \leq i, j \leq k-1$. In turnaround connection, input port $\ell_i$ is connected to output port $\ell_j$, where $0 \leq i \neq j \leq k-1$. No connection is allowed from input port $r_i$ to output $r_j$, where $0 \leq i \neq j \leq k-1$. This property prevents the shortest-path unicast routing from deadlock. Note that multiple arcs in Figure 1 mean the possible connections, not hardware broadcast.

For simplicity, in this paper a channel refers to a unidirectional communication channel. A channel connecting an output port at switch $G_k$ to an input port at switch $G_{k+1}$ is called *forward* channel. A channel connecting an output port at switch $G_k$ to an input port at switch $G_{k-1}$ is called *backward* channel.

An $N$-port multistage cube network built with $k \times k$ switches can be represented as

$$C_0(N)G_0(N/k)C_1(N)\ldots C_{n-1}(N)G_{n-1}(N/k)C_n(N)$$

where $G_i$ refers to the $i^{th}$ stage, $C_i$ refers to the $i^{th}$ connection, and $N = k^n$. There are $n$ stages. Each stage $G_i$ consists of $N/k$ identical $k \times k$ switches and thus is denoted as $G_i(N/k)$. Each connection $C_i$ connects $N$ right-hand side ports at stage $G_{i-1}$ to $N$ left-hand side ports at stage $G_i$ and thus is denoted as



Ex.1161 / IPR2025-00913 / Page 2 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

$C_i(N)$. A connection pattern $C_i$ defines the topology of the one-to-one correspondence between $G_{i-1}$ ports and $G_i$ ports, also known as *permutation*. In a multistage cube network, connection pattern $C_i$ is described by the $i^{th}$ butterfly permutation $\beta_i^k$ which can be formalized as follows:

**Definition 1** *The $i^{th}$ k-ary butterfly permutation $\beta_i^k$, for $0 \leq i \leq n-1$, is defined by*

$$\beta_i^k(x_{n-1}\ldots x_{i+1}x_ix_{i-1}\ldots x_1x_0) = x_{n-1}\ldots x_{i+1}x_0x_{i-1}\ldots x_1x_i$$

*where $0 \leq x_i \leq k-1$.*

For simplicity, $\beta_0^k$ is selected to be connection pattern $C_n$. Figure 2 shows the system architecture which consists of an 8-port multistage cube network built with $2 \times 2$ switches.

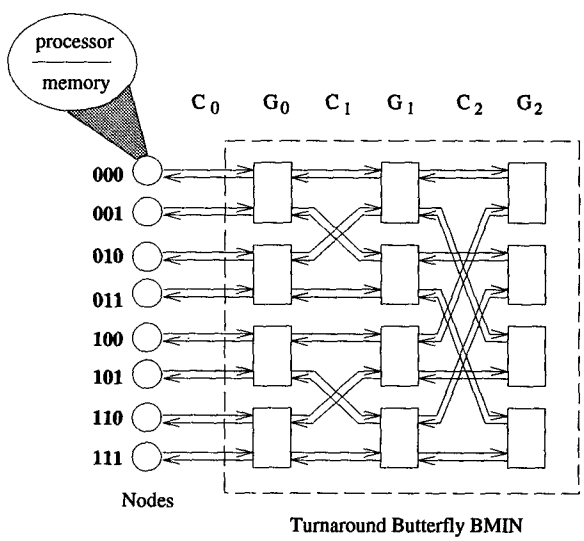


Figure 2. System architecture

In Figure 2, processor nodes represented by circles are interconnected through a multistage cube network. All nodes are attached to the same side of the network. Available ports on the other side of the network are used to configure large networks. Each node has local processor and local memory. Remote memory access can only be achieved by message-passing. Messages are relayed using the *wormhole routing* switching technique [9]. In this paper, it is assumed that there is exactly one pair of input channel and output channel connecting a node to the network, resulting in so-called "one-port communication architecture". This assumption, which is consistent with many existing multistage network systems, implies that the local processor must transmit (receive) message in sequential. Commercial SPCs using multistage cube networks with turnaround routing include the TMC CM-5 [10], Meiko CS-2 [11], and IBM SP-1 [12]. In the CM-5, the first two level stages use $4 \times 2$ switches, yielding a dual-port communication architecture.

Communication latency latency consists of three component values: *start-up latency*, *network latency*, and *blocking time* [13]. The *start-up latency* refers to the time required for message framing/unframing, memory/buffer copying, validation, and so on, at both source and destination nodes. Start-up latency can be further classified into *sending latency*, the software latency at source node, and *receiving latency*, the software latency at destination node. The *network latency* equals the elapsed time after the head of a message has entered the network at the source until the tail of the packet emerges from the network at the destination. The *blocking time* includes all possible delays encountered during the lifetime of a message. These delays are mainly due to conflicts over the use of shared resources, for example, a message encountering a busy channel or a full buffer. In this paper, *multicast latency* is used to measure multicast performance. The multicast latency refers to the time interval from when the source processor begins to send the first copy of the message until the last destination processor has received the message.

