Bandwidth Efficient All-to-All Broadcast on Switched Clusters

Ahmad Faraj
Blue Gene Software Development
IBM Corporation
Rochester, MN 55901
faraja@us.ibm.com

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

Abstract

Clusters of workstations employ flexible topologies: regular, irregular, and hierarchical topologies have been used in such systems. The flexibility poses challenges for developing efficient collective communication algorithms since the network topology can potentially have a strong impact on the communication performance. In this paper, we consider the all-to-all broadcast operation on clusters with cut-through and store-and-forward switches. We show that near-optimal all-to-all broadcast on a cluster with any topology can be achieved by only using the links in a spanning tree of the topology when the message size is sufficiently large. The result implies that increasing network connectivity beyond the minimum tree connectivity does not improve the performance of the all-to-all broadcast operation when the most efficient topology specific algorithm is used. All-to-all broadcast algorithms that achieve near-optimal performance are developed for clusters with cut-through and clusters with store-and-forward switches. We evaluate the algorithms through experiments and simulations. The empirical results confirm our theoretical finding.

Keywords: Collective communication, all-to-all broadcast, cluster of workstations

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. In such clusters, computing nodes are connected by commodity switches: high-end clusters are usually connected by *cut-through* switches, such as InfiniBand and Myrinet, while low-end

*Contact Author: Xin Yuan, xyuan@cs.fsu.edu, phone: (850)644-9133, fax: (850)644-0058.

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. The switch level topology in a cluster can be very flexible: regular, irregular, and hierarchical topologies have been used in such systems. Since the topology of a system has a strong impact on the performance of collective communication operations, it is challenging to design efficient collective communication algorithms for such systems.

In this paper, we investigate one particular collective communication operation, all-to-all broadcast, on clusters of workstations. All–to-all broadcast, also known as all-gather [16], is one of the most common collective communication operations in high performance computing. In this operation, each process sends the same data to all other processes in the system. Figure 1 visualizes the all-to-all broadcast operation on four processes.

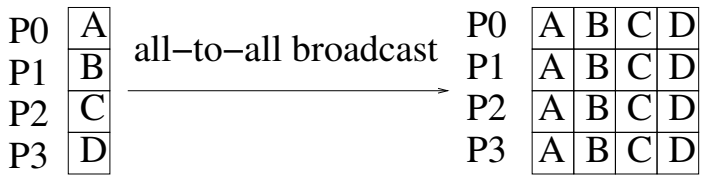


Figure 1: An example of the all-to-all broadcast operation

We focus on the all-gather operation with sufficiently large message sizes that the total communication time is dominated by the bandwidth term. Optimizing the operation with a large message size is very different from optimizing the operation with a small message size: optimizations for small messages usually reduce the communication start-up overheads by minimizing the number of messages while optimizations for large messages usually maximize the bandwidth usage and avoid network contention by considering the network topology. The techniques developed in this paper complement existing techniques (e.g. [2]) for small messages.

We consider clusters where each computing node is equipped with one network interface card (1-port systems). Let the message size for the operation be $msize$ bytes and the number of processes be P. Let us assume that each process runs on one node and that the

bandwidth of the link connecting each node to the network is B. By the definition of the all-to-all broadcast operation, each node must receive a total of $(P-1) \times msize$ bytes data from other nodes. Thus, the minimum time to complete this operation is $\frac{(P-1) \times msize}{B}$. *This is the absolute lower bound of the time to complete the all–to–all broadcast* [1]. Regardless of the type of networks, regular, irregular, or hierarchical, no all–to–all broadcast algorithm can have a shorter communication time.

We show that this absolute lower bound can be approached by only using the links in any spanning tree embedded in the network. Since a spanning tree topology can be embedded in most connected networks, it follows that near-optimal all-to-all broadcast can be obtained for most topologies, regular, irregular, or hierarchical. This also implies that when the best algorithm is used, upgrading a tree topology to other topologies such as fat-tree, mesh, or irregular topologies with more connectivity does not improve the performance of the all-to-all broadcast operation. In other words, the tree topology is as good as any other topologies for realizing this operation. Note that some routing schemes may prevent a tree from being formed in a connected network. Our techniques cannot be applied to such systems.

We develop an all-to-all broadcast algorithm that can theoretically achieve the lower bound completion time on cut-through clusters with arbitrary topologies where a spanning tree can be embedded. The algorithm has the following two properties: (1) each node communicates (sends and receives) exactly $(P-1) \times msize$ data; and (2) all communications in the operations are contention free on the embedded tree topology. These two properties guarantee the optimality of the algorithm, that is, a theoretical communication time of $\frac{(P-1) \times msize}{B}$. To perform all-gather efficiently on a store-and-forward cluster, the communication algorithm must minimize the communication path lengths in addition to having the two properties for cut-through clusters. This turns out to be a harder algorithmic problem. While we cannot

[1] For SMP or multi-core systems, each node can have multiple processors (or cores). Assuming that each node runs N processes, the processes in each node must receive $(P-N) \times msize$ data from outside the node. Hence, the lower bound is $\frac{(P-N) \times msize}{B}$, which is close to $\frac{(P-1) \times msize}{B}$ since N is usually a small number.

prove formally, we suspect this problem to be NP-complete. We identify the conditions for a store-and-forward cluster with multiple switches to support efficient all-to-all broadcast. In addition, we develop schemes that give optimal solutions for cases when each switch is connected to a small number of other switches. Such cases are common for clusters with irregular topologies for two reasons: (1) a network with an irregular topology is typically small; and (2) connecting one switch to many other switches without increasing the link speed creates a hot-spot in the network and is usually not a good practice.

Based on the proposed all-gather algorithms, we develop automatic routine generators that take the topology information as input and generate topology-aware all-gather routines. We evaluate the performance of the algorithms using a 32-machine Ethernet switched cluster with different network topologies. In addition, we also investigate the performance of the proposed algorithms on large networks through simulation. The performance study confirms that our algorithms achieve near-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 experiments also demonstrate that the proposed techniques can be used to improve existing communication libraries even on small clusters with irregular topologies.

The main contributions of this paper include the following. First, we show that the all-gather operation can be realized optimally (in theory) on the spanning tree embedding of any topology, and develop efficient all-gather algorithms that can uniformly apply to clusters with regular, irregular, and hierarchical topologies. Second, we demonstrate empirically that the proposed techniques are practical.

