Pipelined Broadcast on Ethernet Switched Clusters [*]

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

Abstract

We consider unicast-based pipelined broadcast schemes for clusters connected by multiple Ethernet switches. By splitting a large broadcast message into segments and broadcasting the segments in a pipelined fashion, pipelined broadcast may achieve very high performance. We develop algorithms for computing various contention-free broadcast trees on Ethernet switched clusters that are suitable for pipelined broadcast, and evaluate the schemes through experimentation. The conclusions drawn from our theoretical and experimental study include the following. First, pipelined broadcast can be more effective than other common broadcast schemes including the ones used in the latest versions of MPICH and LAM/MPI when the message size is sufficiently large. Second, contention-free broadcast trees are essential for pipelined broadcast to achieve high performance. Finally, while it is difficult to determine the optimal message segment size for pipelined broadcast, finding one size that gives good performance is relatively easy.

1 Introduction

Switched Ethernet is the most widely used local–area–network (LAN) technology. Many Ethernet switched clusters of workstations are used to perform high performance computing. For such clusters to be effective, communications must be carried out as efficiently as possible.

Broadcast is one of the most common collective communication operations. The broadcast operation requires a message from the *root* machine (the sender) to reach all other machines in the system at the end of the operation. The Message Passing Interface routine that realizes this operation is *MPI_Bcast* [14]. Broadcast algorithms are typically classified as either *atomic broadcast* algorithms or *pipelined broadcast* algorithms [1]. Atomic broadcast algorithms distribute the broadcast message as a whole through the network. Such algorithms apply to the cases when there is only one broadcast operation and the broadcast message cannot be split. When there are multiple broadcast operations or when the broadcast message can be split into a number of segments, a pipelined broadcast algorithm can be used, which distributes messages (segments) in a pipelined fashion.

In this paper, we investigate the use of pipelined broadcast to realize *MPI_Bcast* on Ethernet switched clusters when the message size is reasonably large. In this case, broadcasting a large message is carried out by a sequence of pipelined broadcasts with smaller message segments. In the pipelined broadcasts, communications on different branches of the logical broadcast tree can be active at the same time. To maximize the performance, communications that can potentially happen simultaneously should not share the same physical channel and cause network contention. Hence, the logical broadcast trees for pipelined broadcast should be contention-free. We develop algorithms for computing various contention-free broadcast trees that are suitable for pipelined broadcast on Ethernet switched clusters. We theoretically analyze and empirically evaluate pipelined broadcast schemes by comparing their performance with that of other commonly used broadcast algorithms. The results from our theoretical and experimental study indicate the following.

- When the message size is large, pipelined broadcast can be more effective than other broadcast schemes including the ones used in MPICH 2-1.0.1 (the latest MPICH release) [15] and LAM/MPI 7.1.1 (the latest LAM/MPI release) [12] to a large degree. MPICH and LAM/MPI are two widely used open source MPI implementations. Moreover, for large messages, the performance of pipelined broadcast on Ethernet switched clusters using a contention-free linear tree is close to the theoretical limit of the broadcast operation.

[*] This work is partially supported by NSF grants CCR-0208892 and CCF-0342540.

1-4244-0054-6/06/$20.00 ©2006 IEEE

- Contention-free broadcast trees are essential for pipelined broadcast to achieve high performance on clusters with multiple switches. Pipelined broadcast using topology unaware broadcast trees may result in poor performance in such an environment.
- While it is difficult to determine the message segment size for pipelined broadcast to achieve the minimum communication time, finding one segment size that gives good performance is relatively easy since a wide range of message sizes can yield reasonably good performance for a given pipelined broadcast algorithm.

The rest of the paper is organized as follows. The related work is discussed in Section 2. The network model is described and a number of commonly used broadcast algorithms are analyzed in Section 3. Section 4 details the algorithms for computing various contention-free broadcast trees on Ethernet switched clusters. Section 5 reports the results of our experimental evaluation. Finally, Section 6 concludes the paper.

2 Related Work

The broadcast operation in different environments has been extensively studied and a very large number of broadcast algorithms have been proposed. Algorithms developed for topologies used in parallel computers such as meshes and hypercubes (see for example [8, 11]) are specific to the targeted topologies and platforms and cannot be applied to Ethernet switched clusters with physical tree topologies. Many researchers proposed to use logical binomial trees for the broadcast operation and developed algorithms for computing contention-free binomial trees under different constraints [6, 10, 13]. Using the binomial tree, the broadcast operation is partitioned into $log_2(P)$ phases, where P is the number of machines in the system. The contention-free binomial trees only require the communications in each phase to be contention free. Thus, such algorithms cannot be used to compute contention-free trees for pipelined broadcast. Atomic broadcast algorithms over physical tree topologies have also been developed [3, 16]. Such algorithms are different from pipelined broadcast algorithms.

More related to this work are the studies of pipelined broadcast in different environments [1, 2, 7, 17, 19, 20]. In [20], an algorithm was designed to compute contention-free pipelined trees on the mesh topology. In [1, 2], heuristics for pipelined communication on heterogeneous clusters were devised. These heuristics focus on the heterogeneity of links and nodes, but not the network contention issue. The effectiveness of pipelined broadcast in cluster environments was demonstrated in [7, 17, 19]. It was shown that pipelined broadcast using topology unaware trees can be very efficient for clusters connected by a single switch. Our research extends the work in [7, 17, 19] by considering clusters connected by multiple switches. As shown in the performance study, pipelined broadcast using topology unaware trees in such an environment may yield extremely poor performance. To the best of our knowledge, methods for building contention-free trees for pipelined broadcast over a physical tree topology have not been developed and studied. The techniques described in this paper fill this void.