# 3 The Turnaround Routing

**Algorithm:** Turnaround routing in each switch at stage $j$
**Input:** Source address $S : s_{n-1}\ldots s_1s_0$
Destination address $D : d_{n-1}\ldots d_1d_0$
**Procedure:**
$t = FirstDifference(S, D)$ ($j \leq t$ is always true)
If $j = t$, then take a turnaround connection to output port $\ell_{d_j}$
If $j < t$ and the message comes from an input port $\ell_i$, then take a forward connection to any available output port $r_h$ ($0 \leq h \leq k-1$)
If $j < t$ and the message comes from an input port $r_i$, then take a backward connection to the ouput port $\ell_{d_j}$

Figure 3. The turnaround routing algorithm executed in each switch

The turnaround routing algorithm (Figure 3) is used for unicast communication in multistage cube networks. In a multistage cube network built with $k \times k$ switches, source address $S$ and desti-



Ex.1161 / IPR2025-00913 / Page 3 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

nation address $D$ are represented by $k$-ary numbers $s_{n-1} \ldots s_1 s_0$ and $d_{n-1} \ldots d_1 d_0$, respectively. The function *FirstDifference*$(S, D)$ returns $t$, the position where the first (leftmost) different digit appears between $s_{n-1} \ldots s_1 s_0$ and $d_{n-1} \ldots d_1 d_0$. More formally, it can be defined as follows:

**Definition 2** *FirstDifference(S,D)= $t$ if and only if $s_t \neq d_t$ and $s_i = d_i$ for $t < i < n$.*

To route a message from source to destination, the message is first sent forward to stage $G_t$. It does not matter which switch at stage $G_t$ the message reaches. Then, the message is turned around and sent backward to destination. As it moves forward to stage $G_t$, a message always takes forward channels. There may be multiple choices as to which forward channel to take in each stage $G_j$ where $0 \leq j < t$. The decision can be resolved by randomly selecting from among those forward channels which are not blocked by other messages. After the message has attained a switch at stage $G_t$, it takes the unique path from that switch backward to its destination. The backward routing path only consists of backward channels and can be determined by the "destination tag" routing: the message takes backward channel ejecting from output port $\ell_{d_j}$ in a switch at stage $G_j$. Note that the number of forward channels and the number of backward channels traversed by a turnaround routing path are always same. The turnaround routing is essentially the fat tree routing [14]. The detailed study and performance evaluation of turnaround routing can be found in [15].

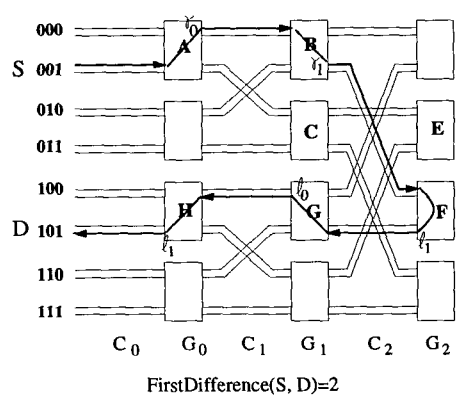


Figure 4. An example of turnaround routing

Figure 4 gives an example of turnaround routing in an 8-port multistage cube network built with $2 \times 2$ switches. In Figure 4, the function *FirstDifference*(001,101) returns 2. The message is first sent from $S$ to any switch at stage $G_2$, say $F$. Note that path $A \rightarrow B \rightarrow F$ is randomly selected. An alternative could be $A \rightarrow C \rightarrow E$. Then, the message is turned around in $F$ and sent backward to $D$. Path $F \rightarrow G \rightarrow H \rightarrow D$ is the unique path from $F$ to $D$. This path is determined by taking output port $\ell_1$ on $F$, output port $\ell_0$ on $G$, and output port $\ell_1$ on $H$.

Since a message only turns around once from a forward channel to a backward channel, the dependency graph for the routing paths selected in this way is free from cycles. Therefore, the turnaround routing is deadlock free.

