Techniques for Pipelined Broadcast on Ethernet Switched Clusters [*]

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

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

Abstract

By splitting a large broadcast message into segments and broadcasting the segments in a pipelined fashion, pipelined broadcast can achieve high performance in many systems. In this paper, we investigate techniques for efficient pipelined broadcast on clusters connected by multiple Ethernet switches. Specifically, we develop algorithms for computing various contention-free broadcast trees that are suitable for pipelined broadcast on Ethernet switched clusters, extend the parametrized LogP model for predicting appropriate segment sizes for pipelined broadcast, show that the segment sizes computed based on the model yield high performance, and evaluate various pipelined broadcast schemes through experimentation on Ethernet switched clusters with various topologies. The results demonstrate that our techniques are practical and efficient for contemporary fast Ethernet and Giga-bit Ethernet clusters.

Keywords: Broadcast, Ethernet, Collective Communication.

1 Introduction

Switched Ethernet is the most widely used local–area–network (LAN) technology. Many Ethernet switched clusters of workstations are used for 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. In this operation, the *root* process (the sender) sends a message to all other processes in the system. The Message Passing Interface routine that realizes this operation is *MPI_Bcast* [19]. Broadcast

[*]A preliminary version of this paper is published in IEEE IPDPS 2006.

[†]Corresponding author: Xin Yuan, email: xyuan@cs.fsu.edu, phone: (850)644-9133, fax: (850)644-0058.

algorithms are classified as either *atomic broadcast* algorithms or *pipelined broadcast* algorithms. 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 message segments, a pipelined broadcast algorithm, which distributes message segments in a pipelined fashion, can usually achieve higher performance than atomic broadcast algorithms.

Two main issues must be addressed in order for pipelined broadcast to achieve high performance in a cluster. First, a broadcast tree that allows efficient pipelined broadcast must be determined. In pipelined broadcast, communications on different branches of the 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 broadcast trees for pipelined broadcast should be contention-free. Second, appropriate segment sizes must be selected since the segment sizes directly affect the total broadcast time. A small segment size may introduce excessive communication start-up overheads while a large segment size may decrease pipeline efficiency.

We investigate efficient pipelined broadcast for realizing *MPI_Bcast* with large message sizes on clusters connected by multiple Ethernet switches. In this case, broadcasting a large message is carried out by a sequence of pipelined broadcasts with smaller message segments. We develop algorithms for computing various contention-free broadcast trees that are suitable for pipelined broadcast on Ethernet switched clusters, and extend the parametrized LogP model [15] for predicting appropriate segment sizes. We evaluate the proposed techniques on fast Ethernet (100Mbps) and Giga-bit Ethernet (1000Mbps) clusters. The results show that the proposed techniques are practical and efficient on such clusters. We will refer to fast Ethernet as 100Mbps Ethernet and Giga-bit Ethernet as 1000Mbps Ethernet in the rest of the paper. The main conclusions include the following:

- Pipelined broadcast is more effective than other broadcast schemes including the ones used in MPICH 2-1.0.1 [20] and LAM/MPI 7.1.1 [16] on both 100Mbps and 1000Mbps Ethernet switched clusters in many situations.
- 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.
- Pipelined broadcast is relatively insensitive to the segment size in that the range of segment sizes that can yield high performance for a given operation is large. Our extended parameterized LogP model is sufficiently accurate for finding the appropriate segment size for a given pipelined broadcast on a platform.

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

2 Related Work

The broadcast operation has been extensively studied and a very large number of broadcast algorithms have been proposed. More closely related to this work are various pipelined broadcast schemes. Various binomial tree based pipelined broadcast algorithms have been developed [12, 14, 25, 26, 27]. In these schemes, each node sends successive segments to its children in a round-robin fashion. One example is the k-binomial tree algorithm [14], which is shown to have a better performance than traditional binomial trees. Although these schemes can achieve theoretical optimal or nearly optimal performance, the shapes of the broadcast trees are fixed. Such trees require high network connectivity to be contention free. For networks with lower connectivity, such as Ethernet that has a tree topology, such broadcast trees do not have a contention-free embedding and thus, the techniques in [12, 14, 25, 26, 27] cannot be extended to clusters connected by multiple Ethernet switches.

Pipelined broadcast has also been investigated in other environments. 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. In [29], a pipelined broadcast technique is proposed for the mesh topology. The effectiveness of pipelined broadcast in cluster environments was demonstrated in [11, 23, 28]. It was shown that pipelined broadcast using topology unaware trees can be very efficient for clusters connected by a single switch. In [22], a scheme was proposed where the broadcast tree changes smoothly from a binary tree to a linear tree as the message size increases.

In this paper, we do not propose new pipelined broadcast schemes. Instead, we develop practical techniques to facilitate the deployment of pipelined broadcast on clusters connected by multiple Ethernet switches. Similar to other architecture specific collective communication algorithms [8, 10, 17], the techniques developed in this paper can be used in advanced communication libraries [7, 9, 13, 30]. Our research extends the work in [11, 23, 28] by considering 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 fully contention-free trees for pipelined broadcast over a physical tree topology have not been developed. Moreover, although various models that can be used to determine appropriate segment sizes for pipelined broadcast have been proposed [3, 4, 6, 15, 22, 26, 27], these schemes cannot directly apply to Ethernet switched clusters either because the model assumptions do not hold or because the model parameters cannot be measured with sufficient accuracy. We extend the parameterized LogP model in [15] for determining the appropriate segment sizes in pipelined broadcasts. Notice that we could have extended the Log_nP and Log_3P models [4] to have more powerful models (e.g. having the ability to deal with non-contiguous data types). However, these models mainly focus on dealing with non-contiguous memory accesses, which is not the emphasis in our paper. As such, we extended the simpler parameterized LogP model that is sufficient for our purpose.

The pipelined broadcast approach can only be efficient for broadcasting large messages. For small messages, other broadcast algorithms are needed. There are techniques to develop adaptive MPI routines that use different algorithms according to the message sizes [7, 20]. These adaptive techniques allow our algorithms and the complementary algorithms for broadcasting small messages to co-exist in one MPI routine.

3 Network Model

We consider Ethernet switched clusters where each workstation is equipped with one Ethernet port and each Ethernet link operates in the duplex mode that supports simultaneous communications in both directions with 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 to determine forwarding paths that follow a tree structure [24]. 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 [13], which is complex. In this paper, we only consider 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. $V = S \cup M$. 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 simple path (path without a loop) between any two nodes. Figure 1 shows an example cluster. We assume that all links have the same bandwidth.