3 Network Model

We consider Ethernet switched clusters where each workstation is equipped with one Ethernet port and links operate in the duplex mode that supports simultaneous communications on both directions of each link with the full bandwidth. Communications in such a system follow the 1-port model [2], that is, at one time, a machine can send and receive one message. The switches may be connected in an arbitrary way. However, a spanning tree algorithm is used by the switches to determine forwarding paths that follow a tree structure [18]. As a result, the physical topology of the network is always a **tree** with switches being the internal nodes and machines being leaves. While hardware broadcast is supported in Ethernet, using such a technology to realize *MPI_Bcast* requires the implementation of a reliable multicast protocol [9], which is complex. In this paper, we focus on unicast-based pipelined broadcast.

The 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. Since the network topology is a tree, the graph is also a tree: there is a unique path between any two nodes. Figure 1 shows an example cluster. We assume that all links have the same bandwidth.

Notion $u \rightarrow v$ denotes a communication from node u to node v. $Path(u \rightarrow v)$ denotes the set of directed edges in the unique path from node u to node v. For example, in Figure 1, $path(n0 \rightarrow n3) = \{(n0, s0), (s0, s1), (s1, s3), (s3, n3)\}$. Two communications, $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$, are said to have contention if they share a common edge, that is, there exists an edge (x, y) such that $(x, y) \in path(u_1 \rightarrow v_1)$ and $(x, y) \in path(u_2 \rightarrow v_2)$. A *pattern* is a set of commu-

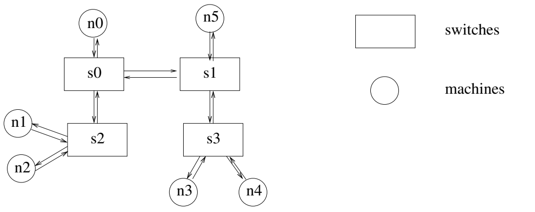


Figure 1. An example Cluster

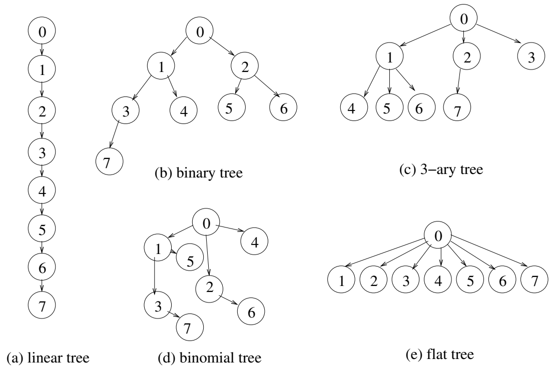
nications. A *contention-free pattern* is a pattern where no two communications in the pattern have contention. We will use the notion $u \rightarrow v \rightarrow w \rightarrow ... \rightarrow x \rightarrow y \rightarrow z$ to represent pattern $\{u \rightarrow v, v \rightarrow w, ..., x \rightarrow y, y \rightarrow z\}$.

3.1 Broadcast on Ethernet Switched Clusters

Let the broadcast message size be $msize$ and the number of machines in the broadcast operation be P. We will assume that the time taken to send a message of size n between any two machines can be modeled as $T(n) = \alpha + n \times \beta$, where α is the startup overhead and β is the per byte transmission time. When an $msize$-byte message is split into segments of sizes s_1, s_2, ..., and s_k, $T(s_1) + T(s_2) + ... + T(s_k) \geq T(msize)$. Splitting a large message into small segments will increase the startup overheads and thus, the total communication time. Under the assumption that the startup overhead is insignificant in $T(s_i)$, $1 \leq i \leq k$, $T(s_1) + T(s_2) + ... + T(s_k) \approx T(msize)$.

Let the *communication completion time* be the duration between the time when the root starts sending and the time when the last machine receives the whole message. In the broadcast operation, each machine receives $msize$ data and the lower bound of the completion time is at least $T(msize)$. We will show later that this lower bound is approached by pipelined broadcast when $msize$ is sufficiently large.

Figure 2 shows some common broadcast trees, including linear tree, binary tree, k-ary tree, binomial tree, and flat tree. Common atomic broadcast algorithms include the flat tree and binomial tree algorithms. In the flat tree algorithm, the root sequentially sends the broadcast message to each of the receivers. The completion time is thus $(P-1) \times T(msize)$. In the binomial tree algorithm[6, 13], broadcast follows a hypercube communication pattern and the total number of messages that the root sends is $log(P)$. Hence, the completion time is $log(P) \times T(msize)$. In both of the flat tree and binomial tree algorithms, the root is busy throughout the communication and pipelined communication cannot be used to improve performance. Another interesting non-pipelined broadcast algorithm is the *scatter followed by all-gather* algorithm, which is used in MPICH [15]. In this algorithm, the $msize$-byte message is first distributed to the P machines by a scatter operation (each machine gets $\frac{msize}{P}$-byte data). After that, an all-gather operation is performed to combine messages to all nodes. In the scatter operation, $\frac{P-1}{P} \times msize$ data must be moved from the root to other nodes, and the time is at least $T(\frac{P-1}{P} \times msize)$. In the all-gather operation, each node must receive $\frac{P-1}{P} \times msize$-byte data from other nodes and the time is at least $T(\frac{P-1}{P} \times msize)$. Hence, the completion time for the whole algorithm is at least $2 \times T(\frac{P-1}{P} \times msize) \approx 2 \times T(msize)$.



(b) binary tree

(c) 3−ary tree

(a) linear tree

(d) binomial tree

(e) flat tree

Figure 2. Examples of broadcast trees

Now, let us consider pipelined broadcast. Assume that the $msize$-byte broadcast message is split into X segments of size $\frac{msize}{X}$, broadcasting the $msize$-byte message is realized by X pipelined broadcasts of segments of size $\frac{msize}{X}$. To achieve good performance, the segment size, $\frac{msize}{X}$, should be small while keeping the startup overhead insignificant in $T(\frac{msize}{X})$. For example, in our experimental cluster, a segment size of $1KB$ results in good performance in most cases. Hence, when $msize$ is very large, X can be large.