By the butterfly connection, stage $G_t$ where $t =$*FirstDifference*$(S, D)$ is the least common ancestor of processors $S$ and $D$. Therefore, the turnaround routing is the shortest-path routing. On the other hand, when it moves forward, a message can choose an arbitrary forward channel at a switch (there is no redundancy for backward channels). There are multiple choices of the shortest path which the turnaround routing may select between a source and a destination. This property can be formalized as follows:

**Theorem 1** *In an $N$-port multistage cube network built with $k \times k$ switches ($N = k^n$), there are $k^t$ number of the shortest paths between source $S$ and destination $D$, each of which can be generated by the turnaround routing, where $t =$FirstDifference(S,D).*

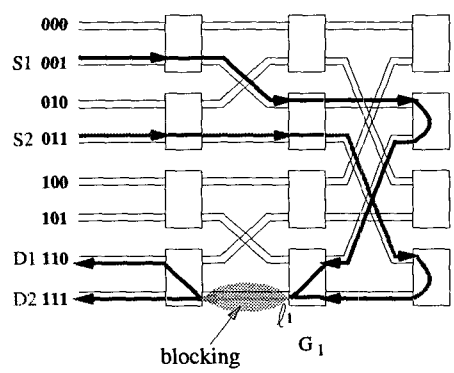


Figure 5. Blocking network

Since the forward channel selection is random, it is possible for two messages transmitted simultaneously to compete for the same backward channel, even though these two messages have the distinct sources and the distinct destinations. As a result, the multistage cube network with turnaround routing is a blocking network. In Figure 5, the message sent from node 011 to node 111 and the message sent from node 001 to node 110 contend for a common channel, backward channel ejecting from port $\ell_1$ at the bottom switch in stage $G_1$.



Ex.1161 / IPR2025-00913 / Page 4 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

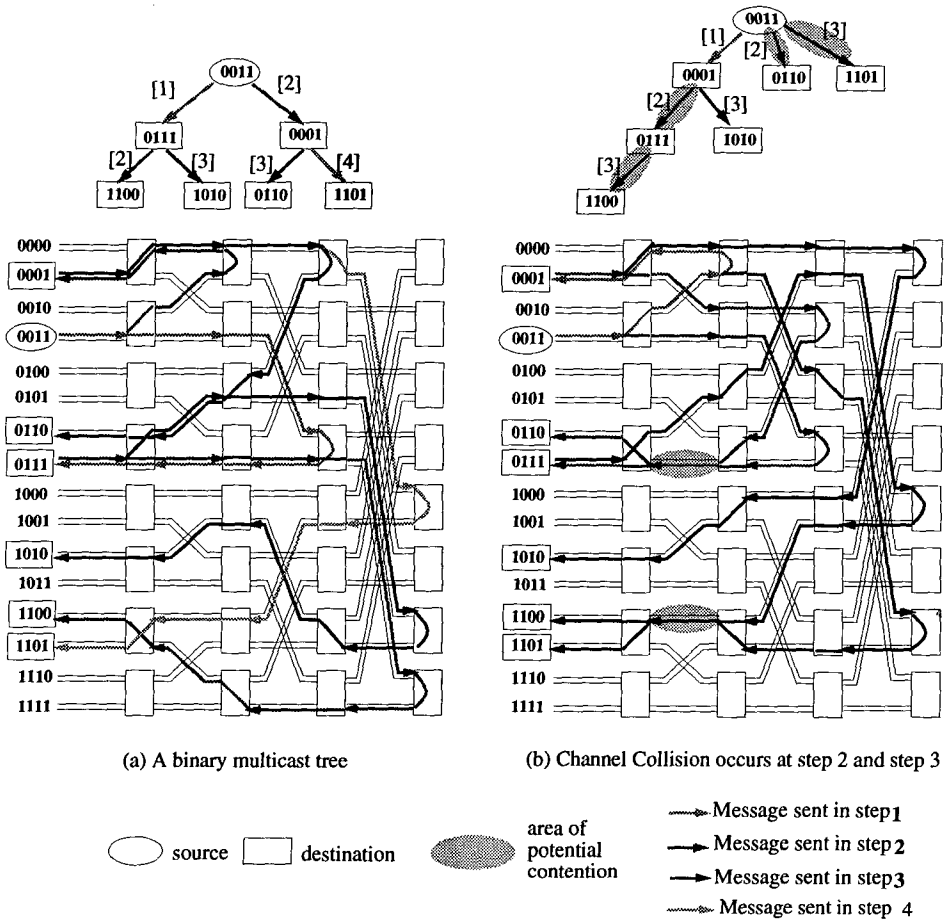


Figure 6. Unicast-based software multicast trees

# 4 The Problem

Although hardware implementations of multicast communication would intuitively offer better performance than software implementations, many such implementations are restricted in their use. For example, the TMC CM-5 can only support hardware broadcast by using a separate control network. The NEC Cenju-3 claims to provide restricted multicast in hardware with the limitation that all destination addresses must be consecutive.