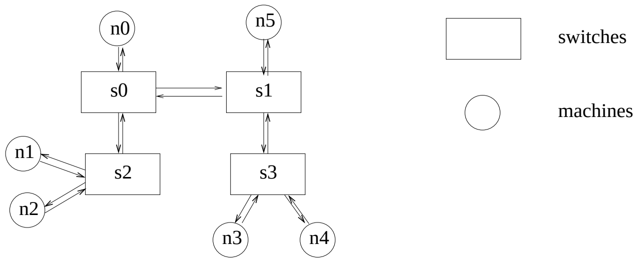


Figure 1: An example Cluster

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 simple 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 communications. 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. Splitting a large message into small segments will increase the startup overheads and thus, the total communication time. Hence, when an $msize$-byte message is split into segments of sizes s_1, s_2, ..., and s_k, $T(s_1) + T(s_2) + ... + T(s_k) \geq T(msize)$. Under the assumption that the startup overhead is insignificant in $T(s_i)$, $1 \leq i \leq k$, $T(s_1) + T(s_2) + ... + T(s_k) \approx T(msize)$. We note that the start-up overheads in Ethernet clusters are not small in absolute terms. However, the significance of the overheads depends on the segment sizes. For example, in our experimental system with

100Mbps Ethernet, the start-up overhead is less than 10% when the segment size is 1KB: by selecting a proper segment size, the start-up overheads can be made insignificant.

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 $T(msize)$. Note that a tighter lower bound is $log(P) \times \alpha + msize \times \beta$ since it takes at least $log(P)$ steps for a message to reach all P processes. In this section, we do not intend to give a detail analysis of the lower bound. To simplify the discussion, we will ignore the α term and use $T(msize)$ as the lower bound. Our simplified analysis is sufficient to justify the broadcast trees that we use. Readers can refer to [22, 26] for more detailed analysis of pipelined broadcast.

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[18], broadcast follows a hypercube communication pattern and the total number of messages that the root sends is $log(P)$. Hence, the completion time is $log(P) \times T(msize)$. Another interesting non-pipelined broadcast algorithm is the *scatter followed by all-gather* algorithm, which is used in MPICH [20]. 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 $2 \times T(\frac{P-1}{P} \times msize) \approx 2 \times T(msize)$. Note that this is a lower bound on the performance. In practice, the all-gather operation may not be completed in $T(\frac{P-1}{P} \times msize)$.

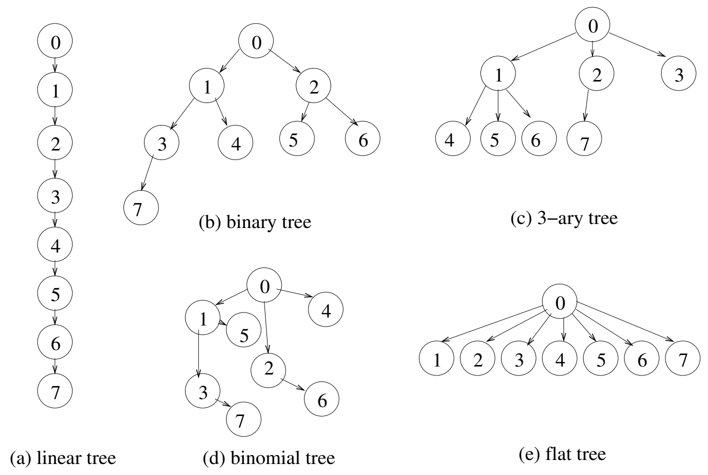


Figure 2: Examples of broadcast trees

Now, let us consider pipelined broadcast. As discussed earlier, binomial trees based pipelined broadcast schemes, such as those in [12, 14], are not applicable on Ethernet switched clusters. We will focus on schemes where each segment is pipelined over a fixed-degree tree. 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 $T(\frac{msize}{X})$, that is, $X \times T(\frac{msize}{X}) \approx T(msize)$. In our experiments, using $\frac{msize}{X} \geq 1KB$ on 100Mbps clusters and $\frac{msize}{X} \geq 8KB$ on 1000Mbps clusters, the start-up overhead is less than 10% of the overall communication time.

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. Since a node must send to each of its D children one at a time, the largest pipeline stage is $D \times T(\frac{msize}{X})$. The total time to complete the communication is

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

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. 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 balanced such that the tree height is $O(log_k(P))$. Hence, it is unlikely that a k-ary tree, $k > 2$, can offer 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 on 100Mbps and 1000Mbps Ethernet clusters. Table 1 summarizes the performance of

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

broadcast algorithms when the broadcast message size is very large.

4 Pipelined broadcast on Ethernet switched networks

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 on Ethernet switched clusters: (1) finding contention-free broadcast trees, and (2) deciding appropriate segment sizes. We will address both issues in this section.

4.1 Computing contention-free broadcast trees

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.

4.1.1 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, $0 \le i \le |S| - 1$, 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. $\square$

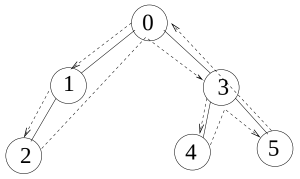


Figure 3: DFS numbering

The proof of Lemma 1 can be found in [10]. 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: each directed edge (direction) of each link is used exactly once in the communication pattern.

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. □

4.1.2 Contention-free binary trees

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. Among all binary trees, the complete binary tree has the smallest tree height. Unfortunately, such a tree (and other fixed-shape trees such as binomial trees) does not always have a contention-tree embedding on an arbitrary physical tree topology. We are not able to develop an algorithm that guarantees finding a contention-free binary tree with the smallest tree height. Instead, we propose a heuristic that computes contention-free binary trees while trying to minimize the tree heights. 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* ($n_{0,0} \rightarrow ... \rightarrow n_{0,X_0-1} \rightarrow n_{1,0} \rightarrow ... \rightarrow n_{1,X_1-1} \rightarrow ... \rightarrow n_{|S|-1,0} \rightarrow ... \rightarrow n_{|S|-1,X_{|S|-1}-1}$) as $m_0 \rightarrow m_1 \rightarrow ... \rightarrow m_{P-1}$. Here, $n_{0,0} = m_0 = n_r$. 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. $\square$

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 $tree(k,l)$.