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 for solving the problems. Section 5 reports experimental results. Finally, the conclusions are presented in Section 6.

2 Related work

Many all–to–all broadcast algorithms were designed for specific network topologies that are used in parallel machines, including hypercube [9, 22], mesh [18, 20, 23], torus [23], k-ary n-cube [22], fat tree [12], and star [17]. Work in [11] optimizes MPI collective communications, including $MPI\_Allgather$, on wide area networks. The study in [6] investigates efficient all-to-all broadcast on SMP clusters. Work in [24] explores the design of NIC-based all-gather with different algorithms over Myrinet/GM. Some efforts [19, 21] have focused on developing algorithms for different message sizes, and some of these algorithms have been incorporated in the recent MPICH library [15]. In [2], the authors developed an efficient all-gather algorithm for small messages. This algorithm is theoretically optimal in that it uses the smallest number of messages to complete the operation. When the communication start-up overhead is the dominating factor, the algorithm achieves the best results. The algorithm, however, will not achieve high performance for large messages on irregular topologies since the network topology is not considered and the link bandwidth is not an optimization objective. Our technique complements this algorithm. The study in [1] compares a dissemination-based all-gather with the recursive doubling algorithm [21] on Ethernet and Myrinet. The most related research to this work is presented in [8] where near-optimal all-to-all broadcast schemes were developed for clusters connected by one or two cut-through switches. However, the algorithm in [8] is not always optimal for arbitrary topologies. In this paper, we develop schemes that allow performing near-optimal all-to-all broadcast on all topologies where a spanning tree can be embedded. Similar to other architecture specific collective communication algorithms [5, 14], the techniques developed in this paper can be used in advanced communication libraries [3, 4, 10, 25].

3 Network model and problem definition

We consider 1-port homogeneous clusters that consist of a number of workstations connected by switches. Each workstation is connected to the network through one network interface card. A cluster can be either a cut-through cluster or a store-and-forward cluster. We assume that all machines and all links are identical. 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.

The clusters may have arbitrary switch level topologies. We assume that the switch level topology has a spanning tree embedding. Our techniques realize the all-to-all broadcast only on the tree embedding. Since there is only a single path between each pair of nodes in the spanning tree, routing is not an issue. Notice that routing may need to be considered in the construction of the embedded spanning tree before our algorithms are applied.

The spanning tree topology (embedded in the original network) is 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. For a 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.

We focus on the all-gather operation with a sufficiently large message size that the total 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 this time 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.

Some 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 $path(u, v)$ or $path(u \rightarrow v)$ denotes the set of directed edges in the path from node u to node v. The *path length* is defined as the number of switches a message travels through. 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.

3.1 Logical ring based all-to-all broadcast algorithms

Our schemes are based on the logical ring all-to-all broadcast algorithms, which were proposed for single-switch clusters and two-switch clusters [8, 15]. The algorithm works as follows. 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 machine $n_{F(i)}$, $0 \leq i \leq P-1$, sends its own data to machine $n_{F((i+1) \ mod \ P)}$ and receives data from machine $n_{F((i-1) \ mod \ P)}$. In subsequent iterations, each

machine $n_{F(i)}$ forwards what it received in the previous iteration to machine $n_{F((i+1) \ mod \ P)}$ and receives from machine $n_{F((i-1) \ mod \ P)}$. After $P-1$ iterations, all data from all machines reach all machines in the system. Note that in each iteration, each machine must copy the data it receives into the right place of the output buffer.

3.2 Problem definition

All logical ring based all-to-all broadcast algorithms operate in the same fashion. The key for such an algorithm to achieve good performance is to find the logical ring pattern that can carry out communications as efficiently as possible. This is the problem we consider.

Let the slowest communication time in the logical ring pattern be $t_{slowest}$. Since the logical ring pattern is repeated $P-1$ times to realize all-to-all broadcast, 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) $\max_{0 \leq i \leq P-1} \{length(n_{F(i)} \rightarrow n_{F((i+1) \bmod P)})\}$ 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 establish the necessary and sufficient conditions for a store-and-forward cluster to have a contention-free logical ring with a maximum path length of 2. We also consider special cases of Problem 2 where each switch is connected to a small number of switches and develop a polynomial algorithm that finds the optimal solutions for such cases. Such cases are common for clusters with irregular topologies for two reasons: (1) a network with an irregular topology is usually small (consisting of a few switches); and (2) connecting one switch to many other switches without increasing the link speed creates a hot-spot in the network and is usually not a good practice. It must be noted that the topologies in most practical clusters have small diameters. The solution for Problem 1 can be directly applied to such clusters to obtain near-optimal performance.

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, $0 \leq i \leq |S| - 1$, 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 logical ring $n_{0,0} \rightarrow ... \rightarrow n_{0,X_0-1} \rightarrow n_{1,0} \rightarrow ... \rightarrow n_{1,X_1-1} \rightarrow ...n_{|S|-1,0} \rightarrow ... \rightarrow n_{|S|-1,X_{|S|-1}-1} \rightarrow n_{0,0}$ is contention free (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.

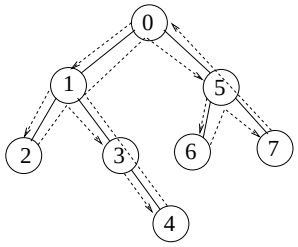


Figure 2: DFS numbering

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: Figure 2 shows an example DFS numbering of the switches. One can easily see that in this example, pattern $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow ... \rightarrow s_7 \rightarrow s_0$ is contention free. Next, 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 a switch s'_i along the path from s_0 to s_{k-1} (excluding s_0 and s_{k-1}), and s_k is a child of s_0, . These three cases are depicted in Figure 3. We use the following facts in the proof of the three cases.

- Fact (a): The link directly connecting switch s_k does not have contention with all communications in the k-switch system, that is, $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_{k-1} \rightarrow s_0$. 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 $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_{k-1}$.

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)).