In fact, many existing wormhole-routed SPCs support only unicast communication in hardware. In these environments, the multicast tree must be implemented in software by sending multiple unicast messages. However, unlike separate addressing, in a multicast tree, the source node actually sends the message to only a subset of the destinations. Each recipient of the message forwards it to some subset of the destinations that have not yet received it. The process continues until all destinations have received the message.

Which types of multicast trees to use depends on the switching strategy and unicast routing algorithm. An efficient multicast tree involves no local processors other than the source and destination processors, exploits the distance-insensitivity of wormhole routing, and is of minimum height, specifically, height $\lceil \log_2(m) \rceil$ for $m - 1$ destination nodes. Another key requirement is that there be no channel contention among the constituent messages of the multicast. That is, the unicast messages involved should not simultaneously require the same channel. Addressing the practical consideration of channel contention among the constituent messages of multicast operations, and the theory behind the resultant algorithms, distinguishes the approach presented in this paper from previous investigations. The following (small-scale) example is used to illustrate issues and difficulties involved in implementing efficient multicast communication in wormhole-routed multistage cube networks with turnaround routing. As shown in Figure 6, consider the 16-port multistage cube net-



Ex.1161 / IPR2025-00913 / Page 5 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

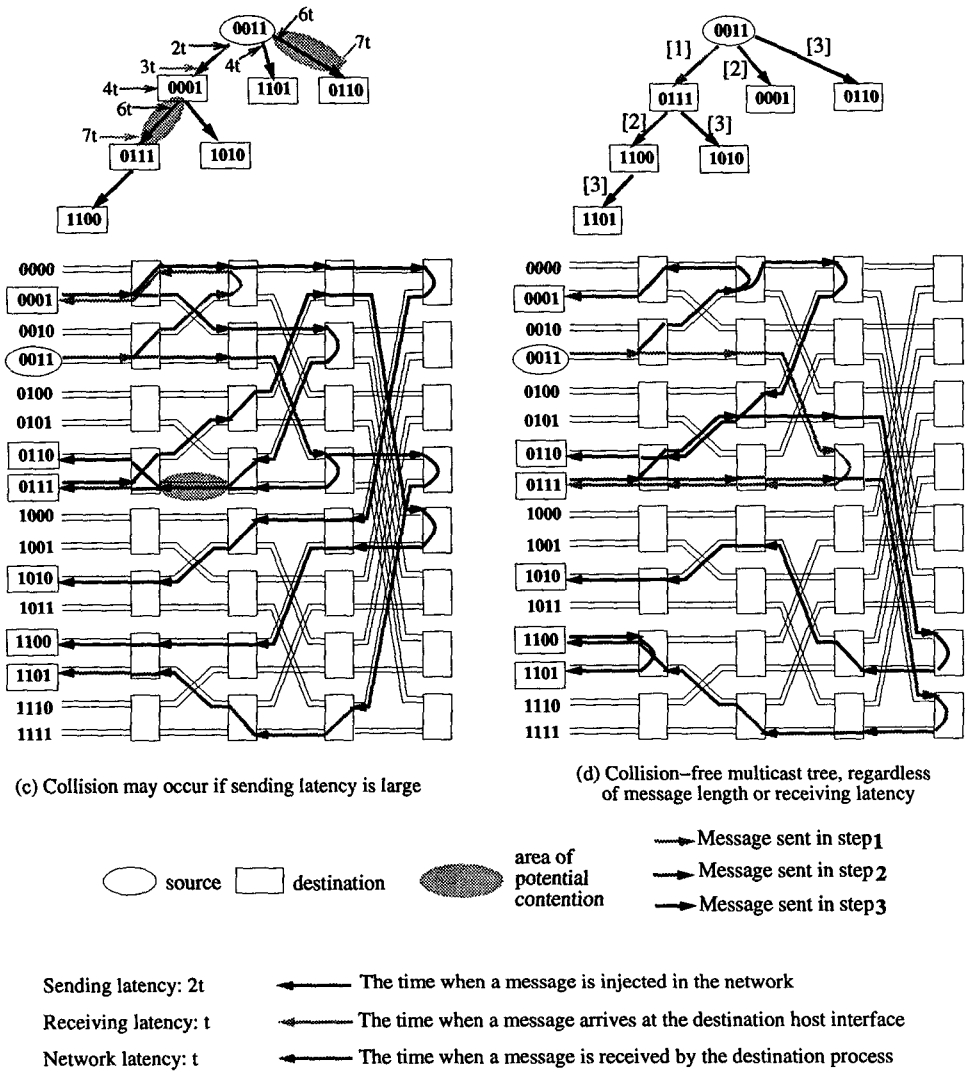


Figure 7. Unicast-based software multicast trees (cont'd)

work built with $2 \times 2$ switches. Suppose a multicast message is sent from source 0011 to six destinations $\{0001, 0110, 0111, 1010, 1100, 1101\}$.