The completion time for the X pipelined broadcasts depends on the broadcast tree, which decides the size of each pipeline stage and the number of pipelined stages. For simplicity, we will assume in this section that there is no network contention in pipelined broadcast. Under the 1-port model, the size of a pipeline stage is equal to the time to send the number of messages that a machine must send in that stage, which is equal to the nodal degree of the machine in the broadcast tree. The number of pipelined stages is equal to the tree height. Let the broadcast tree height be H and the maximum nodal degree of the broadcast tree be D. The largest pipeline stage is $D \times T(\frac{msize}{X})$. The total time to complete the communication is roughly

$$(X + H - 1) \times (D \times T(\frac{msize}{X})).$$

When $msize$ is very large, X will be much larger than $H-1$. In this case, $(X+H-1)(D \times T(\frac{msize}{X})) \approx X \times (D \times T(\frac{msize}{X})) \approx D \times T(msize)$. This simple analysis shows that for large messages, trees with a small nodal degree should be used. For example, using a linear tree, shown in Figure 2 (a), $H = P$ and $D = 1$. The communication completion time is $(X + P - 1) \times T(\frac{msize}{X})$. When X is much larger than P, $(X+P-1) \times T(\frac{msize}{X}) \approx T(msize)$, which is the theoretical limit of the broadcast operation.

Using the linear tree, the number of pipelined stages is P, which results in a long time to drain the pipeline when P is large. To reduce the number of pipelined stages, a general k-ary tree, that is, a tree with a maximum nodal degree of k, can be used. When $k = 2$, we call such trees binary trees. Assuming a complete binary tree is used, $H = log_2(P)$ and $D = 2$. The completion time is $(X+log_2(P)-1) \times 2 \times T(\frac{msize}{X}) = (2X + 2log_2(P)-2) \times T(\frac{msize}{X})$. When X is sufficiently large, $(2X + 2log_2(P) - 2) \times T(\frac{msize}{X}) \approx 2T(msize)$. When broadcasting a very large message, pipelined broadcast with a binary tree is not as efficient as that with a linear tree. However, when $2X + 2log_2(P) - 2 \leq X + P - 1$ or $X \leq P - 2log_2(P) + 1$, the binary tree is more efficient. In other words, when broadcasting a medium sized message, a binary tree may be more efficient than a linear tree. Under the 1-port model, when using general k-ary trees, $k > 2$, for pipelined broadcast, the size of the pipelined stage increases linearly with k while the tree height decreases proportionally to the reciprocal of the logarithm of k, assuming that trees are reasonably balanced such that the tree height is $O(log_k(P))$. Hence, under the 1-port model, it is unlikely that a k-ary tree, $k > 2$, can offer much better performance than a binary tree. For example, assuming a complete k-ary tree, $k > 2$, is used for pipelined broadcast, $H = log_k(P)$ and $D = k$. The completion time is $(X + log_k(P) - 1) \times (k \times T(\frac{msize}{X}))$, which is larger than the time for the complete binary tree for most practical values of X and P. Our empirical study confirms this. Table 1 summarizes the performance of broadcast algorithms when the broadcast message size is very large.

Algorithm	performance
Flat tree	$(P-1) \times T(msize)$
Binomial tree	$log_2(P) \times T(msize)$
scatter/allgather	$2 \times T(msize)$
Linear tree (pipelined)	$T(msize)$
Binary tree (pipelined)	$2 \times T(msize)$
k-ary tree (pipelined)	$k \times T(msize)$

Table 1. The performance of broadcast algorithms for very large messages

4 Computing contention-free broadcast trees

As shown in Table 1, pipelined broadcast is likely to achieve high performance when the broadcast message is large. There are two obstacles that prevent this technique from being widely deployed. First, for pipelined broadcast to be effective, the logical broadcast tree must be contention-free. Finding a contention-free broadcast tree is a challenging task. Second, it is difficult to decide the optimal segment size for pipelined broadcast. In this section, we will present algorithms for computing contention-free broadcast trees over physical tree topologies. In the next section, we will show that while deciding the optimal segment size may be difficult, it is relatively easy to find one size that achieves good performance.

Under the 1-port model, communications originated from the same machine cannot happen at the same time. Thus, a contention-free tree for pipelined broadcast only requires communications originated from different machines to be contention-free. Since each communication in a linear tree originates from a different machine, all communications in the contention-free linear tree must be contention-free. In a contention-free k-ary tree, communications from a machine to its (up to k) children may have contention.

4.1 Contention-free linear trees

Let the machines in the system be n_0, n_1, ..., n_{P-1}. Let $F : \{0, 1, ..., P-1\} \rightarrow \{0, 1, ..., P-1\}$ be any one-to-one mapping function such that $n_{F(0)}$ is the root of the broadcast operation. $n_{F(0)}$, $n_{F(1)}$, ..., $n_{F(P-1)}$ is a permutation of n_0, n_1, ..., n_{P-1} and $n_{F(0)} \rightarrow n_{F(1)} \rightarrow n_{F(2)} \rightarrow ... \rightarrow n_{F(P-1)}$ is a logical linear tree. The task is to find an F such that the communications in the logical linear tree do not have contention.

Let $G = (S \cup M, E)$ be a tree graph with S being the switches, M being the machines, and E being the edges. $P = |M|$. Let n_r be the root machine of the broadcast. Let $G' = (S, E')$ be a subgraph of G that only contains switches and links between switches. A contention-free linear tree can be computed in the following two steps.

- Step 1: Starting from the switch that n_r is directly connected to, perform Depth First Search (DFS) on G'. Number the switches based on the DFS arrival order. An example numbering of the switches in the DFS order is shown in Figure 3. We will denote the switches as s_0, s_1, ..., $s_{|S|-1}$, where s_i is the ith switch arrived in the DFS traversal of G'. The switch that n_r attaches to is s_0.