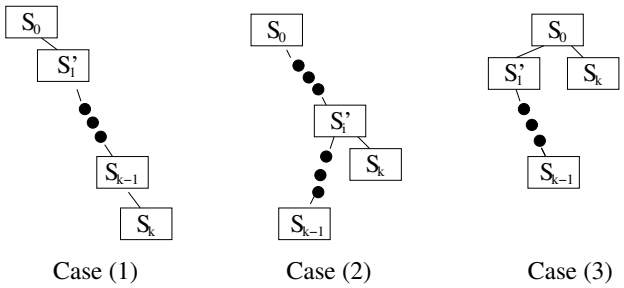


Figure 3: Three cases

- Case (2): Switch s_k is a child of some switch 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 machine 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 for all cases or prove this problem to be NP-complete. Thus, we have to leave the problem open. We make the following two contributions to this problem. First, we identify the sufficient and necessary conditions for a cluster to support a contention free logical ring with a maximum path length of 2. 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 machine that communicates with a machine in another switch. The path length for this communication is at least 2. Hence, this result can be used by a network designer to design a store-and-forward cluster with efficient all-to-all broadcast support. Second, we develop an algorithm that finds optimal solutions for the cases when each switch in the system is connected to a small number of other switches. Here, the term *optimal solutions* means logical rings with the smallest maximum path lengths. A logical ring with a maximum path length of i will be referred to as an i-hop logical ring.

4.2.1 Clusters with 2-hop logical rings

In this sub-section, we will show the sufficient and necessary conditions for a cluster to have a 2-hop logical ring. This result is summarized in the following theorem.

Theorem 2: For a tree graph $G = (S \cup M, E)$, there exists a contention free 2-hop logical

ring 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 contention free 2-hop logical ring, 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 machines, 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 $n_{F(Y-1)} \rightarrow n_{F(0)}$.

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. We prove the sufficient condition by developing a constructive algorithm for finding 2-hop contention free logical ring. 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 to its parent. Let the switch directly connect to X machines, n_0, ..., n_{X-1}. The array pattern for the switch is $n_0 \rightarrow n_1 \rightarrow ... \rightarrow n_{X-1}$ with the first machine n_0 and the last machine n_{X-1}. It can be verified that the three conditions are met.

- Case (2): The switch directly connects to some machines and some switches other than its parent. 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.

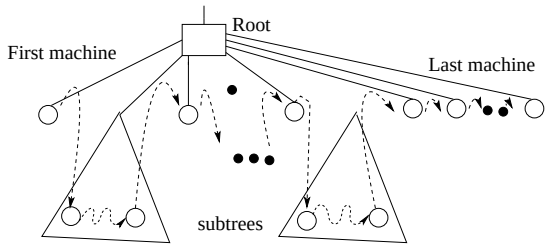
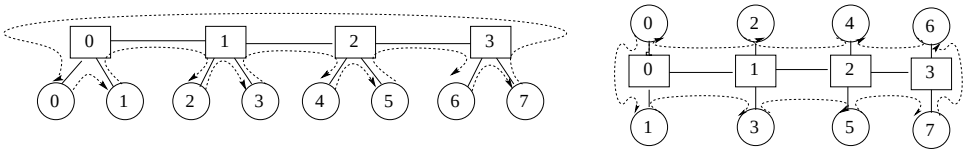


Figure 4: Constructing the logical array for an intermediate switch

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 machine, the last machine, 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 2-hop logical ring can be constructed. □



(a) Logical ring from algorithm 1

(b) Logical ring from algorithm 2

Figure 5: Logical rings from Algorithm 1 and Algorithm 2

The constructive algorithm in the proof of the sufficient condition will be called *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 machine 7 to machine 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.

4.2.2 Finding optimal logical rings

In this sub-section, we describe an algorithm for finding *optimal logical rings*, that is, logical rings with the smallest maximum path lengths. While this algorithm can apply to all topologies, it has a polynomial time complexity only when the number of switches directly connecting to each switch in the system is a small constant. The following lemma provides the foundation for this algorithm.

Lemma 2: Let the P machines in a tree topology $G = (S \cup M, E)$ (rooted at switch R) be numbered as n_0, n_1, ..., n_{P-1}. Let $F : \{0, 1, ..., P-1\} \rightarrow \{0, 1, ..., P-1\}$ be a one-to-one mapping function. The logical ring $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)} \rightarrow n_{F(0)}$ is contention free if and only if for each subtree that contains X machines, there exists a number $0 \leq i \leq P-1$ such that $n_{F(i)}$, $n_{F(i+1 \ mod \ P)}$, ..., $n_{F(i+X-1 \ mod \ P)}$ are machines in the subtree.

Proof: This lemma states that the necessary and sufficient conditions for a logical ring to be contention free is that all machines in each subtree occupy consecutive positions (the consecutive positions can be wrapped around) in the logical ring. Notice that these conditions apply to **all** subtrees in the system assuming any arbitrary switch as the root. Since only the relative positions of the machines in the logical ring will have an impact on the contention free property of the ring, we can assume that the first machine of a top level tree starts at $n_{F(0)}$ in a contention free logical ring without losing generality.

We will first prove the necessary condition by contradiction. Assume that the machines in a subtree are not in consecutive positions in the logical ring, there exist at least two

numbers i and j such that $n_{F(i)}$ and $n_{F(j)}$ are in the subtree while machines $n_{F(i-1 \ mod \ P)}$ and $n_{F(j-1 \ mod \ P)}$ are not in the subtree. Since a communication from a machine outside a subtree to a machine inside a subtree must always use the link connecting the subtree to the rest of the tree, communications $n_{F(i-1 \ mod \ P)} \rightarrow n_{F(i)}$ and $n_{F(j-1 \ mod \ P)} \rightarrow n_{F(j)}$ have contention. This contradicts the assumption that the logical ring is contention free.

To prove the sufficient condition, we will first prove the following claim by induction: Let the P machines in a tree topology rooted at switch R be numbered as n_0, n_1, ..., n_{P-1}. Let $F : \{0, 1, ..., P-1\} \rightarrow \{0, 1, ..., P-1\}$ be a one-to-one mapping function. If the machines in each subtree occupy consecutive positions in the logical array $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)}$, then communications in $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)}$, $R \rightarrow n_{F(0)}$, and $n_{F(P-1)} \rightarrow R$ are contention free. In other words, the logical array $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)}$ is contention free. In addition, the paths from the root to the first machine in the logical array ($n_{F(0)}$) and from last machine in the logical array ($n_{F(P-1)}$) to the root do not have contention with each other and with communications in the logical array.