Figure 6(a) shows a binary multicast tree. At step 1, the source sends the message to node 0111. At step 2, nodes 0011 and 0111 inform nodes 0001 and 1100, respectively. Continuing in this fashion, this implementation requires 4 steps to reach all destinations. Taking the advantage of the distance insensitivity of wormhole routing, the duration of each step must be approximately equal to the duration of a single unicast transmission of the message. In other words, it should be correct to assume that each step requires unit time as long as there exists no channel contention among the messages transmitted during each step. For this reason, the multicast latency in Figure 6(a) is 4 time steps. In Figure 6(b), the shape of the tree is rearranged in a way that the number of steps to complete the tree can be reduced to 3. However, closer inspection reveals that the message sent from node 0001 to node 0111 and the message sent from node 0011 to node 0110 in step 2 use a common channel. The contention for that channel would force one of the messages to block while the other is using the channel. Consequently, these two unicasts cannot take place during a single time step. The channel collision which occurs between two messages transmitted simultaneously within the same step in a multicast tree is known as *stepwise contention* [4]. Similarly, there is another stepwise contention between the message sent from node 0111 to node 1100 and the message sent from node 0011 to node 1101. As a result, the mul-



Ex.1161 / IPR2025-00913 / Page 6 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

ticast latency in Figure 6(b) is actually larger than 3 time steps.

This situation is rectified in Figure 7(c), where the messages sent within a particular time step do not contend for common channels. Contention among messages sent in *different* steps may arise, however, if the message length is small and the sending latency is large. In the IBM SP-1, the sending latency is about $20\mu$sec, the receiving latency is about $9\mu$sec, and the network latency is about $0.16\mu$sec per byte. For ease of explanation, we model the sending latency $2t$ and the receiving latency $t$. The length of the multicast message is chosen in a way that its network latency is $t$. As shown in Figure 7(c), the message transmission from node 0001 to node 0111 and the message transmission from node 0011 to node 0110 take place concurrently during the time period between $6t$ and $7t$, and contention occurs for the shadowed channel. The channel collision which occurs among concurrently transmitted messages due to small message size or large start-up latency is known as *depth contention.*

The multicast tree in Figure 7(d), which is based on the methods to be presented in the following section, is both stepwise contention-free and depth contention-free. Generally speaking, with no channel collision, given $(m-1)$ destination nodes, $\lceil \log_2 m \rceil$ is the greatest lower bound of the number of time steps a multicast algorithm can attain.

# 5 The U-min Algorithm

This section describes *U-min*, an optimal multicast algorithm (Figure 8) which achieves the greatest lower bound of the number of time steps. The name "U-min" is chosen for the unicast-based multicast for multistage interconnection networks.

In Figure 8, source and destination addresses are first sorted in the lexicographic order, known as a *lexicography-ordered chain.* A lexicography-ordered chain is denoted as $\Phi$, at the time when multicast is initiated by calling the U-min algorithm. The source successively divides $\Phi$ in half. If the source is in the lower half, then it sends a copy of the message to the smallest destination (with respect to the lexicographic order) in the upper half. That destination will be responsible for delivering the message to the other destinations in the upper half, using the same U-min algorithm. If the source is in the upper half, then it sends a copy of the message to the largest destination in the lower half. The source continues this procedure until $\Phi$ contains only its own address.

Figure 9 shows how to obtain the optimal multicast implementation (Figure 7(d)) by using the U-min algorithm. The source begins with a lexicography-ordered chain $\Phi = \{0001, 0011, 0110, 0111, 1010, 1100, 1101\}$. As shown in Figure 9, the source 0011 first sends to node 0111, the node with the lowest address in the upper half of $\Phi$. The upper half is deleted from $\Phi$, and therefore the nodes remaining in $\Phi$ are $\{0001, 0011, 0110\}$. Since it locates in the middle of the lower half chain, source 0011 next sends to node 0001, the nearest node on the left-hand side. Finally, 0011 sends to 0110. Each of the receiving nodes is likewise responsible for delivering the message to the nodes in its subtree using the same algorithm. As shown in Figure 9, this multicast implementation requires 3 time steps.