- 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}. $n_r = n_{0,0}$. $X_i = 0$ when there is no machine attaching to s_i. The following logical linear tree is contention-free (we will formally prove this): $n_{0,0}(n_r) \rightarrow ... \rightarrow n_{0,X_0-1} \rightarrow n_{1,0} \rightarrow ... \rightarrow n_{1,X_1-1} \rightarrow ... \rightarrow n_{|S|-1,0} \rightarrow ... \rightarrow n_{|S|-1,X_{|S|-1}-1}$.

We will refer to this algorithm as *Algorithm 1*. There exist many contention-free logical linear trees for a physical tree topology. We will prove that *Algorithm 1* computes one of the contention-free logical linear trees.
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 $\{s_0 \rightarrow s_1, s_1 \rightarrow s_2, ..., s_{|S|-2} \rightarrow s_{|S|-1}, s_{|S|-1} \rightarrow s_0\}$ are contention free. □

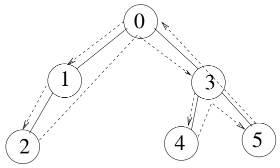


Figure 3. DFS numbering

The proof of Lemma 1 can be found in [5]. Figure 3 shows an example. Clearly, communications in $\{s_0 \rightarrow s_1, s_1 \rightarrow s_2, s_2 \rightarrow s_3, s_3 \rightarrow s_4, s_4 \rightarrow s_5, s_5 \rightarrow s_0\}$ are contention-free.
Lemma 2: Let s_0, s_1, ..., $s_{|S|-1}$ be the DFS ordering of the switches. Let $0 \leq i < j \leq k < l \leq |S| - 1$, $s_i \rightarrow s_j$ does not have contention with $s_k \rightarrow s_l$.
Proof: From Lemma 1, $path(s_i \rightarrow s_{i+1})$, $path(s_{i+1} \rightarrow s_{i+2})$, ..., $path(s_{j-1} \rightarrow s_j)$, $path(s_k \rightarrow s_{k+1})$, $path(s_{k+1} \rightarrow s_{k+2})$, ..., $path(s_{l-1} \rightarrow s_l)$ do not share any edge. It follows that $path(s_i \rightarrow s_{i+1}) \cup path(s_{i+1} \rightarrow s_{i+2}) \cup ... \cup path(s_{j-1} \rightarrow s_j)$ does not share any edge with $path(s_k \rightarrow s_{k+1}) \cup path(s_{k+1} \rightarrow s_{k+2}) \cup ... \cup path(s_{l-1} \rightarrow s_l)$. Since the graph is a tree, $path(s_i \rightarrow s_j) \subseteq path(s_i \rightarrow s_{i+1}) \cup path(s_{i+1} \rightarrow s_{i+2}) \cup ... \cup path(s_{j-1} \rightarrow s_j)$ and $path(s_k \rightarrow s_l) \subseteq path(s_k \rightarrow s_{k+1}) \cup path(s_{k+1} \rightarrow s_{k+2}) \cup ... \cup path(s_{l-1} \rightarrow s_l)$. Thus, $s_i \rightarrow s_j$ does not have contention with $s_k \rightarrow s_l$. □
Theorem 1: The logical linear tree obtained from *Algorithm 1* is contention free.

Proof: The linear tree is formed by grouping all machines attached to each switch together and ordering the switches based on the DFS order. Since each machine occurs in the linear tree exactly once, the link to and from each machine is used at most once in the linear tree. Thus, the intra-switch communications do not have contention. Since the switches are ordered based on DFS, from Lemma 2, the inter-switch communications do not have any contention. Hence, the linear tree is a contention-free linear tree. □

4.2 Contention-free binary trees

As discussed earlier, since the tree height directly affects the time to complete the operation, the ideal binary tree for pipelined broadcast is one with the smallest tree height. Unfortunately, the problem of finding a contention-free binary tree with the smallest tree height is difficult to solve. In this section, we propose a heuristic that computes contention-free binary trees while trying to minimize the tree heights. Although this heuristic may not find trees with the smallest tree heights, our simulation study indicates that the trees found by this heuristic are close to optimal. The heuristic is based on the contention-free linear tree obtained from *Algorithm 1*. The following lemma is the foundation of this heuristic.
Lemma 3: Let us re-number the logical linear tree obtained from *Algorithm 1* ($n_{0,0}(n_r) \rightarrow ... \rightarrow n_{0,X_0-1} \rightarrow n_{1,0} \rightarrow ... \rightarrow n_{1,X_1-1} \rightarrow ... \rightarrow n_{|S|-1,0} \rightarrow ... \rightarrow n_{|S|-1,X_{|S|-1}-1}$) as $m_0(n_r) \rightarrow m_1 \rightarrow ... \rightarrow m_{P-1}$. Let $0 \leq i < j \leq k < l \leq P - 1$, communication $m_i \rightarrow m_j$ does not have contention with communication $m_k \rightarrow m_l$.
Proof: Let $m_i = n_{a,w}$, $m_j = n_{b,x}$, $m_k = n_{c,y}$, and $m_l = n_{d,z}$. Since $i < j \leq k < l$, $a \leq b \leq c \leq d$. $Path(m_i \rightarrow m_j)$ has three components: (m_i, s_a), $path(s_a \rightarrow s_b)$, and (s_b, m_j). $Path(m_k \rightarrow m_l)$ has three components: (m_k, s_c), $path(s_c \rightarrow s_d)$, and (s_d, m_l). When $a = b$, communication $m_i \rightarrow m_j$ does not have contention with communication $m_k \rightarrow m_l$ since (m_i, s_a) and (s_b, m_j) are not in $path(s_c \rightarrow s_d)$. Similarly, when $c = d$, communication $m_i \rightarrow m_j$ does not have contention with communication $m_k \rightarrow m_l$. When $a < b \leq c < d$, from Lemma 2, $path(s_a \rightarrow s_b)$ does not share edges with $path(s_c \rightarrow s_d)$. Hence, communication $m_i \rightarrow m_j$ does not have contention with communication $m_k \rightarrow m_l$ in all cases. □