Base case: It is trivial to show that when there is only a single machine in the system, there is no contention.

Induction case: Consider a tree with n top level subtrees t_0, t_1, ..., t_{n-1}. Since machines in any subtree occupy consecutive positions in the logical array, machines in each subtree t_k, $0 \leq k \leq n-1$ occupy consecutive positions. Let us denote t_k^F, t_k^L, and $T_k = t_k^F \rightarrow ... \rightarrow t_k^L$ be the first machine, the last machine, and the logical array for t_k respectively. Let us denote R_k the root of subtree t_k. Follow the induction hypothesis: communications in $T_k = t_k^F \rightarrow ... \rightarrow t_k^L$, $R_k \rightarrow t_k^F$, and $t_k^L \rightarrow R_k$ are contention free. Let $T_{0'}$, $T_{1'}$, ..., $T_{(n-1)'}$ be a permutation of T_0, T_1, ..., T_{n-1}, where $T_{k'} = t_{k'}^F \rightarrow ... \rightarrow t_{k'}^L$, $0 \leq k \leq n-1$. Since machines in each subtree t_k occupy consecutive positions in the array, we can rewrite the logical array $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)}$ as $t_{1'}^F \rightarrow ... \rightarrow t_{1'}^L \rightarrow t_{2'}^F \rightarrow ... \rightarrow t_{2'}^L \rightarrow ... \rightarrow t_{(n-1)'}^F \rightarrow ... \rightarrow t_{(n-1)'}^L$. Since all subtrees are disjoint, the contentions in the logical array can only be caused by inter-

subtree communications, $t^L_{k'} \rightarrow t^F_{(k+1)'}$, $0 \leq k \leq n-2$. The inter-subtree communication, $t^L_{k'} \rightarrow t^F_{(k+1)'}$, has three components: $t^L_{k'} \rightarrow R_{k'}$, $R_{k'} \rightarrow R \rightarrow R_{(k+1)'}$, and $R_{(k+1)'} \rightarrow t^F_{(k+1)'}$. Since subtree $T_{k'}$ happens once in the logical array, $R_{k'} \rightarrow R \rightarrow R_{(k+1)'}$ will not cause contention. From the induction hypothesis, $t^L_{k'} \rightarrow R_{k'}$ cannot cause contention within subtree $t_{k'}$. Since communications in other subtrees do not use links in $t_{k'}$, $t^L_{k'} \rightarrow R_{k'}$ will not cause contention in the logical array. Similarly, $R_{(k+1)'} \rightarrow t^F_{(k+1)'}$ will not cause contention. Hence, the logical array $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)}$ (or $t^F_{1'} \rightarrow ... \rightarrow t^L_{1'} \rightarrow t^F_{2'} \rightarrow ... \rightarrow t^L_{2'} \rightarrow ... \rightarrow t^F_{(n-1)'} \rightarrow ... \rightarrow t^L_{(n-1)'}$) is contention free. Similar arguments can be applied to show that communications $R \rightarrow t^F_{1'}$ and $t^L_{(n-1)'} \rightarrow R$ do not have contention between each other and do not have contention with $t^F_{1'} \rightarrow ... \rightarrow t^L_{1'} \rightarrow t^F_{2'} \rightarrow ... \rightarrow t^L_{2'} \rightarrow ... \rightarrow t^F_{(n-1)'} \rightarrow ... \rightarrow t^L_{(n-1)'}$. Thus, communications in $t^F_{1'} \rightarrow ... \rightarrow t^L_{1'} \rightarrow t^F_{2'} \rightarrow ... \rightarrow t^L_{2'} \rightarrow ... \rightarrow t^F_{(n-1)'} \rightarrow ... \rightarrow t^L_{(n-1)'}$, $R \rightarrow t^F_{1'}$, and $t^L_{(n-1)'} \rightarrow R$ are contention free. This finishes the proof of the claim.

The path $n_{F(P-1)} \rightarrow n_{F(0)}$ is either equal to or a sub-path of $n_{F(P-1)} \rightarrow R \rightarrow n_{F(0)}$. Hence, $n_{F(P-1)} \rightarrow n_{F(0)}$ does not have contention with $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)}$ and logical ring $n_{F(0)} \rightarrow n_{F(1)} \rightarrow ... \rightarrow n_{F(P-1)} \rightarrow n_{F(0)}$ is contention free. □

Lemma 2 generalizes the results in Algorithm 1 and Algorithm 2, which find two special cases that satisfy the conditions in this lemma. We can see that the optimal logical ring is the concatenation of logical arrays for top-level subtrees. The logical arrays for top-level subtrees are the concatenations of the logical arrays for second level subtrees, and so on. The relation between the optimal logical ring and the logical arrays for subtrees is shown in Figure 6. This relation motivates solving this problem by reducing the optimal logical ring problem into an optimal logical array problem (optimal logical arrays are the arrays with the smallest maximum path length), which has the optimal substructure property: the optimal logical array for a tree contains the optimal logical arrays for its subtrees. Lemma 2 also indicates that we only need to consider logical arrays where machines in each subtree occupy consecutive positions in order to obtain the optimal logical ring. Hence, the dynamic

programming technique can be applied to compute optimal logical arrays for each subtree in a bottom-up fashion.

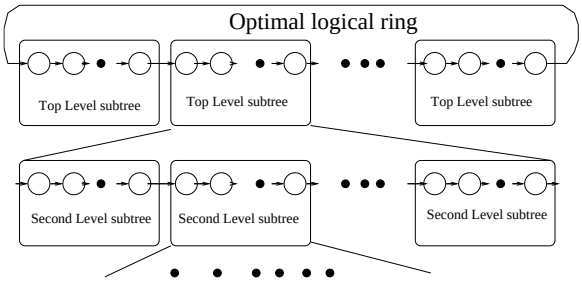


Figure 6: The relationship between the optimal logical ring and logical arrays for subtrees