**Algorithm: U-min Algorithm**
**Inputs:** $\Phi$: lexicography-ordered chain $\{d_\ell, d_{l+1}, \ldots, d_r\}$ for source and destination addresses
$d_s$: the address of source nodes
**Procedure:**
**while** $\ell < r$ **do**
  **if** $s < \frac{\ell+r}{2}$ **then** /* send right */
    $c = \ell + \lceil \frac{r-\ell}{2} \rceil$;
    $D = \{d_c, d_{c+1}, \ldots, d_r\}$;
    $r = c - 1$;
  **else if** $s > \frac{\ell+r}{2}$ **then** /* send left */
    $c = \ell + \lfloor \frac{r-\ell}{2} \rfloor$;
    $D = \{d_\ell, \ldots, d_{c-1}, d_c\}$;
    $\ell = c + 1$;
  **else** /* send left */
    $c = s - 1$;
    $D = \{d_\ell, \ldots, d_{c-1}, d_c\}$;
    $\ell = s$;
  **endif**
  Send a message to node $d_c$ with the address field $D$;
**endwhile**

Figure 8. The U-min algorithm

Developing an optimal software multicast algorithm for a specific system such as a multistage cube network with turnaround routing requires a detailed understanding of potential conflicts among messages. The following properties are critical to the development of the U-min algorithm because they indicate how channel contention may be avoided. The formal proof of these theorems are omitted here and can be found in [16].

**Theorem 2** *In a multistage cube network with turnaround routing, $u$, $v$, $x$, and $y$ are processor node addresses. If $u < v < x < y$ or $v < u < y < x$, then there is no channel shared by the message transmitted*



Ex.1161 / IPR2025-00913 / Page 7 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

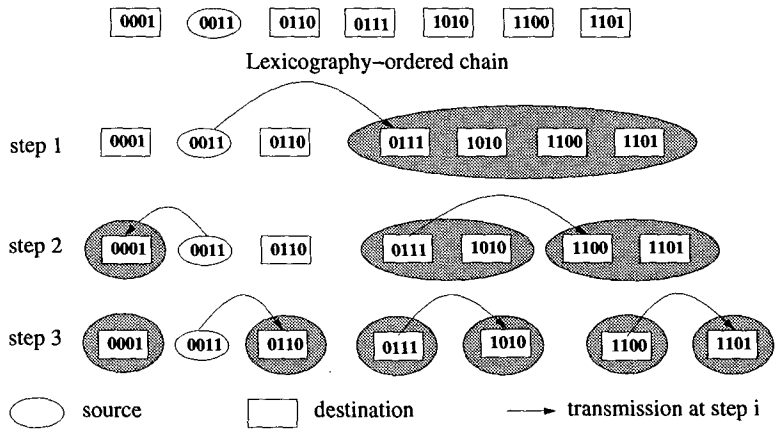


Figure 9. An example of using the U-min algorithm

*from u to v and the message transmitted from x to y.*

**Theorem 3** *In a multistage cube network with turnaround routing, u, v, x, and y are processor node addresses. If $u < v < x < y$, then there is no channel shared by the message transmitted from x to y and the message transmitted from v to u.*

In the U-min algorithm, $\Phi$ can be partitioned into two halves. The *lower half* and the *upper half* are defined on either side (with respect to the lexicographic order) of the source address. By the construction of the U-min algorithm, any message sent by a node $d_i$ in the lower half will be destined for another node $d_j$, $d_j < d_i$, in the lower half. Similarly, any message sent by a node $d_i$ in the upper half will be destined for another node $d_j$, $d_i < d_j$, in the upper half. Thus, Theorem 2 can be used to prove both stepwise contention-free and depth contention-free message transmission within the lower half and the upper half, respectively. Furthermore, Theorem 3 proves that no channel contention can exist in the constituent unicast messages between the two halves. As a result, the proposed algorithm achieves the minimum multicast latency in a turnaround-routed multistage cube network built with any $k \times k$ switches.

# 6 Experimental Results

Our experimental study was conducted on a 64-node IBM SP-1 at Argonne National Laboratory. In the IBM SP-1, each processor node is an IBM RS6000. Each frame consists of 16 processor nodes interconnected through eight $4 \times 4$ switches in two stages, as shown in Figure 10. Note that the rightmost stage in our system model (Figure 2) only performs the turnaround routing. This redundant stage can be removed by the proper wiring which connects two frames directly. The interconnection network architecture of a 32-port IBM SP-1 is topologically equivalent to the model shown in Figure 10 with no need of the rightmost stage. The unused links can be used to construct larger networks.

Only wormhole-routed unicast communication is supported in hardware on the SP-1. Broadcast, multicast, and other collective communication primitives are resorted to the software implementations. A unicast message is transmitted using the turnaround routing when the source processor and the destination processor are allocated to the same frame, such as the message transmitted from node 023 to node 030 and the message transmitted from node 122 to node 131 in Figure 10. If the source processor and the destination processor are allocated to different frames, a unicast message will be sent by taking a shortest path across two frames, such as the message transmitted from node 000 to node 101 in Figure 10. As a result, the unicast routing across two frames can also be characterized by the turnaround routing.