Let $m_0 \rightarrow m_1 \rightarrow ... \rightarrow m_{P-1}$ be the linear tree obtained from *Algorithm 1*. For $0 \leq i \leq j \leq P - 1$, let us denote sub-array $S(i, j) = \{m_i, m_{i+1}, ..., m_j\}$. The heuristic constructs contention-free binary trees for all sub-arrays $S(i, j)$, $0 \leq i \leq j \leq P - 1$. Notice that for a sub-array $S(i, j)$, there always exists at least one contention-free binary tree since the linear tree is a special binary tree. Let $tree(i, j)$ represent the contention-free binary tree computed for $S(i, j)$. $Tree(0, P-1)$ is the binary tree that covers all machines. The heuristic builds $tree(i, j)$ with communications $m_a \rightarrow m_b$, $i \leq a < b \leq j$. Let $0 \leq i \leq j < k \leq l \leq P-1$, from Lemma 3, $tree(i, j)$ does not have contention with $tree(k, l)$.

```
( 1) Let $m_0 \rightarrow m_1 \rightarrow ... \rightarrow m_{P-1}$ be the linear tree
         obtained from *Algorithm 1*.
( 2) for ($i = 0$; $i < P$; $i++$) do
( 3)     $best[i][i] = 0$; $tree[i][i] = \{\}$;
( 4) enddo
( 5) for ($i = 0$; $i < P-1$; $i++$) do
( 6)     $best[i][i+1] = 1$;
( 7)     $tree[i][i+1] = \{m_i \rightarrow m_{i+1}\}$;
( 8) enddo
( 9) for ($i = 0$; $i < P-2$; $i++$) do
(10)     $best[i][i+2] = 1$;
(11)     $tree[i][i+2] = \{m_i \rightarrow m_{i+1}, m_i \rightarrow m_{i+2}\}$;
(12) enddo
(13) for ($j = 3$; $j < P$; $j++$) do
(14)     for ($i = 0$; $i < P-j$; $i++$) do
(15)         $best[i, i+j] = \infty$;
(16)         for ($k = i+2$; $k \leq i+j$; $k++$) do
(17)             if ($m_i \rightarrow m_k$ does not have contention
                         with $tree[i+1][k-1]$) then
(18)                 if ($best[i][i+j] > max(best[i+1][k-1],$
                                         $best[k][i+j]) + 1$) then
(19)                     $best[i][i+j] = max(best[i+1][k-1],$
                                         $best[k][i+j]) + 1$;
(20)                     $index = k$;
(21)                 endif
(22)             endif
(23)         enddo
(24)         $tree[i][i+j] = tree[i+1][index-1] \cup$
             $tree[index][i+j] \cup \{m_i \rightarrow m_{i+1}, m_i \rightarrow index\}$;
(25)     enddo
(26) enddo
(27) $tree[0][P-1]$ stores the final result.
```

Figure 4. Heuristic to compute contention-free binary trees (*Algorithm 2*)

Figure 4 shows the heuristic (*Algorithm 2*). In this algorithm, $tree[i][j]$ stores $tree(i, j)$, and $best[i][j]$ stores the height of $tree(i, j)$. Lines (2) to (12) are the base cases for binary trees with 1, 2, and 3 nodes. Note that under the 1-port model, $m_i \rightarrow m_{i+1}$ and $m_i \rightarrow m_{i+2}$ cannot happen at the same time. Hence, tree $\{m_i \rightarrow m_{i+1}, m_i \rightarrow m_{i+2}\}$ is the contention free binary tree for machines m_i, m_{i+1}, and m_{i+2}. Lines (13) to (26) iteratively compute trees that cover 4 to P machines. To compute $tree(i, j)$, $j > i+2$, the heuristic decides a k, $i+1 < k \leq j$, so that $tree(i, j)$ is formed by having m_i as the root, $tree(i+1, k-1)$ as the left child, and $tree(k, j)$ as the right child. Line (17) makes sure that $m_i \rightarrow m_k$ does not have contention with communications in $tree(i+1, k-1)$, which is crucial to ensure that the binary tree is contention-free. The heuristic chooses a k with the smallest $max(best[i+1][k-1], best[k][j]) + 1$ (lines (18) to (21)), which minimizes the tree height. At the end, $tree[0][P-1]$ stores the contention-free binary tree. Assume that the number of switches is less than P, the complexity of this algorithm is $O(P^4)$.

Theorem 2: The logical binary tree computed by *Algorithm 2* is contention-free.

Proof: We will prove that, for all i and j, $0 \leq i \leq j \leq P-1$, (1) $tree[i][j]$ only consists of communications $m_a \rightarrow m_b$, $i \leq a < b \leq j$; and (2) $tree[i][j]$ is contention free.

Base case: It is trivial to show that trees with 1, 2, or 3 nodes satisfy the two conditions. For example, the 2-node tree rooted at node m_i contains nodes $\{m_i, m_{i+1}\}$ and one edge $m_i \rightarrow m_{i+1}$ (from lines (5) - (8) in Figure 4). This tree satisfies condition (1) since it only consists of communications $m_i \rightarrow m_{i+1}$. This tree is contention free since there is only one communication in the tree.

Induction case: Since $tree[i][j] = tree[i+1][k-1] \cup tree[k][j] \cup \{m_i \rightarrow m_{i+1}, m_i \rightarrow m_k\}$, $i+2 < j$ and $i+1 < k \leq j$, $tree[i][j]$ only consists of communications $m_a \rightarrow m_b$, $i \leq a < b \leq j$.