Let us consider how to reduce the optimal logical ring problem into optimal logical array problems. Let $Opt : \{0, 1, ..., P-1\} \rightarrow \{0, 1, ..., P-1\}$ be a one-to-one mapping function such that $n_{Opt(0)} \rightarrow n_{Opt(1)} \rightarrow ... \rightarrow n_{Opt(P-1)} \rightarrow n_{Opt(0)}$ is an optimal logical ring for a tree topology $G = (S \cup M, E)$ rooted at switch R. Without loss of generality, let us assume that $n_{Opt(0)}$ is the first machine in a top-level subtree. Under the assumption that a switch cannot be a leaf, the root at least has two top-level subtrees. Thus, $n_{Opt(P-1)}$ must be in another top-level subtree and the path $n_{Opt(P-1)} \rightarrow n_{Opt(0)} = n_{Opt(P-1)} \rightarrow R \rightarrow n_{Opt(0)}$. Hence, the optimal logical ring can be considered to have two components: the logical array $n_{Opt(0)} \rightarrow n_{Opt(1)} \rightarrow ... \rightarrow n_{Opt(P-1)}$ and the wrap around link $n_{Opt(P-1)} \rightarrow n_{Opt(0)}$. For a node m, let us use the notation $h(m)$ to denote the height of m, which is defined as the path length from root to m (counting the root as one switch). The height of a node is with respect to a particular subtree (root of the subtree). We have $length(n_{Opt(P-1)} \rightarrow n_{Opt(0)}) = length(n_{Opt(P-1)} \rightarrow R \rightarrow n_{Opt(0)}) = h(n_{Opt(P-1)}) + h(n_{Opt(0)}) - 1$. The length of the wrap around link is a function of the heights of the first machine and the last machine (with respect to the whole tree). Hence, if we can find an optimal logical array $n_{F'(0)} \rightarrow n_{F'(1)} \rightarrow ... \rightarrow n_{F'(P-1)}$ such that $h(n_{F'(0)}) = h(n_{Opt(0)})$ and $h(n_{F'(P-1)}) = h(n_{Opt(P-1)})$, then the maximum path length of logical ring $n_{F'(0)} \rightarrow n_{F'(1)} \rightarrow ... \rightarrow n_{F'(P-1)} \rightarrow n_{F'(0)}$ is less than or equal to the maximum path length of $n_{Opt(0)} \rightarrow n_{Opt(1)} \rightarrow ... \rightarrow n_{Opt(P-1)} \rightarrow n_{Opt(0)}$, and the

logical ring $n_{F'(0)} \rightarrow n_{F'(1)} \rightarrow ... \rightarrow n_{F'(P-1)} \rightarrow n_{F'(0)}$ is also an optimal logical ring. Hence, finding an optimal logical ring can be done by first finding an optimal logical array for each of the possible combinations of $h(n_{Opt(0)})$ and $h(n_{Opt(P-1)})$ and then choosing one that forms a logical ring with the smallest maximum path length. Let the tree height of G be H, the potential values for $h(n_{Opt(0)})$ and $h(n_{Opt(P-1)})$ are in the range of $0..H$.

Next, we will describe the algorithm to determine the maximum path length of the optimal logical ring. By associating each intermediate result in the algorithm with the logical ring/array that yields the result, the algorithm can be modified to obtain the actual optimal logical ring. For each node A (a machine or a switch), we use a two-dimensional table $A.optimal$ to store the maximum path length of the optimal logical arrays for the subtree rooted at A. The entry $A.optimal[i][j]$ stores the maximum path length of the optimal logical array with the height of the first machine being i and the height of the last machine being j. Note that the height is with respect to the subtree rooted at A (the distance from the node to A). Thus, once the *optimal* data structure at the root R, $R.optimal$, is computed, the optimal maximum path length of the logical ring is

$$\min_{i,j}(max(R.optimal[i][j], i + j - 1)).$$

The $R.optimal[i][j]$ is the optimal logical array with the first machine at height i and the last machine at height j, and $i + j - 1$ is the path length of the wrap around link. The term $max(R.optimal[i][j], i + j - 1)$ is the best maximum path length when the ring is formed by having a logical array starting at height i and ending at height j.

Now let us consider how to compute the $A.optimal$ data structure at each node A. As in Algorithm 2, this data structure is computed in a bottom-up fashion (postorder traversal). For each machine A, $A.optimal[0][0] = 0$ and $A.optimal[i][j] = \infty$, $i \neq 0$ or $j \neq 0$. If A is a switch, all subtrees of A have been processed. Let A have n subtrees t_0, t_2, ..., t_{n-1}. Let us assume that among the n subtrees, k are rooted at switches (each of the subtrees is rooted at a switch). The rest $n - k$ are single-machine subtrees (each of the subtrees contains only

a single machine). The algorithm first enumerates all possible different sequences of the n subtrees. Since all machines are the same, switching their positions in the sequence yields the same sequence. Hence, there are at most k-permutation of an n-set (selecting k positions for the k subtrees rooted at switches from the n possible positions in the sequence), that is, $n(n-1)...(n-k+1) = \frac{n!}{(n-k)!} = O(n^k)$, different sequences. Here, $n! = n \times (n-1) \times ... \times 1$.

For each sequence $seq = t_{0'}t_{1'} \ ... \ t_{(n-1)'}$, we compute the *seq.optimal* data structure for the case when the subtrees are concatenated in the particular order $t_{0'} \rightarrow t_{1'} \rightarrow ... \rightarrow t_{(n-1)'}$. There are three cases. First, if *seq* only has one subtree $t_{0'}$, then $seq.optimal[i][j] = t_{0'}.optimal[i][j]$. Second, if *seq* contains two subtrees $t_{0'}$ and $t_{1'}$, the optimal data structure for the sequence is computed as follows:

$$seq.optimal[i][j] = \min_{k,l}\{max(t_{0'}.optimal[i][k], t_{1'}.optimal[l][j], k+l+1)\}.$$

To find the optimal logical array $t_{0'} \rightarrow t_{1'}$ that starts at at height i and ends at height j, the array in $t_{0'}$ must start at height i and the array in $t_{1'}$ must end at height j. However, the array in $t_{0'}$ can end at any position and the array in $t_{1'}$ can start at any position. For a logical array that is composed of the array in $t_{0'}$ that starts at height i and ends at height k and the array in $t_{1'}$ that starts at height l and ends at height j, the maximum path length is the maximum of three elements: the maximum path length of the array in $t_{0'}$ ($t_{0'}.optimal[i][k]$), the maximum path length of the array in $t_{1'}$ ($t_{1'}.optimal[l][j]$), and the length of $t^L_{0'} \rightarrow t^F_{1'} = t^L_{0'} \rightarrow A \rightarrow t^F_{1'}$, which is equal to $k+l+1$. This formula takes into account all possible combinations to concatenate the two subtrees. Third, if *seq* contains more than two subtrees, the *seq.optimal* data structure can be obtained by repeatedly applying the concatenation of two logical arrays (the second case).