The EUI-H, a high-performance external user interface, is provided as the low-overhead implementation of the message-passing library in the IBM SP-1 [17]. In the SP-1 available at Argonne National Laboratory, two message-passing interface softwares are implemented upon the EUI-H: Chameleon and MPI-F. Chameleon [18] is a collection of routines that support parallel programming on various distributed-memory parallel platforms. MPI-F [19] is an experimental implementations of the emerging Message Passing Interface (MPI) standard for the IBM SP-1. The MPI-F collective communication library, however, has not yet been optimized and uses the same routines available in the public domain MPI [8]. Both Chameleon and MPI-F provide collective communication routines including the application-level broadcast. In both environments,



Ex.1161 / IPR2025-00913 / Page 8 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

any subset of processors can form an application group in which a broadcast operation can be invoked. Such an application-level broadcast operation is essentially a system-level multicast.

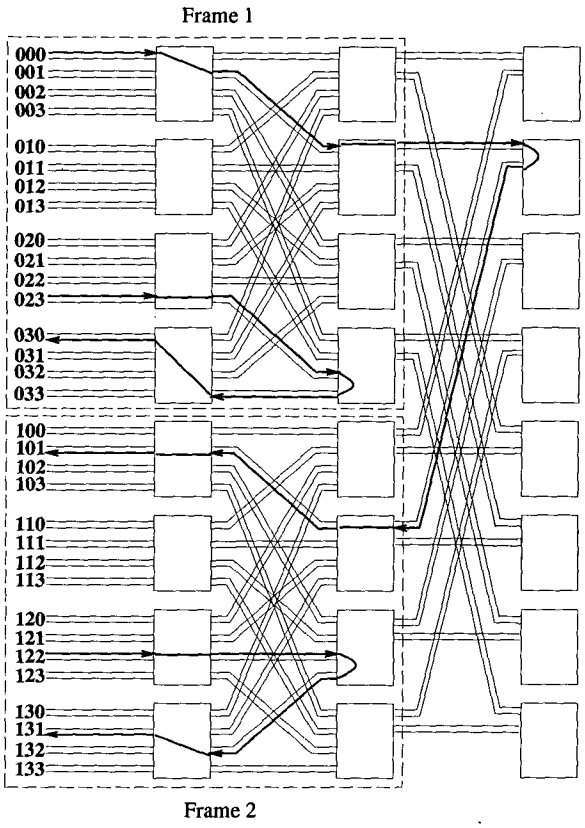


Figure 10. A 32-port multistage network topologically equivalent to a 32-port IBM SP-1

Figure 11 compares four multicast implementations on a 64-node IBM SP-1: separate addressing, Chameleon application-level broadcast, MPI-F application-level broadcast, and software U-min algorithm. The separate addressing and U-min algorithm are implemented upon the EUI-H through Chameleon. As shown in Figure 11, the U-min algorithm outperforms the other three approaches in all cases.

Based on the authors' experience in programming the SP-1 at Argonne National Laboratory, the mapping between the virtual processor identifier and the physical processor identifier is completely random. As a result, the U-min algorithm first sorts source and destinations in the lexicographic order with regard to their physical processor addresses. This ordering property makes the performance of the U-min algorithm superior over Chameleon broadcast and MPI-F broadcast operations. The performance of the U-min algorithm is consistently about 30% better than the

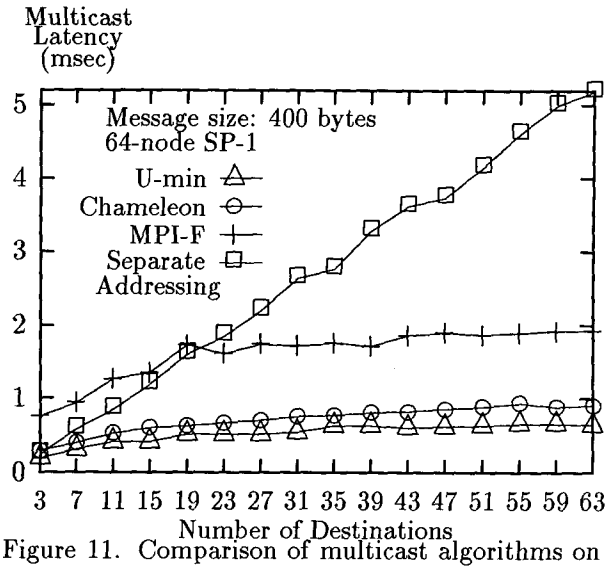


Figure 11. Comparison of multicast algorithms on a 64-node IBM SP-1

Chameleon application-level broadcast. The performance improvement is expected to be greater for a multicast message with a larger size or for a larger system. Using the bit vector representation, the overhead of sorting source and destination addresses into the lexicographic order becomes insignificant and thus can be neglected. In a 64-node system, the elapsed time for executing the sorting routine is less than 9 $\mu$sec which is only 4% of the latency of a single unicast transmission for a 400-byte message.

