Bandwidth Efficient All-to-All Broadcast on Switched Clusters

Ahmad Faraj Pitch Patarasuk Xin Yuan
Department of Computer Science, Florida State University
Tallahassee, FL 32306
{*faraj, patarasu, xyuan*}*@cs.fsu.edu*

Abstract

We develop an all-to-all broadcast scheme that achieves maximum bandwidth efficiency for clusters with tree topologies. Using our scheme for clusters with cut-through switches, any tree topology can support all-to-all broadcast as efficiently as a single switch connecting all machines when the message size is sufficiently large. Since a tree topology can be embedded in almost any connected network, it follows that efficient all-to-all broadcast can be achieved in almost all topologies, regular or irregular. To perform all-to-all broadcast efficiently on clusters with store-and-forward switches, the algorithm must minimize the communication path lengths in addition to maximizing bandwidth efficiency. This turns out to be a harder algorithmic problem. We develop schemes that give solutions to common cases for such systems. The performance of our algorithms is evaluated on Ethernet switched clusters with different topologies. The results confirm our theoretical finding. Furthermore, depending on the topology, our algorithms sometimes out-perform the topology-unaware algorithms used in MPI libraries, including MPICH and LAM/MPI, to a very large degree.

1 Introduction

Clusters of workstations, which employ a pile of inexpensive commodity workstations and networking devices, have become a common environment for high performance computing. High-end clusters are usually connected by *cut-through* switches such as Myrinet [6] while low-end clusters may still use *store-and-forward* switches such as Ethernet. We will use the term cut-through/store-and-forward cluster to refer to a cluster with cut-through/store-and-forward switches. In a cut-through cluster, the communication time of a message is virtually independent of the communication path length. In a store-and-forward cluster, the communication time of a message may be significantly affected by the path length of the message. In this paper, we consider homogeneous clusters with an arbitrary (regular or irregular) topology, where all links and all machines are the same.

All–to-all broadcast, also known as all-gather [13], is one of the most common collective communication operations in high performance computing. In this operation, each node sends the same data to all other nodes in the system. The Message Passing Interface routine that realizes this operation is *MPI_Allgather* [13]. Let the message size be $msize$, the number of machines be P, and the link bandwidth be B. By definition, each node must receive a total of $(P-1) \times msize$ data from other nodes. Thus, the minimum time to complete the operation is $\frac{(P-1)\times msize}{B}$. This is the absolute lower bound on the time to complete all–to–all broadcast. Regardless how the network is connected, no all–to–all broadcast algorithm can have a shorter time.

We develop an algorithm that achieves maximum bandwidth efficiency for all–to–all broadcast on tree topologies. Using this algorithm, all–to–all broadcast on a cut-through cluster with any tree topology has a completion time close to $\frac{(P-1)\times msize}{B}$, the lower bound. In other words, using this algorithm, the reduction in the network connectivity in a tree topology as compared to the system connected by a single switch almost results in no performance degradation for this operation. Since a tree topology can be embedded in most connected networks, it follows that the nearly optimal all-to-all broadcast algorithm can be obtained for most topologies, regular or irregular, by first finding a spanning tree of the network and then applying our algorithm. Note that some routing schemes may prevent a tree from being formed in a connected network. Our approach cannot be applied to such systems.

In order to perform all-to-all broadcast efficiently on a store-and-forward cluster, the algorithm must minimize the communication path lengths in addition to achieving maximum bandwidth efficiency. This turns out to be a much harder algorithmic problem. While we cannot prove formally, we suspect this problem to be NP-complete. We develop schemes that give solutions to special cases that are common in practice. Notice that even for store-and-forward clusters, the major issue is still bandwidth efficiency. This is particularly true for clusters connected by a small number of switches, which are common in practice.

Based on our algorithms, we develop an automatic routine generator that takes the topology information as input and generates topology-aware all-gather routines. We evaluate the performance of the algorithms using an Ethernet switched cluster with different network topologies. Our performance study confirms that our algorithms achieve nearly optimal performance on clusters with different topologies. Using our algorithms, the performance of all–to–all broadcast on multiple switches is similar to that on a single switch. The study also shows that the topology-unaware algorithms used in MPICH[12] and LAM/MPI[11] are not effective on some topologies. Depending on the topology, our algorithms sometimes out-perform the MPICH and LAM/MPI routines to a very large degree (e.g. by a factor of more than 10).

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the network model and the problem definition. Section 4 details the schemes to solve the problems. Section 5 reports experimental results. Finally, the conclusions are presented in Section 6.

2 Related work

All–to–all broadcast has been extensively studied. Many all–to–all broadcast algorithms were designed for specific network topologies that are used in parallel machines, including hypercube [8, 20], mesh [15, 18, 21], torus [21], k-ary n-cube [20], fat tree [10], and star [14]. Work in [9] optimizes MPI collective communications, including $MPI\_Allgather$, on wide area networks. The study in [3] investigates efficient all-to-all broadcast on SMP clusters. Work in [22] explores the design of NIC-based all-gather with different algorithms over Myrinet/GM. Some efforts [16, 19] have focused on developing algorithms for different message sizes, and some of these algorithms have been incorporated in the recent MPICH library [12]. In [2], the authors developed an efficient all-gather algorithm for small messages. The study in [1] compares a dissemination-based all-gather with the recursive doubling algorithm [19] on Ethernet and Myrinet. The most related research to this work is presented in [7] where nearly optimal all-to-all broadcast schemes were developed for clusters connected by one or two switches. Yet, as indicated in [7] and [5], the algorithm proposed for the general topology in [7] is not always optimal for clusters with more than two switches. In this paper, we develop schemes that allow performing nearly optimal all-to-all broadcast on almost all topologies.