The optimal data structure for A can be obtained from the optimal data structures for all possible sequences using the following formula:

$$A.optimal[i][j] = \min_{seq \ is \ a \ sequence}\{seq.optimal[i-1][j-1]\}.$$

This formula examines all possible sequences to determine the optimal logical array for a given i and j. Notice that, for a node m, when $h(m) = i$ in subtree rooted at A, $h(m) = i-1$ in the subtrees of A. Intuitively, this algorithm examines all possible cases when all machines in each subtree must be placed in consecutive positions in the logical ring and stores the optimal results at the root of the subtree. By induction, it can be formally shown that this algorithm finds the maximum path length of the optimal logical ring for the tree.

This algorithm, which will be called Algorithm 3, operates in a similar fashion to Algorithm 2. The difference is that only one optimal logical array for each subtree must be considered under the assumptions for Algorithm 2 to obtain the optimal logical ring for the whole tree. Without those assumptions, we must compute and store many optimal logical arrays for different heights of the first and last machines. In addition, the process to determine the optimal logical arrays becomes more complex.

Let us now examine the complexity of this algorithm. Let the number of nodes be $|V|$, the maximum nodal degree be n (n usually equals to the maximum number of ports in a switch), the maximum number of switches directly connecting to one switch be k, the tree height be H. The size of the table to store the optimal logical arrays is $O(H^2)$. The time to concatenate two logical arrays is $O(H^4)$. Since a node can have at most a sequence of size n, computing the optimal data structure for one sequence is $O(nH^4)$. Given that each node can have at most $O(n^k)$ sequences, the time to process each node is then $O(n^k nH^4)$. Thus, the complexity of the whole algorithm is $O(|V|n^{k+1}H^4)$. When k is a small constant, this algorithm has a polynomial time complexity. In practice, while $|V|$ can be a large number (e.g. a cluster of a few thousand nodes), the values of n, k, H are usually small.

5 Experiments

We used two approaches to evaluate the proposed algorithms. First, we develop an automatic routine generator that takes the topology information as input and generates, based on

the algorithms presented, customized topology-aware all–gather routines. The performance of the generated routines on multiple topologies is compared with that of the all–gather routines in LAM/MPI 7.1.1 [13] and the recently improved MPICH 2-1.0.1 [15]. LAM/MPI and MPICH contain well known and efficient topology unaware all-gather algorithms. The experiments are performed on a 32-machine Ethernet switched cluster. Second, we study the performance of the algorithms on large networks through simulation. Note that although we evaluate our schemes with an Ethernet switched cluster, the proposed algorithms can be applied to other types of networks such as InfiniBand and Myrinet. In particular, for clusters with irregular topologies, the topology specific algorithms should be much more efficient than existing topology unaware algorithms since network contention has a larger impact when the network speed is faster.

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

Figure 7: Code segment for measuring MPI_Allgather performance.

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, LR2, and LR3 to denote the routines obtained from Algorithm 1 (finding a contention free logical ring), Algorithm 2 (finding a contention free 2-hop logical ring), and Algorithm 3 (finding an optimal contention free logical ring) respectively. Note that in cases when LR2 can find 2-hop rings, LR3 can also find 2-hop rings. To report a fair performance comparison, we port the MPICH all–gather implementation to LAM/MPI. We use MPICH-LAM to represent the ported all–gather routine. In the recent versions of MPICH, the performance of the all–gather operation using native MPICH and MPICH-LAM is very close. In the performance evaluation, LR1, LR2, and LR3 are compared with LAM and MPICH-LAM. We use the approach similar to Mpptest [7] to measure

the performance of the $MPI\_Allgather$ routines. Figure 7 shows the 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 machines as the performance metric.

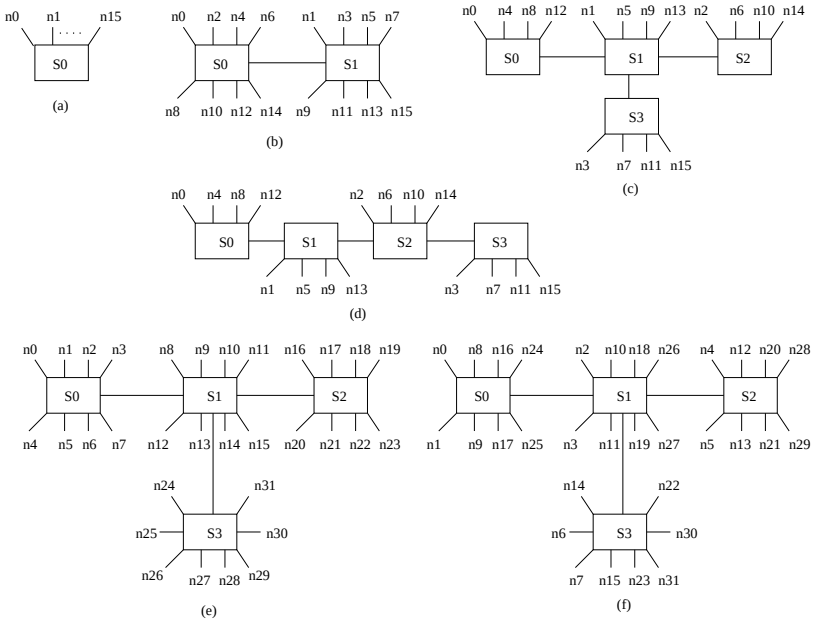
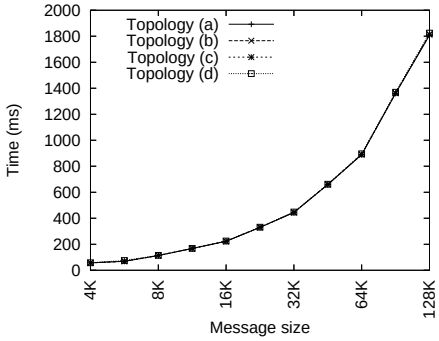