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 a 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 $O(P^4)$. Note that this algorithm is used off-line to compute contention-free trees (e.g it is used in our automatic *MPI_Bcast* routine generator that reads in the topology information and outputs a topology specific routine). Hence, the complexity of $O(P^4)$ is not an issue.

```
( 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*)

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 $tree[i][j]$. $\square$

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.

While *Algorithm 2* tries to minimize the tree height, the trees computed using this algorithm may not be the optimal. We evaluate the trees computed by *Algorithm 2* through simulation. Figure 5 shows the results when applying *Algorithm 2* to random clusters with different sizes (up to 1024 machines). In this experiment, 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 random 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 average heights of the trees computed using *Algorithm 2* are within a factor of two of the heights of the complete binary tree. Notice that the height of the complete binary tree is the lower bound of the height of the optimal contention-free binary tree.

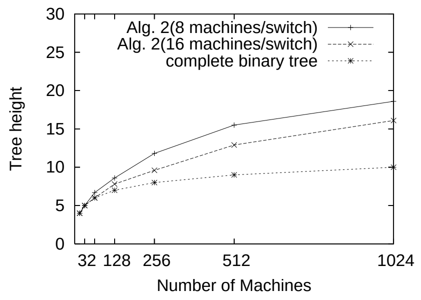


Figure 5: Performance of *Algorithm 2*

4.2 Model for computing appropriate segment sizes

Our model is a minor extension of the parameterized LogP model [15]. In the parameterized LogP model, the point-to-point communication performance is characterized by five parameters, $(L, o_s(m), o_r(m), g(m), P)$. L is the end-to-end delay for a data transfer, which includes all contributing factors such as network set-up and buffer copies. Parameters $o_s(m)$ and $o_r(m)$ are the times that the CPUs are busy sending and receiving a message of size m, respectively. The gap $g(m)$ is the minimum time interval between consecutive message (of size m) transmissions and receptions. The gap $g(m)$ also covers all contributing factors including $o_s(m)$ and $o_r(m)$. It follows that $g(m) \geq o_s(m)$ and $g(m) \geq o_r(m)$. P is the number of processors in the system. This parameterized LogP model extends the original

LogP model [6] in that the parameters o_s, o_r, and g are functions of the message size m, which allows the communication time of large messages to be modeled more accurately.

Since the L term in the parameterized LogP model contains all contributing factors including buffer copying whose time is proportional to the size of the message in an Ethernet switched cluster, to accurately model the communication, the L term must also be a function of the message size (m). Hence, in our extended parameterized LogP model, the latency term, $L(m)$, is a function of the message size and the model is characterized by five terms: $(L(m), o_s(m), o_r(m), g(m), P)$. Figure 6 shows an example for modeling the time when sending 4 consecutive messages of size m using the extended parameterized LogP model.

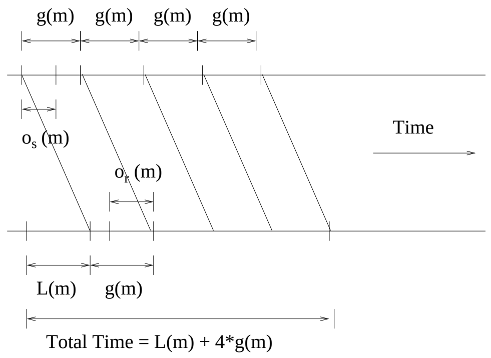


Figure 6: The time for sending 4 consecutive messages of size m

A pipelined broadcast can be decomposed into a set of point-to-point communications. The model for pipelined broadcast with a contention-free broadcast tree does not need to consider the network contention issue. Hence, the communication completion time can be obtained by combining the times of the point-to-point communications in the critical path. Let the message size $msize$ be partitioned into X segments, the segment size is $m = \frac{msize}{X}$ (for easy exposition, we will assume that $msize$ is divisible by X). Figure 7 shows how the pipelined broadcast with a linear broadcast tree can be modeled: since it takes $L(m) + g(m)$ time for a segment to be transferred from one processor to another processor, as shown in Figure 7 (b), it takes $(P-1)(L(m) + g(m))$ time for the first segment to reach the last processor. With pipelined broadcast, the rest (X-1) segments are propagated in a pipelined fashion in the broadcast tree (shown in Figure 7 (c)) and the communication completion time is thus,

$$Time_{linear\_tree} = (P-1)(L(\frac{msize}{X}) + g(\frac{msize}{X})) + (X-1)g(\frac{msize}{X}) \quad (1)$$

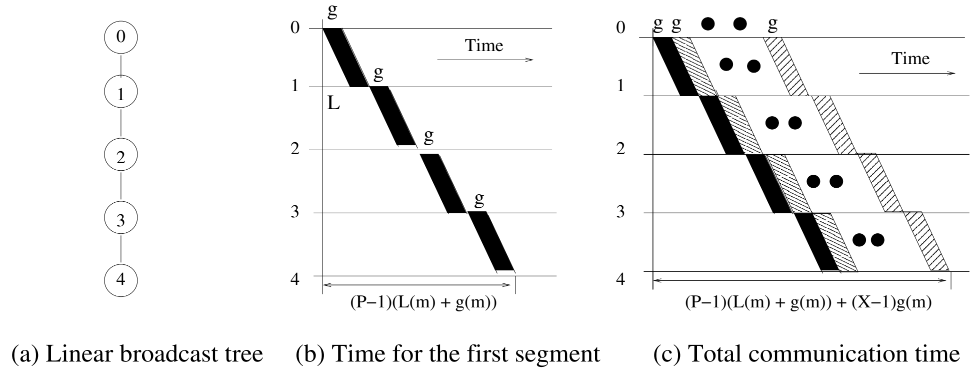
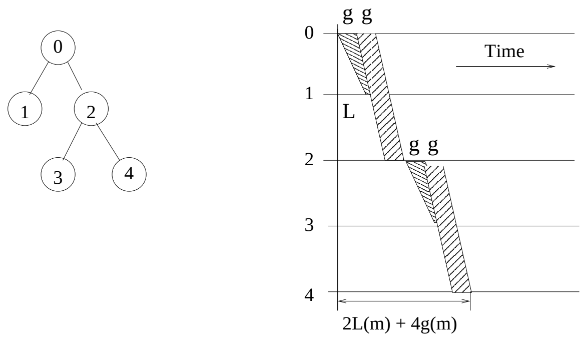

0
1
2
3
4
g
L
Time
(P−1)(L(m) + g(m))
(P−1)(L(m) + g(m)) + (X−1)g(m)
(a) Linear broadcast tree
(b) Time for the first segment
(c) Total communication time

Figure 7: Modeling pipelined communication with a linear tree

The communication completion time for pipelined broadcast with a binary tree depends on the tree topology. Without loss of generality, we will assume that a node in the broadcast tree first sends the segment to the left child and then to the right child. Hence, the time for a segment to be transferred from a processor to its left child is $L(m) + g(m)$ while the time for the right child is $L(m) + g(m) + g(m) = L(m) + 2g(m)$. The time for a segment to reach a receiver can be obtained by summing the times for the segment to go through each intermediate node. Specifically, let the path from the root to a receiver be $m'_0 \rightarrow m'_1 \rightarrow m'_2 \rightarrow ... \rightarrow m'_{x-1} \rightarrow m'_x$ (m'_0 is the root and m'_x is the receiver). The time for each hop $m'_j \rightarrow m'_{j+1}$ is denoted as $Time(m'_j \rightarrow m'_{j+1})$, $0 \leq i < x$. $Time(m'_j \rightarrow m'_{j+1}) = L(m) + g(m)$ if m'_{j+1} is the left child of m'_j; and $Time(m'_j \rightarrow m'_{j+1}) = L(m) + 2g(m)$ if m'_{j+1} is the right child of m'_j. The total time for a segment to reach a receiver m'_x is the sum of the times on each hop and has the form of $A_{m'_x} \times L(m) + B_{m'_x} \times g(m)$, where $A_{m'_x}$ and $B_{m'_x}$ are constants. The time for the first segment to reach the last processor is thus $max_{m'_x}\{A_{m'_x} \times L(m) + B_{m'_x} \times g(m)\}$, denoted as $A \times L(m) + B \times g(m)$. After the first segment reaches the last processor, the rest $X-1$ segments will be sent in a pipelined fashion, which introduces $(X-1) \times 2 \times g(m)$ time. Hence, the communication time for pipelined broadcast with a binary tree is

$$Time_{binary\_tree} = A \times L(\frac{msize}{X}) + B \times g(\frac{msize}{X}) + 2(X-1)g(\frac{msize}{X}) \quad (2)$$

Figure 8 shows an example for modeling the binary tree. For the topology in Figure 8 (a), the last processor to receive the message is processor 4. As shown in the Figure 8 (b), $A = 2$ and $B = 4$. Hence, the communication completion time for pipelined broadcast on this tree is thus, $2L(m) + 4g(m) + 2(X-1)g(m)$.



(a) An example binary broadcast tree (b) Time for the first segment

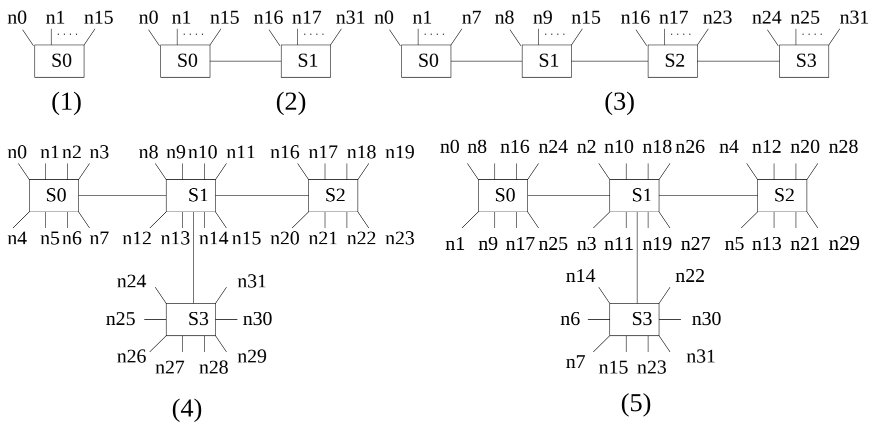
Figure 8: Modeling pipelined communication with a binary tree

As can be seen from formulas (1) and (2), the communication completion time for pipelined broadcast can be decided once P, the broadcast tree topology, $L(m)$, and $g(m)$ are decided. Since P is known and the broadcast tree topology is computed based on the algorithms discussed in the previous section, we only need to know the values for $L(m)$ and $g(m)$ in order to apply the model.

5 Experiments

In this section, we validate the performance model and evaluate the performance of pipelined broadcast with various broadcast trees on 100Mbps (fast) Ethernet and 1000Mbps (Gigabit) Ethernet clusters with different physical topologies. The physical topologies for both 100Mbps and 1000Mbps Ethernet clusters used in the evaluation are shown in Figure 9. We will refer to the topologies in Figure 9 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, 640MB 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 (10/100/1000 Mbps Ethernet card) with the driver from Broadcom. The 100Mbps Ethernet switches are Dell Powerconnect 2224 (24-port 100Mbps Ethernet switches) and the 1000Mbps Ethernet switches are Dell Powerconnect 2724 (24-port 1000Mbps switches).



n0 n1 n15
S0
(1)
n0 n1 n15 n16 n17 n31
S0 S1
(2)
n0 n1 n7 n8 n9 n15 n16 n17 n23 n24 n25 n31
S0 S1 S2 S3
(3)
n0 n1 n2 n3 n8 n9 n10 n11 n16 n17 n18 n19
S0 S1 S2
n4 n5 n6 n7 n12 n13 n14 n15 n20 n21 n22 n23
n24 n31
n25 S3 n30
n26 n27 n28 n29
(4)
n0 n8 n16 n24 n2 n10 n18 n26 n4 n12 n20 n28
S0 S1 S2
n1 n9 n17 n25 n3 n11 n19 n27 n5 n13 n21 n29
n14 n22
n6 S3 n30
n7 n15 n23 n31
(5)

Figure 9: Topologies used in the evaluation

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 with MPICH point-to-point primitives.

In the following sub-sections, we will first validate the performance model and show that this model can be used to compute appropriate segment sizes for pipelined broadcast. We will then investigate different aspects of pipelined broadcast and compare pipelined broadcast with the original *MPI_Bcast* in LAM/MPI 7.1.1 [16] and MPICH 2-1.0.1 [20]. The code segment for measuring communication completion time is shown in Figure 10. 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 reasonable large messages, the barrier overhead is insignificant to the communication time. The programs are compiled with the mpicc compiler in MPICH with no additional flags. In the experiments, *ITER_NUM* is set to 20, and the average of five experiments of each experience is reported unless specified otherwise.