3 Network model and problem definition

We consider homogeneous clusters of workstations that consist of a number of workstations connected by switches. A cluster can be either a cut-through cluster or a store-and-forward cluster. We assume that all machines and all links in the cluster are the same. Each switch may connect to a number of workstations and to some other switches. The links operate in the duplex mode that supports simultaneous communications on both directions of the link with the full bandwidth. Each switch provides a crossbar connectivity for its input and output ports.

We focus on bandwidth efficient all–to–all broadcast schemes, which can be applied when the message size is sufficiently large. Hence, we will assume that the message size, $msize$, is sufficiently large such that communication time is dominated by the bandwidth term. Other communication overheads, such as software startup overheads, are relatively insignificant and are ignored. Let the path length for the message be d. In a cut-through cluster with no network contention, the communication time for an $msize$-byte message is roughly $\frac{msize}{B}$. Note that the communication time in a cut-through cluster is independent of d. Let pkt be the packet size in a store-and-forward cluster. The communication time for an $msize$-byte message in a store-and-forward cluster is roughly $\frac{msize}{B} + (d-1) \times \frac{pkt}{B}$. Depending on the value of $msize$ and pkt, the term $(d-1) \times \frac{pkt}{B}$, introduced by the store-and-forward mechanism, may significantly affect the overall communication time.

In a topology where there are multiple paths between two nodes, the routing issue needs to be considered. However, the major result of this paper is that a tree topology can support the all–to–all broadcast as efficiently as any other topology. Since our techniques are developed for the tree topology, where there is only a single path between each pair of nodes and the routing issues do not exist, we will focus on the tree topology and ignore the routing issues in the rest of the paper. Figure 1 shows an example cluster that has a tree topology. Some networks such as Ethernet only support the tree topology [17] while others such as Myrinet can support arbitrary irregular topologies.

The network can be modeled as a directed graph $G = (V, E)$ with nodes V corresponding to switches and machines and edges E corresponding to unidirectional channels. Let S be the set of switches in the network and M be the set of machines in the network. $V = S \cup M$. Let $u, v \in V$, a directed edge $(u, v) \in E$ if and only if there is a link between node u and node v. We will call the physical connection between node u and node v link (u, v). Thus, link (u, v) corresponds to two directed edges (u, v) and (v, u) in the graph. For the tree topology, we assume that the switches can only be intermediate nodes while the machines can only be leaves. A switch as a leaf in the tree will not participate in any communication and, thus, can be removed from the graph. We assume that there is at least one switch in the tree and the root of the tree is a switch. The *root switch* is only used to identify the starting point for the Depth First Search (DFS) and postorder traversal algorithms. The location of the root switch is otherwise insignificant in our algorithms.

The terminologies used in this paper are defined next. A *message*, $u \rightarrow v$, is a communication transmitted from node u to node v. A *pattern* is a set of messages. We will use the notion $u \rightarrow v \rightarrow w \rightarrow ... \rightarrow x \rightarrow y$ to represent the pattern that consists of messages $u \rightarrow v$, $v \rightarrow w$, ..., and $x \rightarrow y$. The notion $path(u, v)$ denotes the set of directed edges in the unique path from node u to node v. For example, in Figure 1, $path(n1, n4) = \{(n1, S1), (S1, S0), (S0, n4)\}$. The *path length* is defined as the number of switches a message travels through. For example, the path length of $n1 \rightarrow n4$ is 2. Two messages, $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$, are said to have *contention* if they share a common edge. A pattern is *contention free* if there is no contention between each pair of the messages in the pattern. $|S|$ denotes the size of set S.

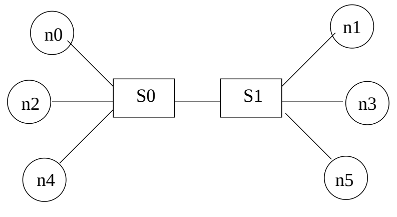


Figure 1. An example switched cluster with a tree topology

3.1 The logical ring algorithm

As discussed in the introduction, the minimum time to complete all-to-all broadcast with a message size of $msize$ is $\frac{(P-1)\times msize}{B}$. We will show that this lower bound can be approached for clusters with a tree topology, which implies that this bound can be approached for almost any topology since a tree topology can be embedded in most connected networks. Our schemes use the logical ring algorithms, which were proposed for single-switch clusters and two-switch clusters [7, 12]. Let the cluster contain P machines, numbered as n_0, n_1, ..., n_{P-1}. Let $F : \{0, ..., P-1\} \rightarrow \{0, ..., P-1\}$ be a one-to-one mapping function. Thus, $n_{F(0)}$, $n_{F(1)}$, ..., $n_{F(P-1)}$ is a permutation of n_0, n_1, ..., n_{P-1}. The algorithm works by repeating the following *logical ring pattern* $P-1$ times:

$$n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)} \rightarrow n_{F(0)}.$$

In the first iteration, each node $n_{F(i)}$, $0 \le i \le P-1$, sends its own data to node $n_{F((i+1)\ mod\ P)}$ and receives data from node $n_{F((i-1)\ mod\ P)}$. In subsequent iterations, each node $n_{F(i)}$ forwards what it received in the previous iteration to node $n_{F((i+1)\ mod\ P)}$ (and receives from node $n_{F((i-1)\ mod\ P)}$). After $P-1$ iterations, all data from all nodes reach all nodes in the system. Note that in each iteration, each node must copy the data it receives into the right place of the output buffer.

3.2 Problem definition

All logical ring algorithms operate in the same fashion. The key for a logical ring algorithm to achieve good performance is to find the logical ring pattern that carries out communications as efficiently as possible. This is the problem we consider in this paper.

Let the slowest communication time in the logical ring pattern be $t_{slowest}$, the total communication time is $(P-1) \times t_{slowest}$. In a cut-through cluster, if there exists a mapping such that the logical ring pattern is contention free, $t_{slowest} \approx \frac{msize}{B}$ and the total time for the all-to-all broadcast operation is $T \approx \frac{(P-1)\times msize}{B}$, which is the theoretical lower bound. Hence, for a cut-through cluster, the challenge is to find a mapping such that the logical ring pattern is contention free. This problem is stated as follows.