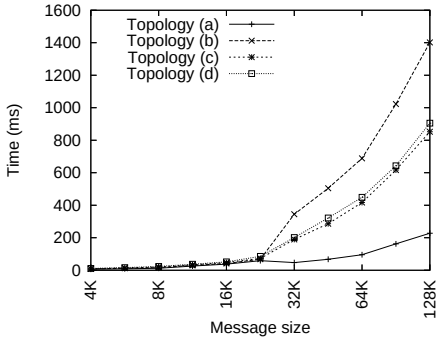
Figure 8: Topologies used in the experiments

The experiments are performed on a 32-machine Ethernet switched cluster. The machines of the cluster are Dell Dimension 2400 with a 2.8GHz P4 processor, 128MB of memory, and 40GB 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 8 shows the topologies we used in the experiments. Parts (a) to (d) of the figure represent clusters of 16 machines connected by 1, 2, and 4 switches with different topologies. Parts (e) and (f) show 32-machine clusters of 4 switches, each having 8 machines attached. These two clusters have exactly the same physical topology,

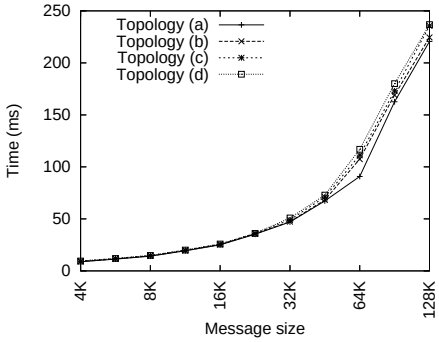
but different node assignments. We will refer to the topologies in the figure as topology (a), topology (b), topology (c), topology (d), topology (e), and topology (f).



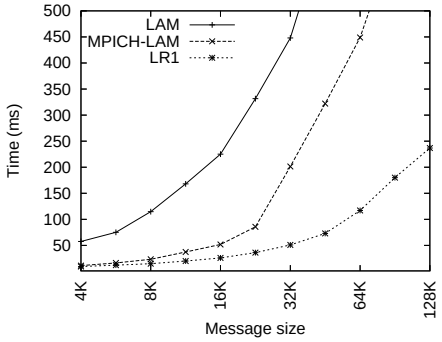
(a) LAM



(b) MPICH-LAM



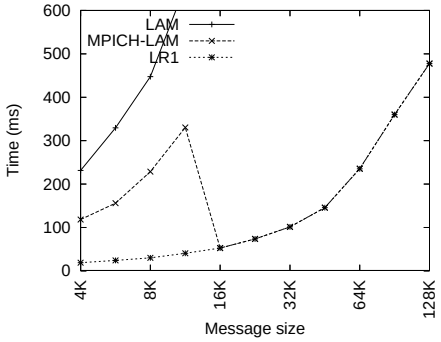
(c) LR1



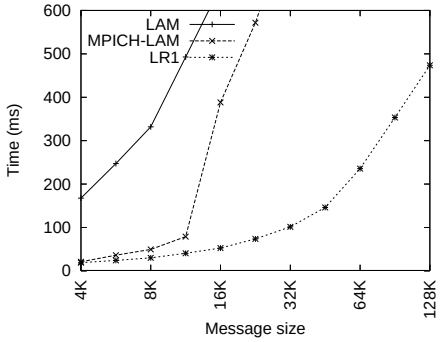
(d) Results for topology (d)

Figure 9: Performance of LR1, LAM, and MPICH-LAM on topologies (a) to (d)

Figure 9 shows the performance of LAM, MPICH-LAM, and LR1 on topologies (a) to (d). Note that the time scales are different in different figures. Figure 9 (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 LAM 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 9 (d), the LAM/MPI routine performs much worse than MPICH and LR1. The results for MPICH are shown in Figure 9 (b). MPICH changes the algorithm when $msize = 32KB$. When $msize < 32KB$, MPICH uses the recursive doubling



(a) Results for topology (e)



(b) Results for Topology (f)

Figure 10: Performance of LR1, LAM, and MPICH-LAM on topologies (e) and (f).

algorithm, which has similar performance for all topologies. Using a topology-unaware logical ring algorithm when $msize \geq 32KB$, MPICH provides very different performance for the four topologies. It performs best on topology (a), where the cluster is connected by a single switch, but significantly worse on topologies (b), (c), and (d), which indicates that the network topology can significantly affect MPICH performance. From Figure 9 (c), 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 near-optimal all-to-all broadcast performance on different topologies. Figure 9 (d) compares LR1 with LAM and MPICH on topology (d). It is shown that LR1 performs substantially better than LAM and MPICH.

Figures 10 (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 10 (a), the topology-unaware ring algorithm used in MPICH, when $msize \geq 16KB$, achieves near-optimal performance (same as LR1) for this topology. In this case, the topology-unaware ring algorithm operates exactly the same as LR1. However, with the same physical topology and a different node assignment in topology (f), the

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)

topology-unaware algorithm performs much worse than LR1 as shown in Figure 10 (b). This again shows that the performance of MPICH depends heavily on the network configuration. Unlike LAM and MPICH, LR1 consistently achieves high performance for different topologies. 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.

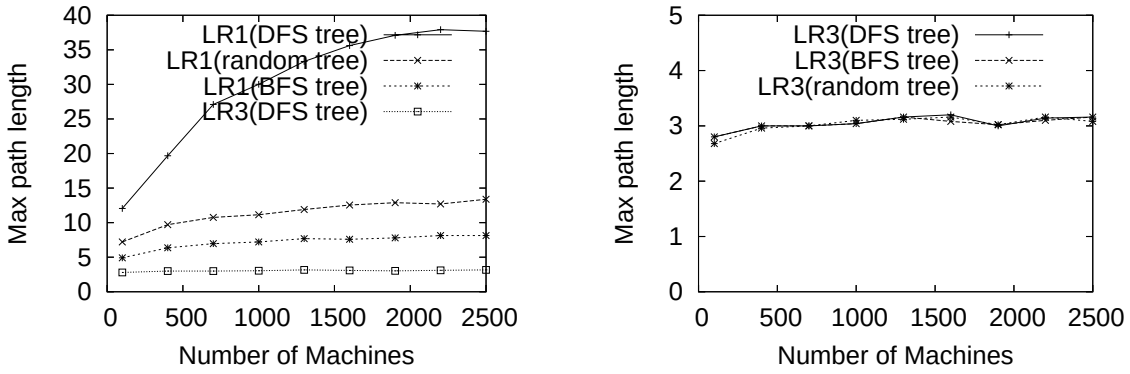
Table 1 shows the impact of selecting a logical ring with a shorter path length. For topology (d), LR1 results in a logical ring with a maximum path length of 4 hops, and LR2 results in a logical ring with a maximum path length of 2 hops. In addition to the results of LR1 and LR2, the table also includes 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-machine 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.