```
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 10: Code segment for measuring MPI_Bcast performance.

5.1 Performance model validation

Our extended parameterized LogP model characterizes the system with five parameters, $(L(m), o_s(m), o_r(m), g(m), P)$. Among these five parameters, only $L(m)$, $g(m)$, and P are used in the formulas (1) and (2). Since L and g are functions of m, ideally, one would like to build a table for L and g for every possible value of m. This is practically impossible. In [26], it is shown that under the linear cost model, the optimal segment size is $O(\sqrt{msize})$. Guided by this result, we select a range of potential sizes from 256B to 32KB: 256B, 512B, 1024B, 2048B, 4096B, 8192B, 16384B, and 32768B. Using these eight sizes guarantees that the optimal segment size is within a factor of two of the selected sizes assuming that the optimal segment sizes fall within the range. The L and g values for these message sizes in our cluster are shown in Table 2. We measure $g(m)$ as follows. For each message size m that is of interest, we perform one way communication for a very large number ($M = 100000$) of times and measure the sender side latency for all of the M sends. $g(m)$ is equal to the latency divided by M. To obtain $L(m)$, we use a pingpong program to measure the round trip time for the messages of size m ($RTT(m)$) and derive $L(m)$ based on the formula, $RTT(m) = L(m)+g(m)+L(m)+g(m)$, or $L(m) = RTT(m)/2-g(m)$. The round trip time is the average of 1000 iterations. The experiments for both $g(m)$ and $RTT(m)$ are repeated 36 times and the average of the 36 experiments are reported. Both $g(m)$ and $RTT(m)$ are very consistent in our cluster: the confidence interval for the 99% confidence level is less than 0.5% of the average value. Notice that by using $RTT(m) = 2g(m) + 2L(m)$, we implicitly assume that the network is the bottleneck of the communication. The assumption is typically

	100Mbps Ethernet			1000Mbps Ethernet		
message size(m)	$g(m)$	RTT(m)	$L(m)$	$g(m)$	RTT(m)	$L(m)$
256B	0.030ms	0.280ms	0.110ms	0.013ms	0.118ms	0.046ms
512B	0.051ms	0.414ms	0.156ms	0.017ms	0.141ms	0.054ms
1024B	0.089ms	0.678ms	0.250ms	0.025ms	0.185ms	0.068ms
2048B	0.177ms	0.988ms	0.318ms	0.039ms	0.246ms	0.083ms
4096B	0.351ms	1.334ms	0.317ms	0.069ms	0.301ms	0.082ms
8192B	0.695ms	2.038ms	0.324ms	0.133ms	0.422ms	0.078ms
16384B	1.389ms	3.428ms	0.325ms	0.267ms	0.658ms	0.062ms
32768B	2.781ms	6.208ms	0.323ms	0.541ms	1.142ms	0.030ms

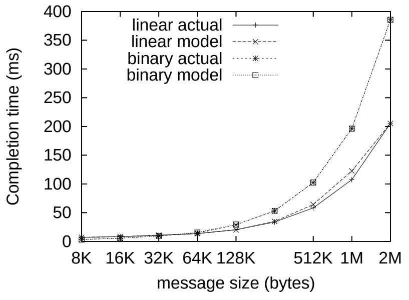
Table 2: The parameters for the extended parameterized LogP model

true for the relatively slow Ethernet: it is true for 100Mbps Ethernet. The assumption also holds for 1000Mbps when the message size is small. When the CPU speed is the bottleneck, however, $L(m)$ can be under-estimated with the formula. In our experimental system, the CPU is the bottleneck with 1000Mbps Ethernet when the message size is more than 8KB. This is why $L(m)$ decreases when m increases from 8KB to 32KB for the 1000Mbps case in Table 2.