Problem 1 (finding a contention free logical ring): Let $G = (S \cup M, E)$ be a tree graph. Let the number of machines in the system be $P = |M|$, and let the machines in the system be numbered as n_0, n_1, ..., n_{P-1}. The problem is to find a one-to-one mapping function $F : \{0, 1, ..., P-1\} \rightarrow \{0, 1, ..., P-1\}$ such that the logical ring pattern $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(p-1)} \rightarrow n_{F(0)}$ is contention free.

For clusters with store-and-forward switches, assuming that the logical ring pattern is contention free and that the longest path length in the pattern is d, $t_{slowest} \approx (d-1)\frac{pkt}{B} + \frac{msize}{B}$, and the total time is $T \approx \frac{(P-1)\times msize}{B} + (d-1) \times (P-1) \times \frac{pkt}{B}$. Hence, to minimize the communication time, the logical ring must (1) be contention free, and (2) have the smallest d, the longest path length in the ring. This problem is stated as follows.

Problem 2 (Finding a contention free logical ring with the smallest maximum path length): Let $G = (S \cup M, E)$ be a tree graph. Let the number of machines in the system be $P = |M|$, and let the machines in the system be numbered as n_0, n_1, ..., n_{P-1}. The problem is to find a mapping function $F : \{0, 1, ..., P-1\} \rightarrow \{0, 1, ..., P-1\}$ such that (1) the logical ring pattern $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)} \rightarrow n_{F(0)}$ is contention free, and (2) $\max_{0 \le i \le P-1} \{length(n_{F(i)} \rightarrow n_{F((i+1) \ mod \ P)})\}$ is minimized.

Clearly, Problem 1 is a sub-problem of Problem 2. Unfortunately, we are only able to develop a polynomial time solution for Problem 1, but not for Problem 2. We strongly suspect that Problem 2 is NP-complete although we cannot prove it formally. We consider a special case of Problem 2 that is common in practice and establish the necessary and sufficient conditions for a cluster to have a contention-free logical ring with a maximum path length of 2. It must be noted that obtaining a contention free logical ring pattern is the major issue even in store-and-forward clusters. The topologies in most practical clusters have small diameters. The solution for Problem 1 can be directly applied to such clusters to obtain nearly optimal performance.

Since the network contention can significantly affect the performance of the logical ring algorithm, using the logical ring algorithm without carefully mapping the machines, as in MPICH[19], may result in extremely poor performance. Consider the example in Figure 1. Using the logical ring algorithm in MPICH [19], where the mapping function is the identity function, the communication pattern in each iteration is $n0 \rightarrow n1 \rightarrow n2 \rightarrow n3 \rightarrow n4 \rightarrow n5 \rightarrow n0$. The link from $S0$ to $S1$ is used three times in an iteration. The proper mapping may have a logical ring of $n0 \rightarrow n2 \rightarrow n4 \rightarrow n5 \rightarrow n3 \rightarrow n1 \rightarrow n0$. In this case, the MPICH algorithm would perform roughly three times worse than the algorithm with a properly mapped logical ring due to the contention on link (S0, S1). This behavior is observed in our performance evaluation.

4 Constructing contention free logical rings

4.1 Problem 1

Let $G = (S \cup M, E)$ be a tree graph. Let the number of machines in the system be $P = |M|$ and the machines in the system be numbered as n_0, n_1, ..., n_{P-1}. We will call this numbering scheme global numbering. We assume that all switches are intermediate nodes in the tree. Let $G' = (S, E')$ be a subgraph of G that only contains switches and the links between switches. The algorithm, which will be called *Algorithm 1*, determines the mapping for a contention free logical ring pattern in two steps.

- Step 1: Number the switches based on the Depth First Search (DFS) of G'. An example DFS numbering of the switches is shown in Figure 2. We will denote the switches as s_0, s_1, ..., $s_{|S|-1}$, where s_i is the ith switch arrived in DFS traversal of G'.

- Step 2: Let the X_i machines connecting to switch s_i, $0 \le i \le |S|-1$, be numbered as $n_{i,0}$, $n_{i,1}$, ..., n_{i,X_i-1}. We will call this local numbering. A one-to-one mapping function (and its reverse function) can be established between the global numbering and local numbering. X_i may be 0 when no machine is connected to s_i. The contention free logical ring is $n_{0,0} \rightarrow ... \rightarrow n_{0,X_0-1} \rightarrow n_{1,0} \rightarrow ... \rightarrow n_{1,X_1-1} \rightarrow ...n_{|S|-1,0} \rightarrow ... \rightarrow n_{|S|-1,X_{|S|-1}-1} \rightarrow n_{0,0}$ (we will formally prove this). The mapping function F for the above logical ring pattern can be obtained using the mapping function from the global numbering to the local numbering.

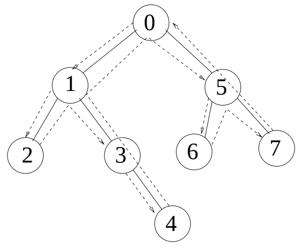


Figure 2. DFS numbering

Lemma 1: Let $G' = (S, E')$ be the subgraph of G that contains only switches and links between switches. Let s_0, s_1, ..., $s_{|S|-1}$ be the DFS ordering of the switches, where s_i is the ith switch arrived in DFS traversal of G'. Communications in the following pattern are contention free: $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_{|S|-1} \rightarrow s_0$.

Proof: An example is depicted in Figure 2, which shows the contention free pattern $0 \rightarrow 1 \rightarrow 2 \rightarrow ... \rightarrow 7 \rightarrow 0$. We will formally prove this lemma by induction.

Base case: When there is one switch, there is no communication and thus no contention.