Next, we will study how these algorithms perform on large store-and-forward clusters. This study is performed through simulation. In the simulation, we assume that the networks have arbitrary topologies. This allows us to evaluate the impacts of tree construction methods and to compare the results of different methods in computing logical rings. Since the logical rings found by all of our algorithms are contention free, we use the maximum path length as the performance metric.

In the simulation, we first generate each random cluster. After the random cluster is generated, a tree construction method is used to build a spanning tree for the random cluster. Finally, the proposed algorithms are applied to compute logical rings. A random cluster is generated as follows. First, we decide the number of machines and the number of switches for the cluster. In the experiments, we fix the ratio between the number of machines and the number of switches to be 5:1 (on average, each switch has five machines). The random connectivity among switches is generated using the *Georgia Tech Internetwork Topology Models (GT-ITM)* [26] with an average nodal degree of 4. Once the topology for the switches is generated, each machine is randomly distributed to any switch with an equal probability. We consider three tree construction methods: Breadth First Search (BFS), Depth First Search (DFS), and random. The BFS tree is created by first randomly selecting a root and then performing BFS on the graph. The DFS tree is created by first randomly selecting a root and then performing DFS. The random tree is created by repeatedly adding a random link to form a tree (if adding a link forms a loop, the link is not added).

Figure 11 shows the maximum path lengths in the logical rings computed using LR1 and LR3 with the three tree construction methods. Each point in the figure is the average of 50 random topologies. Figure 11 (a) shows that (1) the maximum path lengths of the logical rings computed by LR1 is much larger than those computed by LR3, and (2) the performance of LR1 depends heavily on the method used to construct the spanning tree. Using BFS trees



(a) (b)

Figure 11: Performance of LR1 and LR3 with different tree construction methods

yields much better results than using random trees, which in turn has much better results than DFS trees. This is because BFS trees usually have small tree heights while DFS trees are usually tall. Figure 11 (b) shows that LR3 produces much smaller maximum path lengths in its rings. In all experiments that we performed on different kinds of random graphs, there is a very high probability that the maximum path length of each ring produced by LR3 is 3. This is reflected in Figure 11 (b): the average maximum path length is around 3, regardless of the cluster sizes. Furthermore, LR3 is not sensitive to the tree construction methods. BFS trees, DFS trees, and random trees yield very similar results. This indicates that LR3 is a robust algorithm for large networks.

6 Conclusion

In this paper, we develop bandwidth efficient all-to-all broadcast schemes that can be applied uniformly to systems with regular, irregular, and hierarchical topologies. Using the proposed schemes, a cut-through cluster with any topology that has a spanning 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. Performance evaluation results indicate that these algorithms are robust for both large and small networks, and that they can provide much better performance than traditional topology unaware all-to-all broadcast algorithms on switched clusters.

Acknowledgement

This research is supported in part by National Science Foundation (NSF) under grants CCF-0342540, CCF-0541096, and CCF-0551555. Experiments are also performed on resources sponsored through an NSF Teragrid grant CCF-050010T.

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):1143–1156, November 1997.

[3] A. Faraj and X. Yuan, "Automatic Generation and Tuning of MPI Collective Communication Routines," The *19th ACM International Conference on Supercomputing* (ICS'05), pages 393-402, Cambridge, MA, June 20-22, 2005.

[4] A. Faraj, X. Yuan, and D.K. Lowenthal, "STAR-MPI: Self Tuned Adaptive Routines for MPI Collective Operations," the *20th ACM International Conference on Supercomputing*, pages 199-208, Cairns, Australia, June, 2006.

[5] A. Faraj, X. Yuan, and Pitch Patarasuk, "A Message Scheduling Scheme for All-to-all Personalized Communication on Ethernet Switched Clusters," *IEEE Transactions on Parallel and Distributed Systems*, 18(2):264-276, Feb. 2007.

[6] M. Golebiewski, R. Hempel, and J. L. Traff, "Algorithms for Collective Communication Operations on SMP Clusters," *In the 1999 Workshop on Cluster-Based Computing*, page 1115, June 1999.

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

[8] 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, pages 325-329, April 1999.

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

[10] A. Karwande, X. Yuan, and D. K. Lowenthal, "An MPI Prototype for Compiled Communication on Ethernet Switched Clusters," *Journal of Parallel and Distributed Computing*, 65(10):1123-1133, October 2005.

[11] 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.

[12] 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.

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

[14] R. G. Lane, S. Daniels and X. Yuan, "An Empirical Study of Reliable Multicast Protocols over Ethernet-Connected Networks," *Performance Evaluation Journal*, 64(3):210-228, March 2007.

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

[16] 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.

[17] 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)*, pages 65-70, 2002.

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

[19] A. Tam and C. Wang, "Efficient Scheduling of Complete Exchange on Clusters," *the ISCA 13th International Conference on Parallel and Distributed Computing Systems*, pages 111-116, August 2000.

[20] R. Thakur and A. Choudhary, "All-to-all Communication on Meshes with Wormhole Routing," *8th International Parallel Processing Symposium (IPPS)*, pages 561-565, April 1994.

[21] 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.

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

[23] 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.

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

[25] Xin Yuan, Rami Melhem and Rajiv Gupta, "Algorithms for Supporting Compiled Communication," *IEEE Transactions on Parallel and Distributed Systems*, 14(2):107-118, February 2003.

[26] E. W. Zegura, K. Calvert and S. Bhattacharjee. "How to Model an Internetwork." *IEEE Infocom '96*, pages 594-602, April 1996.