From Lemma 3, communications in $tree[i+1][k-1]$ do not have contention with communications in $tree[k][j]$; $m_i \rightarrow m_{i+1}$ does not have contention with communications in $tree[k][j]$ and $tree[i+1][k-1]$; and $m_i \rightarrow m_k$ does not have contention with $tree[k][j]$. Thus, only $m_i \rightarrow m_k$ can potentially cause contention with communications in $tree[i+1][k-1]$. Since the algorithm makes sure that $m_i \rightarrow m_k$ does not cause contention with communications in $tree[i+1][k-1]$ (line (17)), there is no contention in $tree[i][j]$. □

Algorithm 2 can easily be extended to compute general k-ary trees. $S(i, j)$ can basically be partitioned into k sub-arrays which form the k subtrees. Precautions must be taken to prevent the communications from root to a subtree from causing contention with communications in the subtrees.

We evaluate the trees computed by *Algorithm 2* through simulation. Figure 5 shows the results when applying *Algorithm 2* to clusters with different sizes (up to 1024 machines). We consider two cases, on average 16 machines per switch and on average 8 machines per switch. For the 8 machines/switch case, a 1024-machine cluster has 128 switches. The cluster topologies are generated as follows. First, the size of the clusters to be studied is decided and the random tree topologies for the switches are generated by repeatedly adding random links between switches until a tree that connects all nodes is formed (links that violate the tree property are not added). After that, machines are randomly distributed to each switch with a uniform probability. For each size, 20 random topologies are generated and the average height of the 20 trees computed using *Algorithm 2* is reported. For comparison, we also show the tree heights of complete binary trees for all the sizes. As can be seen from the figure, the trees computed using *Algorithm 2* are not much taller than the complete binary tree, which indicates that the tree computed using Algorithm 2 is close to optimal. Notice that the height of the complete binary tree is the lower bound of the height of the optimal contention-free binary tree. In most cases, contention-free complete binary trees do not exist.

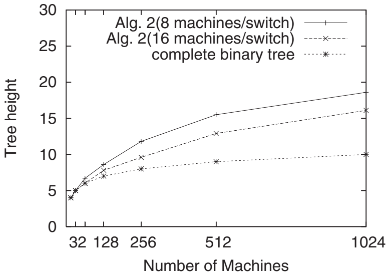


Figure 5. Performance of *Algorithm 2*

5 Experiments

In this section, we evaluate the performance of pipelined broadcast with different types of broadcast trees on different physical topologies. The physical topologies used in the evaluation are shown in Figure 6. We will refer to the topologies in Figure 6 as topologies (1), (2), (3), (4), and (5). Topology (1) contains 16 machines connected by a single switch. Topologies (2), (3), (4), and (5) are 32-machine clusters with different network connectivity. Topologies (4) and (5) have exactly the same physical topology, but different node assignments. The machines are Dell Dimension 2400 with a 2.8 GHz 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. The switches of the clusters are Dell Powerconnect 2224 (24-port 100Mbps Ethernet switches).

To evaluate the pipelined broadcast schemes, we implement automatic routine generators that take the topology information as input and automatically generate customized *MPI_Bcast* routines that employ pipelined broadcast with different contention-free broadcast trees. The generated routines are written in C. They use MPICH point-to-point primitives and are compiled with the mpicc compiler in MPICH with no additional flags in the evaluation.

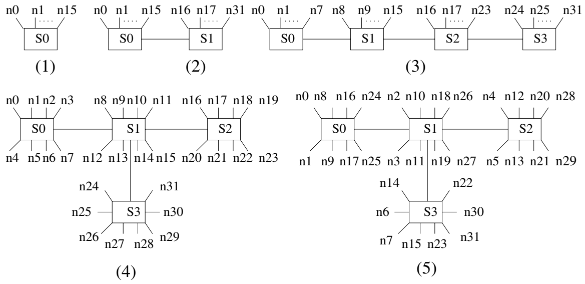


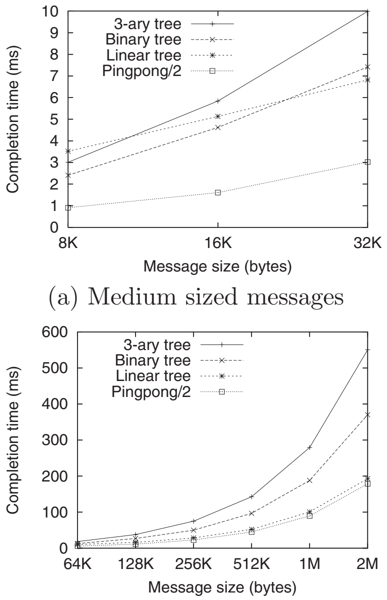
Figure 6. Topologies used in the evaluation

We compare these routines with the original *MPI_Bcast* in LAM/MPI 7.1.1 [12] and MPICH 2-1.0.1 [15]. The code segment for performance measurement is shown in Figure 7. Multiple iterations of *MPI_Bcast* are measured. Within each iteration, a barrier is added to prevent pipelined communication between iterations. Since we only consider broadcasts with $msize \geq 8KB$, the barrier overhead is insignificant to the total communication time. When reporting the performance of pipelined broadcast, by default, we only report the results with optimal segment sizes, which are determined by an automatic tuning system [4] using an empirical approach. Note that, as shown in Figure 12, while the optimal segment sizes are difficult to obtain, a wide range of segment sizes can yield performance close to the optimal.

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

Figure 7. Code segment for measuring *MPI_Bcast* **performance.**

Figure 8 shows the performance of pipelined broadcast using different contention-free trees on topology (1). The performance of pipelined broadcast on topologies (2), (3), (4), and (5) has a similar trend. As can be seen from the figure, when the message size is large ($\geq 32KB$), the linear tree offers the best performance.