Induction case: Assume that the communication pattern in a k-switch system does not have contention. Consider a $(k+1)$-switch system with switches s_0, s_1, ..., s_k. Removing the last switch s_k from the system, we obtain a k-switch system. The DFS ordering of the k-switch system is exactly the same as the $(k+1)$-switch system with s_k removed. Hence, from the induction hypothesis, the communication pattern in the k-switch system, that is, $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_{k-1} \rightarrow s_0$, does not have contention. Now, let us consider the $(k+1)$-switch system where we need to prove that pattern $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_k \rightarrow s_0$ does not have contention. The pattern in the $(k+1)$-switch system adds two communications $s_{k-1} \rightarrow s_k$ and $s_k \rightarrow s_0$ to and removes one communication $s_{k-1} \rightarrow s_0$ from the pattern in the k-switch system. Thus, to prove that the pattern in the $(k+1)$-switch system is contention free, we only need to show that communications $s_{k-1} \rightarrow s_k$ and $s_k \rightarrow s_0$ do not introduce contention.

Based on the way DFS operates, switch s_k must be the child of one of the switches along the path from s_0 to s_{k-1}. Hence, there are three cases to be considered: s_k is a child of s_{k-1}, s_k is a child of s_0, and s_k is a child of a switch s'_i along the path from s_0 to s_{k-1}. These three cases are depicted in Figure 3. In the proof of the three cases, we use the following two facts.

- Fact (a): the link directly connecting switch s_k does not have contention with all communications in the k-switch system. This is because the link is not part of the k-switch system.
- Fact (b): from the induction hypothesis, communication $s_{k-1} \rightarrow s_0$ does not have contention with communications in pattern $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_{k-1}$.

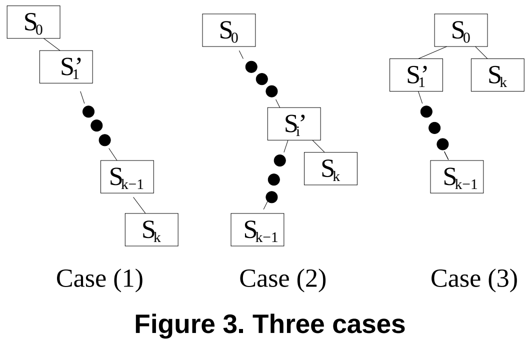


Figure 3. Three cases

Now, let us consider the three cases in Figure 3.

- Case (1): Switch s_k is a child of s_{k-1}. $s_{k-1} \rightarrow s_k$ does not have contention with any other communications (Fact (a)). $s_k \rightarrow s_0$ is the concatenation of two paths: $s_k \rightarrow s_{k-1}$ and $s_{k-1} \rightarrow s_0$. $s_k \rightarrow s_{k-1}$ does not have contention with all other communications (Fact (a)) and $s_{k-1} \rightarrow s_0$ does not introduce contention (Fact (b)).
- Case (2): Switch s_k is a child of some node s'_i along the path from s_0 to s_{k-1}. In this case, $s_{k-1} \rightarrow s_k$ is the concatenation of two paths: $s_{k-1} \rightarrow s'_i$ and $s'_i \rightarrow s_k$. $s_{k-1} \rightarrow s'_i$ does not have contention with all other communications since it is a sub-path of $s_{k-1} \rightarrow s_0$ (Fact (b)). Path $s'_i \rightarrow s_k$ does not cause contention (Fact (a)). Similar arguments apply to $s_k \rightarrow s_0$.
- Case (3): Switch s_k is a child of s_0. This follows similar arguments as in Case (1).

Thus, the pattern $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_{|S|-1} \rightarrow s_0$ is contention free. □

Theorem 1: The logical ring pattern resulted from Algorithm 1 is contention free.

Proof: Algorithm 1 basically obtains the logical ring mapping by (1) grouping all machines connected to a switch together, and (2) ordering the groups of machines based on the DFS order of the switches. To prove that the mapping is contention free, we must show that all links between a machine and a switch are contention free and all links between switches are contention free. Since each node sends and receives exactly once in the logical ring pattern, a link between a machine and a switch is used in both direction exactly once, which indicates that there is no contention on these links. For the links between switches, since the algorithm orders the group of machines (connected to each switch) based on the DFS order, it can be easily shown that the usage of the inter-switch links in the logical ring is exactly the same as the pattern described in Lemma 1. From Lemma 1, there is no contention on the links between switches. □

Using Algorithm 1, the contention free logical ring can be found for a tree topology. In networks with an arbitrary topology, this contention free logical ring can be found by first finding a spanning tree and then applying Algorithm 1. The two steps may be combined by using the DFS tree to realize the logical ring.

4.2 Problem 2

To solve Problem 2, we must find a contention free logical ring with the smallest maximum path length.

We were not able to either design a polynomial algorithm that exactly solves Problem 2 or prove this problem to be NP-complete. Thus, we have to leave this general problem open. However, we develop a scheme that gives the exact solution for a common case when each switch in the cluster directly connects to either more machines than switches or the same number of machines and switches. This special case is significant also because such a cluster is the only type that allows a contention free logical ring with a maximum path length of 2 to be formed. The result can be used by a network designer to design a store-and-forward cluster with efficient all-to-all broadcast support. Note that logical rings with a maximum path length of 1 only exist for clusters connected by a single switch. For clusters with multiple switches, the smallest possible maximum path length in the logical ring is 2 since for each switch, there exists at least one node that communicates with a node in another switch. The path length for this communication is at least 2. The following theorem summarizes the results.

Theorem 2: For a tree graph $G = (S \cup M, E)$, there exists a contention free logical ring with a maximum path length of 2 if and only if the number of machines directly connected to each switch is larger than or equal to the number of switches directly connected to the switch.