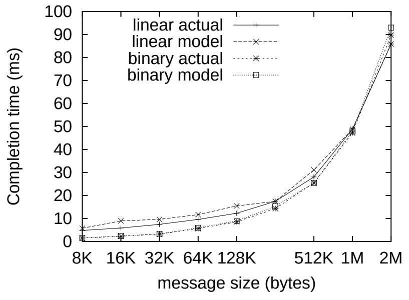
Tables 3 shows the optimal segment sizes (among the eight segment sizes) computed from our performance model with those obtained from empirical measurements on 100Mbps and 1000Mbps Ethernet clusters. Figure 11 compares the measured optimal pipelined broadcast performance with the performance using the predicted optimal segment size by the model. As can be seen from the Table 3, for both 100Mbps and 1000Mbps clusters and both linear tree and binary tree algorithms, the trend in the predicted optimal segment sizes is the same as that in the measured optimal segment sizes. However, the predicted optimal segment sizes sometimes differ from the measured sizes. Several factors contribute to this. First, in our modeling of pipelined broadcast, we assume the 1-port model where each node can send and receive at the link speed. While this assumption holds for the clusters with 100Mbps Ethernet, it is not the case for clusters with 1000Mbps Ethernet: the CPU (2.8GHz P4) cannot keep up with sending and receiving at 1000Mbps at the same time. The insufficient CPU speed significantly affects the performance of the linear tree algorithm on 1000Mbps clusters. The second reason is the inaccuracy in the performance and parameter measurements. In some cases, the performance difference for two adjacent segment sizes is too close to be distinguished due to the measurement inaccuracy. Even though the optimal segment sizes computed based on the performance model do not always match the measured optimal sizes, the performance of pipelined broadcast using the segment sizes computed based on the performance model closely matches the measured optimal pipelined communication performance as shown in Figure 11. One reason is that the performance of the pipelined broadcast algorithms is not very sensitive to the segment size. These results demonstrate that our performance model is sufficiently accurate for predicting appropriate segment sizes on contemporary 100Mbps and 1000Mbps Ethernet switched clusters.

message size	100Mbps Ethernet				1000Mbps Ethernet			
	Linear tree		Binary tree		Linear tree		Binary tree	
	Model	Actual	Model	Actual	Model	Actual	Model	Actual
8KB	256B	256B	256B	512B	256B	512B	512B	1024B
16KB	256B	512B	512B	512B	256B	512B	1024B	2048B
32KB	256B	512B	1024B	1024B	512B	2048B	2048B	2048B
64KB	256B	512B	1024B	1024B	1024B	2048B	4096B	4096B
128KB	512B	512B	1024B	1024B	1024B	2048B	4096B	8192B
256KB	512B	1024B	1024B	2048B	2048B	2048B	4096B	8192B
512KB	1024B	1024B	1024B	4096B	4096B	2048B	8192B	8192B
1MB	1024B	2048B	2048B	4096B	4096B	4096B	8192B	16384B
2MB	1024B	1024B	4096B	4096B	4096B	4096B	8192B	16384B

Table 3: Predicted optimal segment sizes versus actual optimal segment sizes on 100Mbps and 1000Mbps clusters (topology (4))



(a) 100Mbps Ethernet



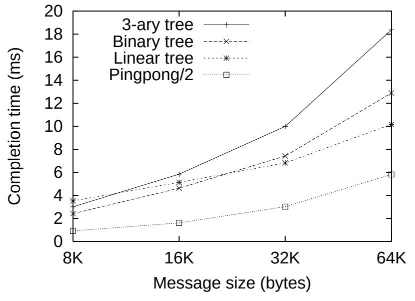
(b) 1000Mbps Ethernet

Figure 11: The measured optimal performance versus the performance using the predicted optimal segment size (topology (4))

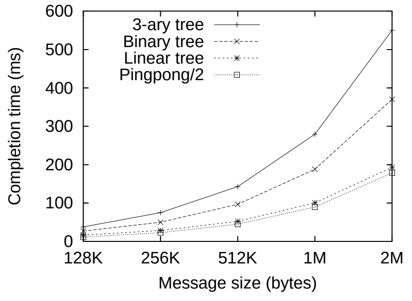
Since the extended parameterized LogP model can be used to compute appropriate segment sizes that result in a performance close to optimal, when reporting the performance of pipelined broadcast, we only report the results with best segment sizes among the eight sizes that we consider.

5.2 Results on 100Mbps Ethernet clusters

In this section, we investigate various aspects of pipelined broadcast on 100Mbps Ethernet switched clusters. Figure 12 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. For medium sized messages ($8KB$ to $16KB$), the binary tree offers the best performance. In all 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 12 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 ($\geq 128KB$), 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 100Mbps Ethernet switched clusters when the message is large. The time for binary trees is about twice the time to send a single message.



(a) Medium sized messages



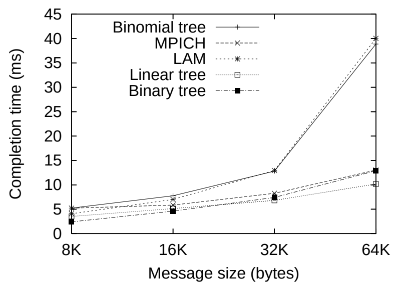
(b) Large sized messages

Figure 12: Pipelined broadcast with different broadcast trees (Topology (1), 100Mbps)

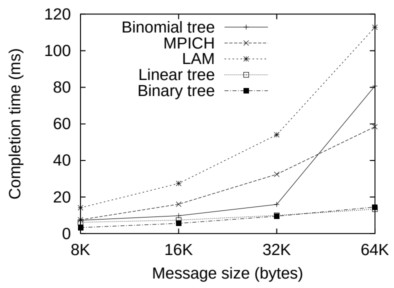
Figures 13 and 14 compare the performance of pipelined broadcast using contention-free trees with the algorithms used in LAM/MPI and MPICH on topologies (1), (4), and (5). The results for topologies (2) and (3) are similar to those for topology (4). Since all algorithms run over MPICH except LAM, we 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 when the message size is larger than $512KB$ and the binomial tree for smaller messages. When the message size is reasonably large ($\geq 8KB$), the pipelined broadcast routines (with a linear tree) significantly out-perform the non-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 with 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 $msize \geq 512KB$. On topology (5), all of the algorithms in LAM and MPICH perform poorly. This shows that the performance of these topology unaware algorithms is sensitive to the physical topology, which manifests the advantage of performing pipelined broadcast with contention-free trees.