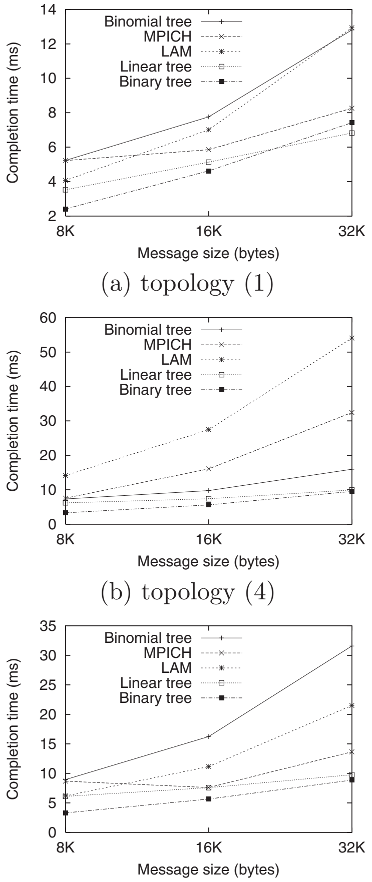
(a) Medium sized messages

(b) Large sized messages

Figure 8. Performance of pipelined broadcast with different broadcast trees on Topology (1)

For medium sized messages ($8KB$ to $16KB$), the binary tree offers the best performance. In all the experimental settings, the 3-ary tree is always worse than the binary tree, which confirms that k-ary trees, $k > 2$, are not effective. In the rest of the section, we will only show the performance of the linear tree and the binary tree. The line titled “pingpong/2” in Figure 8 shows the time to send a single message of a given size between two machines, that is, $T(msize)$. When the message size is large ($\geq 256KB$), the communication completion time for linear trees is very close to $T(msize)$, which indicates that pipelined broadcast with the linear tree is clearly a good choice for Ethernet switched clusters when the message is large. The time for binary trees is about twice the time to send a single message.

Figures 9 and 10 compare the performance of pipelined broadcast using contention-free trees with the algorithms used in LAM/MPI and MPICH on topologies (1), (4), and (5). The results for topologies (2) and (3) are similar to those for topology (4). Since all algorithms run over MPICH except LAM, we also include a binomial tree implementation (the algorithm used in LAM) over MPICH in the comparison. MPICH uses the scatter followed by all-gather algorithm for large messages ($> 12KB$) and the binomial tree for small messages. When the message size is reasonably large ($\geq 8KB$), the pipelined broadcast routines significantly out-perform the none-pipelined broadcast algorithms used in LAM and MPICH. For topology (1) and (4), when the message size is large ($\geq 512KB$), MPICH has similar performance to the pipelined broadcast using binary trees. This is compatible with our analysis in Section 3.1 that both should have a completion time of around $2 \times T(msize)$. However, pipelined broadcast with linear trees is about twice as fast as MPICH when $msize \geq 512KB$. For topology (5), MPICH performs much worse than pipelined broadcast with binary trees. This is because the MPICH all-gather routine uses topology unaware algorithms and its performance is sensitive to the physical topology [5]. Note that the MPICH all-gather routine changes algorithms when the broadcast message size is $512KB$. Hence, the performance curve for MPICH is non-continuous at this point ($512KB$) for some topologies.

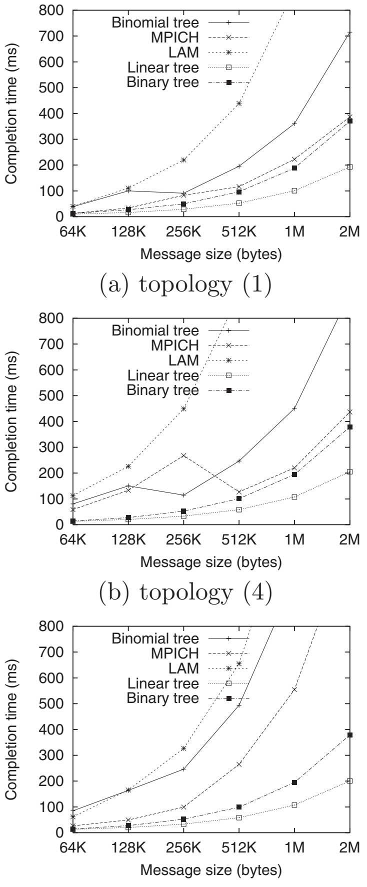


(a) topology (1)

(b) topology (4)

(c) topology (5)

Figure 9. Performance of different broadcast algorithms (medium sized messages)

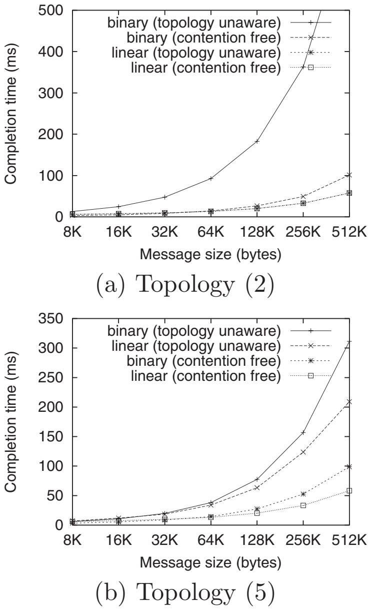


(a) topology (1)

(b) topology (4)

(c) topology (5)

Figure 10. Performance of different broadcast algorithms (large sized messages)



(a) Topology (2)

(b) Topology (5)

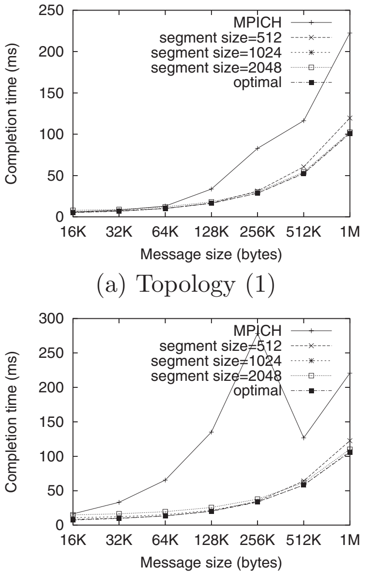
Figure 11. Contention-free broadcast trees versus topology unaware broadcast trees