Proof: We will first prove the necessary condition. Assume that there exists a switch, A, that directly connects to more switches than machines. Let us refer to the switches directly connected to A as A-*neighbor* switches. Let all nodes connecting to A through an A-neighbor switch form an A-neighbor subtree. Clearly, the number of A-neighbor subtrees is equal to the number of A-neighbor switches. Under the assumption that all switches are intermediate nodes in the tree topology, each A-neighbor subtree contains at least one machine. Since there are more A-neighbor subtrees than the number of machines attached to A, in the logical ring, at least one machine in an A-neighbor subtree must send a message to a machine in another A-neighbor subtree. The path length of this communication is at least 3 (2 A-neighbor switches plus A). Hence, to have a logical ring with a maximum path length of 2, each switch must directly connect to at least the same number of machines as the number of switches.

Before we prove the sufficient condition, let us introduce the concept of a logical *array* pattern of a tree (or a subtree), which is rooted at a switch. We also use the term *the logical array of a switch* to denote the logical array of the tree rooted at the switch. Let the tree (subtree) contain Y nodes, n_0, n_1, ..., and n_{Y-1}. Let $F : \{0, ..., Y-1\} \rightarrow \{0, ..., Y-1\}$ be a one-to-one mapping function. The logical array pattern is $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(Y-1)}$. We distinguish the first machine, $n_{F(0)}$, and the last machine, $n_{F(Y-1)}$, of the logical array from other machines in the logical array since these two machines must be treated differently. From the definition, we can see that the logical array differs from the logical ring by excluding the last communication $n_{F(Y-1)} \rightarrow n_{F(0)}$.

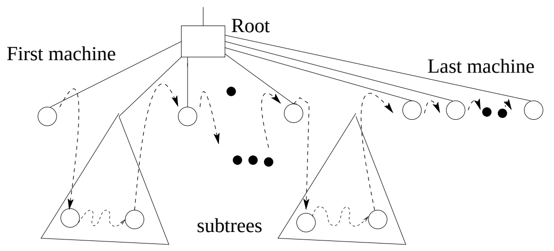


Figure 4. Constructing the logical array for an intermediate switch

Now, let us consider the sufficient condition. Assume that the number of machines directly connected to each switch is equal to or larger than the number of switches directly connected to the switch. The algorithm performs a postorder traversal of the switches. For each subtree associated with a non-root switch, the algorithm finds the logical array pattern that satisfies the following three conditions: 1) the logical array pattern is contention free, 2) the maximum path length in the pattern is less than or equal to 2, and 3) the first machine, $n_{F(0)}$, and the last machine, $n_{F(Y-1)}$, are directly connected to the root of the subtree. More specifically, the algorithm processes each non-root switch as follows.

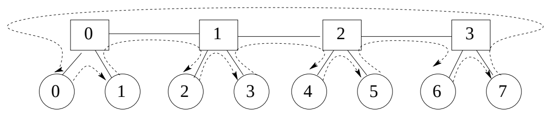
- Case (1): the switch does not directly connect to other switches except its parent. If the switch is directly connected to a single node n_0, then the first node as well as the last node in the subtree is n_0, and the logical array pattern is n_0. When the switch is directly connected to X nodes, n_0, ..., n_{X-1}, the first node is n_0, the last node is n_{X-1}, and the array pattern is $n_0 \rightarrow n_1 \rightarrow ... \rightarrow n_{X-1}$. It can be verified that the three conditions are met.
- Case (2): the switch directly connects to some switches other than its parent and some machines. We will use the term "sub-switches" to denote the switches directly connected to the current switch other than its parent. Each sub-switch is the root of a subtree. Since the switches are processed following the postorder traversal order, the logical arrays for all sub-switches have been computed. Let the current switch connect to i sub-switches, denoted as t_0, t_1, ..., t_{i-1}, and j machines, denoted as

m_0, m_1, ..., m_{j-1}. We have $j \geq i+1$ since the parent switch does not count in i. For sub-switch t_k, $0 \leq k \leq i-1$, we will use t_k^F, t_k^L, and $t_k^F \rightarrow ... \rightarrow t_k^L$ to denote the first machine, the last machine, and the logical array respectively. The logical array for the current switch is $m_0 \rightarrow t_0^F \rightarrow ... \rightarrow t_0^L \rightarrow m_1 \rightarrow t_1^F \rightarrow ... \rightarrow t_1^L \rightarrow m_2 \rightarrow ... \rightarrow m_{i-1} \rightarrow t_{i-1}^F \rightarrow ... \rightarrow t_{i-1}^L \rightarrow m_i \rightarrow m_{i+1} \rightarrow ... \rightarrow m_{j-1}$. This case is depicted in Figure 4.

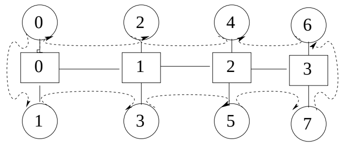
Now let us examine the three conditions for the logical array of the current switch. It is obvious that the logical array of the current switch is contention free if the logical arrays of the sub-switches are contention free. The path length for messages $m_k \rightarrow t_k^F$, $0 \leq k \leq i-1$, and messages $t_k^L \rightarrow m_{k+1}$, $0 \leq k \leq i-1$, is exactly 2 since t_k^F and t_k^L are attached to the sub-switch t_k. Since the logical arrays of sub-switches $t_k^F \rightarrow ... \rightarrow t_k^L$, $0 \leq k \leq i-1$, have a maximum path length of 2, the logical array pattern of the current switch has a maximum path length of 2. The first machine m_0 and the last machine m_{j-1} are attached to the current switch. Hence, all three conditions are met.