# 7 Conclusions

This paper has proposed an optimal multicast algorithm for wormhole routed multistage networks that offer the turnaround unicast routing and provide a single bidirectional port at each node connecting the local processor to the interconnection network. The proposed multicast algorithm attains both stepwise contention freedom and depth contention freedom. This property guarantees that the minimum multicast latency can always be achieved regardless of the message size or startup latency. The algorithm has been implemented on a 64-node IBM SP-1 and has demonstrated its superiority over other application-level broadcast services provided by currently existing collective communication libraries. The proposed algorithm can be readily applied to the Meiko CS-2 which uses the similar interconnection network as the IBM SP-1.

Though the one-port communication architecture model used in this paper is consistent with most commercial SPCs, the research of efficient data communication on multiple-port communication architecture is



Ex.1161 / IPR2025-00913 / Page 9 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

becoming popular. One area of the authors' current research concerns construction of an efficient algorithm specifically for wormhole-routed multistage networks which support multiple-port communication, such as the TMC CM-5. In addition, the concern of channel collision avoidance can be applied to other collective communication primitive design in order to explore the maximum system performance.

## Acknowledgements

The authors would like to thank Dr. Howard C.T. Ho of IBM Almaden Research Center for his many useful suggestions regarding this work and the Argonne National Laboratory for the use of their IBM SP-1 machine, respectively.

## References

1. Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," tech. rep., University of Tennessee, Mar. 1994.

2. H. Xu, P. K. McKinley, and L. M. Ni, "Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 172 – 184, October 1992.

3. H. Xu, E. T. Kalns, P. K. McKinley, and L. M. Ni, "ComPaSS: A communication package for scalable software design," accepted to appear in *Journal of Parallel and Distributed Computing*, 1994.

4. P. K. McKinley, H. Xu, A. H. Esfahanian, and L. M. Ni, "Unicast-based multicast communication in wormhole-routed direct networks," in *Proceedings of the 1992 International Conference on Parallel Processing*, vol. II, pp. 10–19, Aug. 1992.

5. A. Bar-Noy, J. Bruck, C.-T. Ho, S. Kipnis, and B. Schieber, "Computing global combine operations in the multi-port postal model," in *Proceedings of the fifth IEEE symposium on parallel and distributed processing*, pp. 336–343, Dec. 1993.

6. D. F. Robinson, D. Judd, P. K. McKinley, and B. H. C. Cheng, "Efficient collective data distribution in all-port wormhole-routed hypercubes," in *Proceedings of Supercomputing'93*, pp. 792–801, Nov. 1993.

7. C.-T. Ho and M.-Y. Kao, "Optimal broadcast on hypercubes with wormhole and E-cube routings," in *Proceedings of the 1993 International Conference on Parallel and Distributed Systems*, pp. 694–697, 1992.

8. W. D. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.

9. W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

10. C. E. Leiserson *et al.*, "The network architecture of the Connection Machine CM-5," in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, (San Diego, CA.), pp. 272–285, Association for Computing Machinery, 1992.

11. Meiko Limited, Waltham, MA., *Computing Surface: CS-2 Communications Networks*, 1993.

12. C. B. Stunkel *et al.*, "Architecture and implementation of Vulcan," in *Proc. of the 8th International Parallel Processing Symposium*, pp. 268–274, Apr. 1994.

13. L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, pp. 62 – 76, Feb. 1993.

14. C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, pp. 892–901, Oct. 1985.

15. L. M. Ni, Y. Gui, and S. Q. He, "Performance evaluation of multistage wormhole networks with turnaround routing," Tech. Rep. MSU-CPS-ACS-96, Michigan State University, July 1994.

16. H. Xu, Y.-D. Gui, and L. M. Ni, "Optimal software mulitcast in wormhole-routed multistage networks," Tech. Rep. MSU-CPS-ACS-91, Michigan State University, Mar. 1994.

17. W. Gropp, E. Lusk, and S. Pieper, "Users Guide for the ANL IBM SP-1 DRAFT," Tech. Rep. ANL/MCS-TM-00, Argonne National Laboratory, Feb. 1994.

18. W. Gropp and B. Smith, "Users manual for the Chameleon parallel programming tools," Tech. Rep. ANL-93/23, Argonne National Laboratory, June 1993.

19. H. Franke, "MPI-F: An MPI implementation for IBM SP-1," Feb. 1994. Available on anonymous ftp from info.mcs.anl.gov.



Ex.1161 / IPR2025-00913 / Page 10 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.