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 ($512KB$). MPICH algorithms are topology unaware. From Figures 14 (b) and (c), we can see that although the algorithm for $\geq 512KB$ messages performs better on topology (4), it performs worse on topology (5). In all topologies with 100Mbps connection, the linear tree algorithm is much better than the MPICH algorithm; the improvement ranges from more than 30% for 64KB data to more than 200% for 1MB data for all physical topologies.

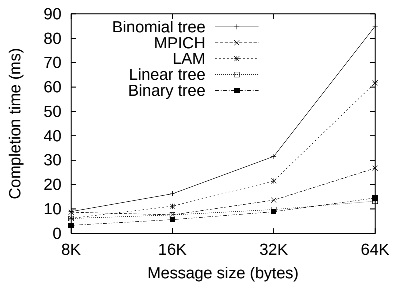
Figure 15 compares pipelined broadcast using contention-free trees with that using topology unaware trees. In the comparison, we use the topology unaware linear tree in [11, 28]: $n_0 \rightarrow n_1 \rightarrow ... \rightarrow n_{P-1}$ (n_0 is the root). For topology unaware binary trees, we assume the



(a) Topology (1)

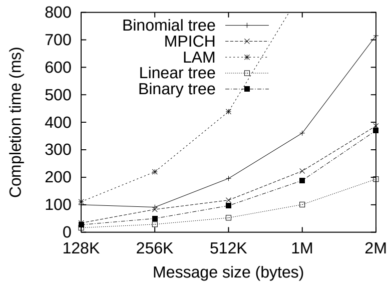


(b) Topology (4)

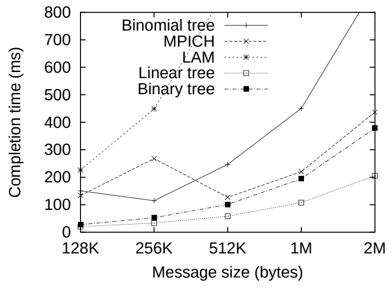


(c) Topology (5)

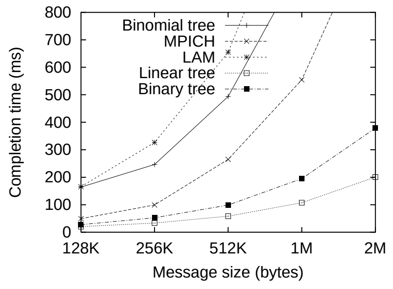
Figure 13: Performance of different algorithms (100Mbps, medium sized messages)



(a) Topology (1)



(b) Topology (4)

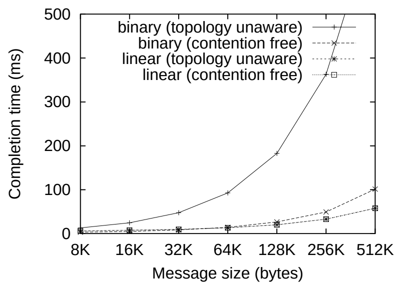


(c) Topology (5)

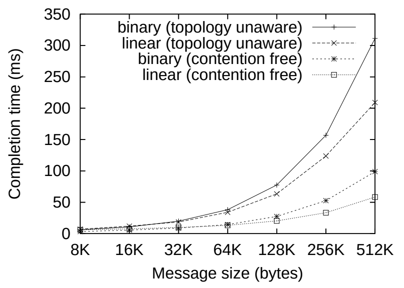
Figure 14: Performance of different algorithms (100Mbps, large messages)

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. Topology unaware binary trees cause contention in all the topologies except topology (1) 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.

Figure 16 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

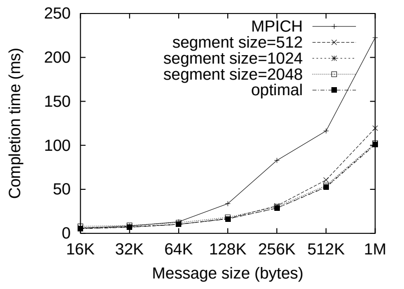


(a) Topology (2)

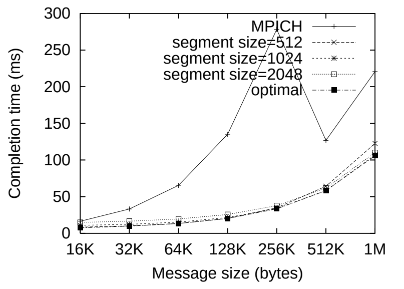


(b) Topology (5)

Figure 15: Contention-free broadcast trees versus topology unaware broadcast trees



(a) Topology (1)



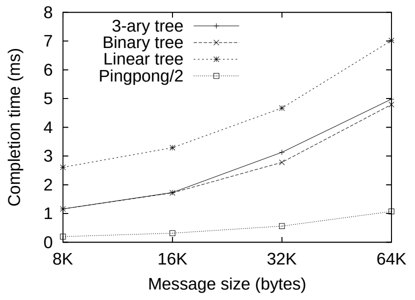
(b) Topology (3)

Figure 16: Pipelined broadcast using linear trees with different segment sizes

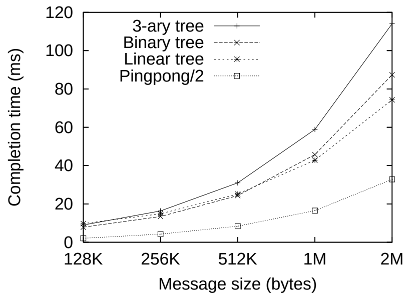
performance, especially in comparison to using a different algorithm as shown in Figure 16. This explains the reason that the optimal segment sizes computed by the performance model yield close to optimal performance even though the segment sizes do not always match the measured optimal segment sizes.

5.3 Results on 1000Mbps Ethernet switched clusters

In this section, we investigate pipelined broadcast on 1000Mbps Ethernet. Figure 17 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. In 1000Mbps Ethernet, the linear tree performs better than binary tree only when message size is larger than 1MB (compared to 32KB on 100Mbps clusters). Two major factors contribute to this. First, on 1000Mbps Ethernet, the CPU (2.8GHz P4) cannot keep up with sending and receiving data at 1000Mbps at the same time. This affects the linear tree algorithm more than it does the binary tree algorithm since the binary tree pipelined broadcast algorithm is less computational intensive than the linear tree algorithm. Assuming that the CPU speed is not a limiting factor, in the binary tree algorithm, each node takes two units of time to perform two sends and one receive while the linear tree algorithm requires each node to perform one send and one receive in one unit of time. The other factor is the relatively larger software start-up overheads in 1000Mbps Ethernet. Figure 17 also shows that the 3-ary tree is always worse than the binary tree on the 1000Mbps cluster, which confirms that k-ary trees, $k > 2$, are not effective. The line titled "pingpong/2" in Figure 17 shows $T(msize)$. As can be seen from the figure, unlike the results for 100Mbps Ethernet, on 1000Mbps Ethernet, the performance of pipelined broadcast cannot approach $T(msize)$ even for very large message sizes (e.g. 2MB). This is due to the insufficient CPU speed, which significantly affects the performance of the linear tree algorithm (the only algorithm that can theoretically achieve close to $T(msize)$ time).