The processing of root is similar except that we construct the logical ring pattern instead of the logical array pattern. Let the root directly connect to i top level sub-switches and j machines. When the root does not connect to sub-switches, $i = 0$. Let us denote the i sub-switches as as t_0, t_1, ..., t_{i-1} and the j machines as m_0, m_1, ..., m_{j-1}. We have $j \geq i$. For each sub-switch t_k, $0 \leq k \leq i-1$, we use t_k^F, t_k^L, and $t_k^F \rightarrow ... \rightarrow t_k^L$ to denote the first node, the last node, and the logical array respectively. The logical ring pattern for the tree is $m_0 \rightarrow t_0^F \rightarrow ... \rightarrow t_0^L \rightarrow m_1 \rightarrow t_1^F \rightarrow ... \rightarrow t_1^L \rightarrow m_2 \rightarrow ... \rightarrow m_{i-1} \rightarrow t_{i-1}^F \rightarrow ... \rightarrow t_{i-1}^L \rightarrow m_i \rightarrow m_{i+1} \rightarrow ... \rightarrow m_{j-1} \rightarrow m_0$. Note that when $i = j$, t_{i-1}^L sends to m_0 in the logical ring. Following similar arguments as in Case (2), the ring pattern for the root is contention free with a maximum path length less than or equal to 2. Thus, when each switch connects to at least the same number of machines as the number of switches, a contention free logical ring with a maximum path length of 2 can be constructed. $\square$

The proof of the sufficient condition is a constructive algorithm. We will call this algorithm that finds a 2-hop contention free logical ring *Algorithm 2*. Figure 5 shows the results of applying Algorithm 1 and Algorithm 2 to an 8-machine cluster. As can be seen from the figure, both mappings are contention free. Algorithm 1 computes a logical ring that has a maximum hop of 4 (from node 7 to node 0 in Figure 5 (a)) while the logical ring computed using Algorithm 2 has a maximum hop of 2 as shown in Figure 5 (b). For a store-and-forward cluster, using a 2-hop logical ring is expected to perform better than a 4-hop logical ring.



(a) Logical ring from algorithm 1



(b) Logical ring from algorithm 2

Figure 5. Logical rings from Algorithm 1 and Algorithm 2

5 Experiments

We develop an automatic routine generator that takes the topology information as input and generates, based on the algorithms presented earlier, customized topology-aware all–gather routines. In the experiments, we examine the performance of the generated routines on multiple topologies and compare it to the performance of the all–gather routines in LAM/MPI 7.1.1 [11] and the recently improved MPICH 2-1.0.1 [12]. To realize the all-gather operation, LAM/MPI uses an algorithm that performs a gather operation to a root followed by a broadcast from the root to all other nodes. The gather and broadcast operations are performed using tree-based algorithms. Depending on the message size and the number of nodes, MPICH uses different algorithms to realize the all-gather operation [19]. Related to the setting in our experiments where only medium and large sized messages are considered, in a 16-machine cluster, MPICH uses a recursive doubling algorithm [19] when $msize < 32KB$ and the topology-unaware ring algorithm when $msize \geq 32KB$. In a 32-machine cluster, MPICH uses the recursive doubling algorithm when $msize < 16KB$ and the topology-unaware ring algorithm when $msize \geq 16KB$. The topology-unaware ring algorithm is basically the logical ring algorithm with the logical ring pattern of $n_0 \rightarrow n_1 \rightarrow ... \rightarrow n_{P-1} \rightarrow n_0$.

```
MPI_Barrier(MPI_COMM_WORLD);
start = MPI_Wtime();
for (count = 0; count < ITER_NUM; count ++) {
  MPI_Allgather(...);
}
elapsed_time = MPI_Wtime() - start;
```

Figure 6. Code segment for measuring MPI_Allgather performance.

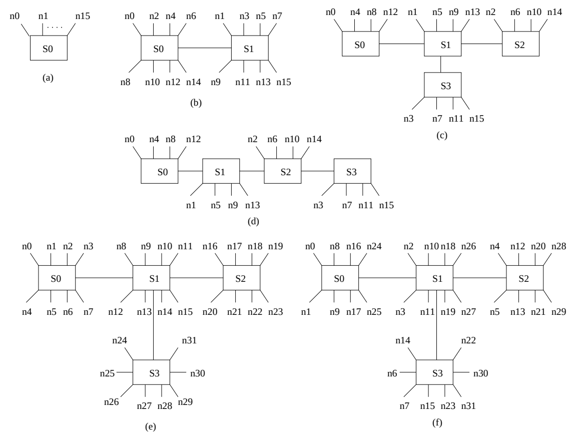


Figure 7. Topologies used in the experiments

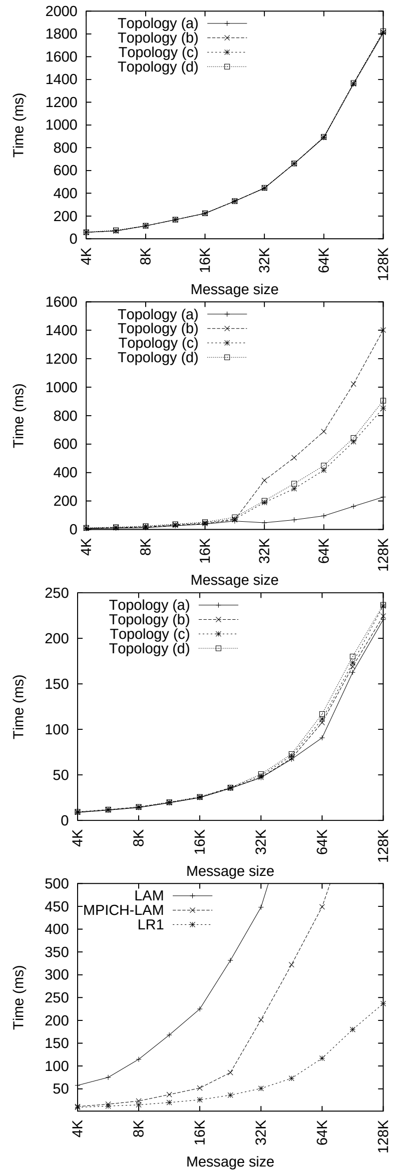
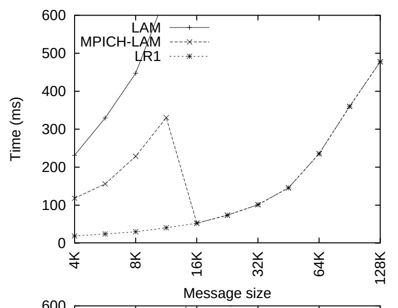