Figure 11 compares pipelined broadcast using contention-free trees with that using topology unaware trees. In the comparison, we use the topology unaware linear tree in [7, 19]: $n_0 \rightarrow n_1 \rightarrow ... \rightarrow n_{P-1}$ (n_0 is the root). For topology unaware binary trees, we assume the complete binary tree, where node n_k has children n_{2k+1} and n_{2k+2} and parent $n_{\frac{k-1}{2}}$. For topology (2), the topology unaware linear tree happens to be contention free. As a result, its performance is exactly the same as the contention-free linear tree. However, for topology (5), this is not the case, the topology unaware linear tree incurs significant network contention and its performance is much worse than the contention-free linear tree. For example, for broadcasting 1MB data on topology (5), the communication completion time is 107.3ms for the contention-free linear tree and 409.7ms for the topology unaware linear tree. The contention-free tree is 300% faster than the topology unaware tree. Topology unaware binary trees cause contention in all the topologies except topology (1) in the experiments and their performance is significantly worse than the contention-free binary trees. These results indicate that to achieve high performance, contention-free broadcast trees must be used.

One of the important issues in pipelined broadcast is how to find a segment size that can achieve good performance. Figure 12 shows the impacts of segment sizes on the performance of pipelined broadcast with contention-free linear trees. The results for pipelined broadcast with binary trees have a similar trend. These figures indicate that pipelined broadcast is not very sensitive to the segment size. Changing from a segment size of $512B$ to $2048B$ does not significantly affect the performance, especially in comparison to using a different algorithm as shown in Figure 12. While finding the optimal segment size may be hard, deciding one size that can achieve good performance should not be very difficult since a wide range of segment sizes can result in near optimal performance.

6 Conclusion

In this paper, we consider pipelined broadcast on Ethernet switched clusters with multiple switches. Al-



(a) Topology (1)

(b) Topology (3)

Figure 12. Pipelined broadcast using linear trees with different segment sizes

gorithms for computing different kinds of contention-free broadcast trees on Ethernet switched clusters are developed. Pipelined broadcast is compared with other commonly used broadcast algorithms and is shown to have higher performance when the message size is large. While our techniques are developed for Ethernet switched clusters with physical tree topologies, the techniques can be applied to other types of clusters since the tree topology can be embedded on most connected networks.

References

[1] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Pipelined Broadcasts on Heterogeneous Platforms." *IEEE Trans. on Parallel and Distributed Systems*, 16(4):300-313, 2005.

[2] O. Beaumont, L. Marchal, and Y. Robert, "Broadcast Trees for Heterogeneous Platforms." *IEEE IPDPS*, 2005.

[3] J. Cohen, P. Fraigniaud, and M. Mitjana, "Scheduling Calls for Multicasting in Tree Networks." In *10th ACM-SIAM Symp. on Discrete Algorithms* (SODA '99), pages 881-882, 1999.

[4] A. Faraj and X. Yuan, "Automatic Generation and Tuning of MPI Collective Communication Routines," the *19th ACM ICS*, pages 393-402, June 20-22, 2005.

[5] A. Faraj, P. Patarasuk and X. Yuan, "Bandwidth Efficient All-to-all Broadcast on Switched Clusters," The *2005 IEEE Cluster 2005*, Sept. 27-30, 2005.

[6] S. M. Figueira, "Improving Binomial Trees for Broadcasting in Local Networks of Workstations." *VECPAR'02*, June 2002.

[7] J.Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. Dongarra, "Performance Analysis of MPI Collective Operations," *IEEE IPDPS*, 2005.

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

[9] A. Karwande, X. Yuan, and D. K. Lowenthal, "CC-MPI: A Compiled Communication Capable MPI Prototype for Ethernet Switched Clusters." *ACM SIGPLAN PPoPP*, pages 95-106, 2003.

[10] R. Kesavan, K. Bondalapati, and D.K. Panda, "Multicast on Irregular Switch-based Networks with Wormhole Routing." *IEEE HPCA*, Feb. 1997.

[11] H. Ko, S. Latifi, and P. Srimani, "Near-Optimal Broadcast in All-port Wormhole-routed Hypercubes using Error Correcting Codes." *IEEE TPDS*, 11(3):247-260, 2000.

[12] LAM/MPI Parallel Computing. http://www.lam-mpi.org/.

[13] P.K. McKinley, H. Xu, A. Esfahanian and L.M. Ni, "Unicast-Based Multicast Communication in Wormhole-Routed Networks." *IEEE TPDS*, 5(12):1252-1264, Dec. 1994.

[14] The MPI Forum. *The MPI-2: Extensions to the Message Passing Interface*, July 1997. Available at http://www.mpi-forum.org/docs/mpi-20-html/ mpi2-report.html.

[15] MPICH - A Portable Implementation of MPI. http://www.mcs.anl.gov/mpi/mpich.

[16] A Proskurowski, "Minimum Broadcast Trees." *IEEE TC*, c-30, pp. 363-366, 1981.

[17] *SCI-MPICH: MPI for SCI-connected Clusters*. Available at: www.lfbs.rwth-aachen.de/ users/joachim/SCI-MPICH/pcast.html.

[18] Andrew Tanenbaum, "Computer Networks", 4th Edition, 2004.

[19] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra, "Automatically Tuned Collective Communications," In *Proceedings of SC'00: High Performance Networking and Computing*, 2000.

[20] J. Watts and R. Van De Gejin, "A Pipelined Broadcast for Multidimentional Meshes." *Parallel Processing Letters*, 5(2)281-292, 1995.