(a) Medium sized messages



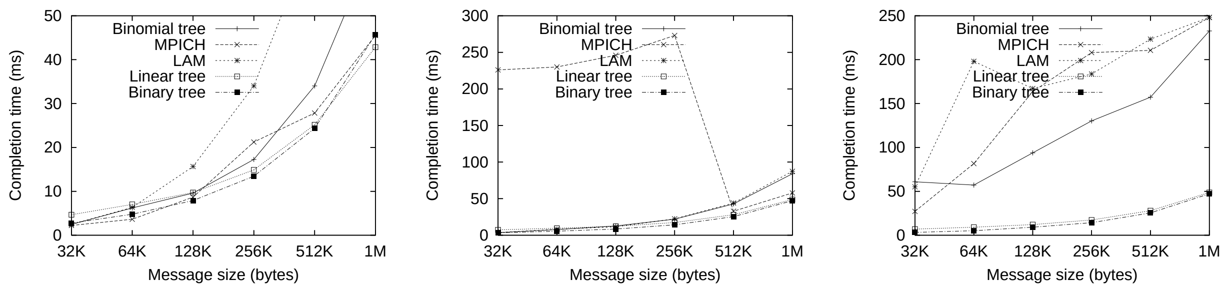
(b) Large sized messages

Figure 17: Pipelined broadcast with different broadcast trees (Topology (1), 1000Mbps)

Figure 18 compares 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) have the same trend as those for topology (4). Since all algorithms run over MPICH except LAM, we also include a binomial tree implementation over MPICH in the comparison. These three topologies ((1), (4), and (5))) show the behavior of these algorithms in different situations. In topology (1), all nodes are connected to a single switch.

Thus, network contention is not a problem for all algorithms. As seem in the Figure 18 (a), pipelined broadcast is noticeably better than other algorithms in this configuration when $msize \geq 128KB$. For example, at 128KB, the time for pipelined broadcast with binary tree is 7.8ms, which is 11.5% better than the 8.7ms for MPICH. At 512KB, the time is 24.3ms for pipelined broadcast (with binary tree), which is a 14.4% improvement over the 27.8ms for MPICH. In comparison to the results on 100Mbps clusters, where the improvement is more than 200% for large messages, the improvement in 1000Mbps systems is smaller. On this topology, the relative performance of the MPICH algorithm and pipelined broadcast with a binary tree on the 1000Mbps cluster is similar to that on the 100Mbps cluster. However, the insufficient CPU speed significantly affect the linear tree algorithm. We expect a larger improvement when the processing nodes are faster.

The cases when the network contention is a problem for topology unaware algorithms in LAM and MPICH are shown in Figures 18 (b) and (c) for topologies (4) and (5). As shown in Figure 18 (b), on topology (4), the recursive-doubling algorithm used in MPICH for the message size range from 32KB to 256KB introduces severe network contention in this range and yields extremely poor performance. The binomial tree algorithm used in LAM is not affected by network contention as much. However, its performance is poor compared to the pipelined broadcast algorithms. For example, at 1MB, the time for binary tree pipelined broadcast is 47.2ms, a 85% improvement over the 87.4ms for LAM. The MPICH algorithm for $msize \geq 512KB$, which is also topology unaware, does not introduce contention in this particular topology (topology (4)). Yet, its performance is worse than pipelined broadcast. At 1MB, pipelined broadcast with a binary tree (47.2ms) out-performs MPICH (57.4ms) by 21.6%. On topology (5), all algorithms in LAM and MPICH incur severe network contention and perform much worse than the pipelined broadcast across all the message sizes in Figure 18 (c). These experiments show that pipelined broadcast performs better than the algorithms in MPICH and LAM on 1000Mbps Ethernet clusters in all different situations with or without network contention.



(a) Topology (1) (b) Topology (4) (c) Topology (5)

Figure 18: Performance of different algorithms on 1000Mbps clusters

5.4 Properties of pipelined broadcast algorithms

The results in the previous sub-sections show that pipelined broadcast algorithms achieve high performance in many situations. In this sub-section, we further investigate the properties of the pipelined algorithms. In particular, we will try to answer the following three questions: What is the "sufficiently large" message size for a pipelined broadcast algorithm to be efficient? Can the pipelined broadcast algorithms be applied to other networks? Why is the performance of the pipelined algorithm not very sensitive to segment sizes?

As shown in Section 3, the total communication time for the pipelined broadcast algorithms can be expressed as $(X + H - 1) \times (D \times T(\frac{msize}{X}))$, where X is the number of segments, $\frac{msize}{X}$ is the segment size, H is the tree height ($H = P$ for linear tree algorithms and $H = O(log(P))$ for binary tree algorithms), and D is the nodal degree of the broadcast tree ($D = 1$ for linear tree and $D = 2$ for the binary tree). The communication time can be partitioned into two components: $D \times X \times T(\frac{msize}{X})$ and $D \times (H - 1) \times T(\frac{msize}{X})$. The part $D \times X \times T(\frac{msize}{X})$, which will be called the *pipeline* term, reflects the additional software start-up overheads for splitting the broadcast message into segments; the part $D \times (H - 1) \times T(\frac{msize}{X})$, which will be called the *delay* term, reflects the time to propagate the message across the broadcast tree. There are two conditions for the pipelined broadcast algorithms to be effective: (1) the software overhead for splitting a large message into segments should not be excessive, that is, $X \times T(\frac{msize}{X})$ must be close to $T(msize)$; and (2) the pipeline term must dominate the delay term, that is, X must be much larger than $H-1$.

Table 4 shows the values of $X \times T(\frac{msize}{X})$ with different broadcast message sizes and segment sizes. The last row is $T(msize)$ (segment size $= msize$). As can be seen from the table, the ratio between $X \times T(\frac{msize}{X})$ and $T(msize)$, which reflects the additional start-up overheads introduced by splitting the message into segments, mainly depends on the segment size ($\frac{msize}{X}$), and not on other factors such as the message size and the number of segments on both 100Mbps and 1000Mbps clusters. In particular, across all broadcast message sizes, for the 100Mbps Ethernet, when the segment size $\frac{msize}{X} \geq 1024B$, $X \times T(\frac{msize}{X})$ is within 10% of $T(msize)$; for the 1000Mbps Ethernet, when the segment size $\frac{msize}{X} \geq 8KB$, $X \times T(\frac{msize}{X})$ is within 10% of $T(msize)$. When the message size is smaller than these thresholds, the communication start-up overheads increase more dramatically. Note that the optimal segment size for a scheme may be less than the thresholds due to the compromise between software overhead and pipeline efficiency.