Figure 8. Performance of LR1, LAM, and MPICH-LAM on topologies (a) to (d)

Our generated all–gather routines use LAM/MPI 7.1.1 point-to-point primitives and run over LAM/MPI 7.1.1. We use LR1 and LR2 to denote the routines obtained from Algorithm 1 and Algorithm 2 respectively. To report fair performance comparison, we port the MPICH all–gather implementation to LAM/MPI. We use MPICH-LAM to represent the ported all–gather routine. We found that, in the newest MPICH, the performance of the all–gather operation using native MPICH and MPICH-LAM is very close. In the performance evaluation, LR1 and LR2 are compared with LAM and MPICH-LAM. We use the approach similar to Mpptest [4] to measure the performance of the $MPI\_Allgather$ routines. Figure 6 shows an example code segment for measuring the performance. The number of iterations is varied according to the message size: more iterations are used for small message sizes to offset the clock inaccuracy. For the message ranges 4KB-12KB, 16KB-96KB, and 128KB, we use 50, 20, and 10 iterations respectively. The results are the averages of three executions. We use the *average* time among all nodes as the performance metric.

The experiments are performed on a 32-node Ethernet switched cluster. The nodes of the cluster are Dell Dimension 2400 with a 2.8MHz P4 processor, 128MB of memory, and 40GHz of disk space. All machines run Linux (Fedora) with 2.6.5-1.358 kernel. The Ethernet card in each machine is Broadcom BCM 5705 with the driver from Broadcom. These machines are connected to Dell PowerConnect 2224 and Dell PowerConnect 2324 100Mbps Ethernet switches. Figure 7 shows the topologies we used in the experiments. Parts (a) to (d) of the figure represent clusters of 16 nodes connected by 1, 2, and 4 switches with different topologies. Parts (e) and (f) show 32-node clusters of 4 switches, each having 8 machines attached. These two clusters have exactly the same physical topology, but different node assignments. We will refer to topologies in the figure as topology (a), topology (b), topology (c), topology (d), topology (e), and topology (f).

Figure 8 shows the performance of LAM, MPICH-LAM, and LR1 on topologies (a) to (d). Figure 8 (a) shows the performance of LAM. The LAM algorithm has almost the same performance for topologies (a)-(d). The tree-based algorithms used in the all–gather implementation do not exploit all network links and do not create bottleneck links in all four topologies. However, the network is under-utilized. As can be seen in Figure 8 (d), the LAM/MPI routine performs much worse than MPICH and LR1. The results for MPICH are shown in Figure 8 (b). MPICH changes the algorithm when $msize = 32KB$. When $msize < 32KB$, MPICH uses the recursive doubling algorithm, which has simi-



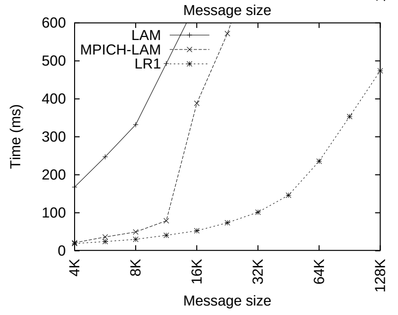


Figure 9. Performance of LR1, LAM, and MPICH-LAM on topologies (e) and (f).

lar performance for all topologies. Using the topology-unaware ring algorithm when $msize \geq 32KB$, MPICH provides very different performance for the four topologies. MPICH performs best on topology (a), where the cluster is connected by a single switch. Topologies (b), (c), and (d) result in different network contention, which significantly affects the performance of MPICH. Figure 8 (c) shows the results for LR1. We can see that LR1 achieves very similar performance for all four topologies, which is attributed to the ability of LR1 in finding the contention free logical ring on different topologies. The performance of LR1 on all topologies is similar to the performance of LR1 on the single switch topology (topology (a)). This demonstrates the optimality of LR1 in terms of achieving nearly optimal all-to-all broadcast performance on different topologies. Figure 8 compares LR1 with LAM and MPICH on topology (d). As seen in the figure, LR1 performs substantially better than LAM and MPICH. For example, when the message size is 64KB, the completion times for LR1, LAM, and MPICH-LAM are 116.9ms, 895.6ms, and 448.9ms respectively. This translates to a 666.1% speedup over LAM and 284.0% over MPICH.

Figures 9 (a) and (b) show the performance results for LAM, MPICH-LAM, and LR1 on topologies (e) and (f) respectively. We can see the extreme poor performance of LAM on both topologies. As shown in Figure 9 (a), the topology-unaware ring algorithm used in MPICH, when $msize \geq 16KB$, achieves nearly optimal performance (same as LR1) for this topology. In this case, the topology-unaware ring algorithm operates exactly the same as LR1. However,

msize	LR1 Topo.(d) (4-hop)	LR2 Topo.(d) (2-hop)	LR1 Topo.(a) (1-hop)
32KB	50.9ms	48.0ms	47.0ms
48KB	72.9ms	68.4ms	67.2ms
64KB	116.9ms	95.7ms	90.8ms
96KB	180.0ms	172.7ms	162.6ms
128KB	236.8ms	233.7ms	220.6ms

Table 1. LR1 .vs. LR2 on topology (d)

with the same physical topology and a different node assignment in topology (f), the topology-unaware algorithm performs much worse than LR1 as shown in Figure 9 (b). This again shows that the performance of MPICH depends heavily on the network configuration. Unlike LAM and MPICH, LR1 achieves the best performance for topologies (e) and (f). To illustrate, when the message size is 128KB, the completion times for LR1, LAM, and MPICH-LAM on topology (f) are 473.7ms, 5346ms, and 3595ms respectively. This means that LR1 achieves a performance that is more than 11 times better than LAM and almost 8 times better than MPICH. The figures also show that LR1 performs better than the recursive doubling algorithm when $msize > 4KB$.

Table 1 shows the impact of selecting a logical ring with a shorter path length. For topology (d), LR1 results in a logical ring with a maximum path length of 4 hops, and LR2 results in a logical ring with a maximum path length of 2 hops. In addition to the results of LR1 and LR2, the table also includes results for topology (a), which is essentially a 1-hop ring. The results for topology (a) is provided for references since no logical ring algorithm can out-perform 1-hop ring. There are two observations from the table. First, the impact of path length on the performance is noticeable, but not very large in comparison to the impact of contention. Second, by minimizing the maximum path length of the ring on the Ethernet switched cluster, some performance improvement can be obtained. In general, the 2-hop ring performs better than the 4-hop ring, but worse than the 1-hop ring. Note that the theoretical lower bound time for all-to-all broadcast with a message size of 64KB on a 16-node 100Mbps cluster is $\frac{15\times64\times1024\times8}{100\times10^6} = 78.6ms$. Considering the protocol overheads in MPI and TCP/IP layers as well as the software/hardware delays, the performance of LR2 (95.7ms) is very close to optimal.

6 Conclusion

In this paper, we develop a bandwidth efficient all-to-all broadcast scheme for switched clusters. Using the proposed scheme, a cut-through cluster with any topology that has a tree embedding can support all-to-all broadcast as efficiently as a single switch connecting all machines. We also develop techniques that can achieve good all-to-all broadcast performance on store-and-forward clusters that are common in practice. This paper posts an open question: is the problem of finding a contention free logical ring with the smallest path length on a tree topology NP-complete? We hope this open problem can be solved in the near future.

References

[1] G. D. Benson, C. Chu, Q. Huang, and S. G. Caglar, "A Comparison of MPICH Allgather Algorithms on Switched Networks," *In Proceedings of the 10th EuroPVM/MPI 2003 Conference*, Venice, Italy, pages 335–343, September 2003.

[2] J. Bruck, C. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport messagepassing systems," *IEEE Transactions on Parallel and Distributed Systems*, 8(11), pages 1143–1156, November 1997.

[3] M. Golebiewski, R. Hempel, and J. L. Traff, "Algorithms for Collective Communication Operations on SMP Clusters," *In The 1999 Workshop on Cluster-Based Computing held in conjunction with 13th ACM-SIGARCH International Conference on Supercomputing (ICS'99)*, page 1115, June 1999.

[4] W. Gropp and E. Lusk, "Reproducible Measurements of MPI Performance Characteristics." *Tech. Report ANL/MCS-P755-0699*, Argonne National Labratory, June 1999.

[5] J. Han and C. Han, "Efficient All-to-All Broadcast in Switch-Based Networks with Irregular Topology," *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, Beijing, China, pages 112–116, May 2000.

[6] Myricom homepage, http://www.myri.com.

[7] M. Jacunski, P. Sadayappan, and D.K. Panda, "All-to-All Broadcast on Switch-Based Clusters of Workstations," *Proceedings of 1999 International Parallel Processing Symposium*, San Juan, Puerto Rico, April 1999.

[8] S. L. Johnsson and C. T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Transactions on Computers*, 38(9):1249-1268, Sept. 1989.

[9] T. Kielmann and R. F. H. Hofman and H. E. Bal and A. Plaat and R.A. F. Bhoedjang, "MagPIe:MPI's Collective Communication Operations for Clustered Wide Area Systems," *In Proceeding Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Atlanta, GA, pages 131–140, May 1999.

[10] S. Kumar and L. V. Kale, "Scaling All-to-All Multicast on Fat-tree Networks," *The 10th International Conference on Parallel and Distributed Systems (ICPADS 2004)*, Newport Beach, CA, pages 205-214, July 2004.

[11] LAM/MPI Parallel Computing, Available at http://www.lam-mpi.org.

[12] "MPICH – A Portable Implementation of MPI," Available at http://www.mcs.anl.gov/mpi/mpich.

[13] The MPI Forum, "The MPI-2: Extensions to the Message Passing Interface," Available at http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html.

[14] E. Oh and I. A. Kanj, "Efficient All-to-All Broadcast Schemes in Distributed-Memory Parallel Computers," *The 16th International Symposium on High Performance Computing Systems and Applications (HPCS '02)*, IEEE, pages 65-70, 2002.

[15] D.S. Scott, "Efficient All–to–All Communication Patterns in Hypercube and Meshtopologies," *the Sixth Distributed Memory Computing Conference*, pages 398-403, 1991.

[16] A. Tam and C. Wang, "Efficient Scheduling of Complete Exchange on Clusters," *the ISCA 13th International Conference on Parallel and Distributed Computing Systems*, August 2000.

[17] A. Tanenbaum, "Computer Networks", 4th Edition, 2004.

[18] R. Thakur and A. Choudhary, "All-to-all Communication on Meshes with Wormhole Routing," *8th International Parallel Processing Symposium (IPPS)*, April 1994.

[19] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimizing of Collective Communication Operations in MPICH," *ANL/MCS-P1140-0304*, Mathematics and Computer Science Division, Argonne National Laboratory, March 2004.

[20] E. A. Varvarigos and D. P. Bertsekas, "Communication Algorithms for Isotropic Tasks in Hypercubes and Wraparound Meshes," *Parallel Computing*, 18(11):1233-1257, 1992.

[21] Y. Yang and J. Wang, "Efficient all-to-all broadcast in all-port mesh and torus networks," *Proceedings of 5th IEEE International Symposium on High-Performance Computer Architecture (HPCA-5)*, Orlando, FL, pages 290-299, January 1999.

[22] W. Yu, D. Buntinas, and D. K. Panda, "Scalable and High Performance NIC-Based Allgather over Myrinet/GM," TR-22, OSU-CISRC, April 2004.