Segment size ($\frac{msize}{X}$)	100Mbps Ethernet				1000Mbps Ethernet			
	32KB	128KB	512KB	2MB	32KB	128KB	512KB	2MB
256	3.73	15.03	60.28	200.93	2.52	8.52	26.87	102.07
512	3.35	13.00	51.79	198.61	1.51	5.19	18.00	70.55
1024	**3.10**	**11.82**	**46.28**	**184.15**	0.89	3.40	13.31	53.01
2048	3.08	11.65	45.60	181.35	0.73	2.73	10.75	42.83
4096	3.01	11.45	45.12	180.00	0.63	2.36	9.36	37.31
8192	3.00	11.41	46.65	179.13	**0.60**	**2.17**	**8.56**	**33.57**
16384	2.99	11.37	44.86	178.79	0.59	2.11	8.26	32.74
32768	2.99	11.36	44.81	178.60	0.57	2.08	8.06	31.93
msize	2.99	11.35	44.91	178.57	0.57	2.10	8.31	32.64

Table 4: $X \times T(\frac{msize}{X})$ (milli-second) for different message sizes and segment sizes

Let us now consider the first question: what is the "sufficiently large" message size? There are two conditions for the pipelined algorithms to be efficient. First, the segment size needs to be sufficient large such that $X \times T(\frac{msize}{X})$ is close to $T(msize)$. From Table 4, we can see that the good segment size is 1KB for 100Mbps clusters and 8KB for 1000Mbps clusters. Second, X must be sufficiently larger than $H-1$. When other algorithm parameters are fixed, the term *sufficiently large message size* can be quantified. For example, assume that we want to ensure that the delay term is less than one third of the total time, $X \geq 2\times(H-1)$. In this case, to broadcast on a 32-process system using the linear tree algorithm, the broadcast message is sufficiently large when the size is larger than $2 \times (32-1) \times 1KB = 62KB$ for 100Mbps clusters and $2\times(32-1)\times 8KB = 496KB$ for 1000Mbps clusters. Using the binary tree algorithm, assuming $H = log(P)$, the message size is sufficiently large when it is larger than $2 \times (log(32)-1) \times 1KB = 8KB$ for 100Mbps clusters and $2 \times (log(32)-1) \times 8KB = 64KB$ for 1000Mbps clusters. Notice that the number of processes and the broadcast tree also significantly affect the message size for a pipelined algorithm to be effective: the linear tree pipelined algorithm is usually efficient for broadcasting on a small number of processes while the binary tree algorithm can be applied to a large number of processes.

The proposed pipelined broadcast algorithms are network oblivious. Whether such algorithms can be efficient on other types of networks depends on whether the two conditions can be met. The second condition (the pipeline term must dominate the delay term) is not system dependent. Hence, whether the algorithms can be efficient on a cluster depends on whether it is possible to split a large message into smaller segments without incurring excessive overheads. We note that in networks with higher speeds than Ethernet, such as the 20Gbps InfiniBand, communication start-up overheads are in general more significant with respect to the network bandwidth in comparison to Ethernet. This may render pipelined broadcast schemes less efficient. However, to decide whether pipelined broadcast can be effective on a network, further study is need to examine the additional start-up overheads introduced when a large message is split in the network.

Table 4 also explains why the pipelined broadcast algorithms are not very sensitive to segment sizes in Ethernet clusters: when the pipeline term dominates the total communication time, the total time is mainly affected by $X \times T(\frac{msize}{X})$; and for a wide range of segment sizes, $X \times T(\frac{msize}{X})$ is close to $T(msize)$ (and not sensitive to segment sizes).

6 Conclusion

We consider pipelined broadcast on Ethernet switched clusters with multiple switches. Algorithms for computing various contention-free broadcast trees on Ethernet switched clusters are developed. A performance model that can be used to compute the appropriate segment size for a given broadcast operation is described. We show that pipelined broadcast is more efficient than other commonly used broadcast algorithms on contemporary 100Mbps and 1000Mbps Ethernet switched clusters in many situations. 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: the near-optimal broadcast performance can be achieved on a system with an irregular topology by first finding a spanning tree on the irregular topology and then applying our techniques to perform pipelined broadcast.

Acknowledgment

This research is supported in part by National Science Foundation grants CCF-0342540, CNS-0551555, and CCF-0541096.

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.” *The 19th IEEE Int'l Parallel and Distributed Processing Symposium*, page 80b, 2005.

[3] K.W. Cameron and X.-H. Sun, “Quantifying Locality Effect in Data Access Delay: Memory LogP,” *IEEE Int'l Parallel and Distributed Processing Symposium* (IPDPS), page 48b, 2003.

[4] K.W. Cameron, R. Ge, and X. -H. Sun, “Log$_n$P and log$_3$P: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems,” *IEEE Trans. on Computers*, 56(3):314-327, 2007.

[5] 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.

[6] D. Culler, et al., "LogP: Towards a Realistic Model of Parallel Computation." *Proceedings of the fourth ACM SIGPLAN Symposium on Principle and Practice of Parallel Programmingn* (PPoPP), pages 1-12, 1993.

[7] A. Faraj and X. Yuan, "Automatic Generation and Tuning of MPI Collective Communication Routines," the *19th ACM International Conference on Supercomputing*, pages 393-402, 2005.

[8] 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.

[9] A. Faraj, P. Patarasuk, and X. Yuan, "A Study of Process Arrival Patterns for MPI Collective Operations." *International Journal of Parallel Programming*, accepted for publication.

[10] A. Faraj, P. Patarasuk and X. Yuan, "Bandwidth Efficient All-to-all Broadcast on Switched Clusters." *International Journal of Parallel Programming*, accepted for publication.

[11] J.Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. Dongarra, "Performance Analysis of MPI Collective Operations." *The 19th IEEE International Parallel and Distributed Processing Symposium*, pages 8-8, 2005.

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

[13] 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.

[14] R. Kesavan and D. K. Panda, "Optimal Multicast with Packetization and Network Interface Support," *Proceedings of International Conference on Parallel Processing*, pages 370-377, 1997.

[15] T. Kielmann, H. E. Bal, and K. Verstoep, "Fast Measurement of LogP Parameters for Message Passing Platforms," *Proceedings of 2000 IPDPS Workshop on Parallel and Distributed Processing*, Pages 1176-1183, Cancun, Mexico, May 2000.

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

[17] 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.

[18] P.K. McKinley, H. Xu, A. Esfahanian and L.M. Ni, “Unicast-Based Multicast Communication in Wormhole-Routed Networks.” *IEEE Trans. on Parallel and Distributed Systems*, 5(12):1252-1264, Dec. 1994.

[19] 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.

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

[21] A Proskurowski, “Minimum Broadcast Trees.” *IEEE Trans. on Computers*, 30:363-366, 1981.

[22] P. Sanders and J.F. Sibeyn, “A Bandwidth Latency Tradeoff for Broadcast and Reduction.” *Information Processing Letters*, 86(1):33-38, 2003.

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

[24] Andrew Tanenbaum, “Computer Networks”, 4th Edition, 2004.

[25] J.-Y. Tien, C.-T. Ho, and W.-P Yang, “Broadcasting on Incomplete Hypercubes.” *IEEE Transactions on Computers*, 42(11):1393-1398, 1993.

[26] J. L. Traff and A. Ripke, “Optimal Broadcast for Fully Connected Networks.” *Proceedings of High-Performance Computing and Communication* (HPCC-05), pages 45-56, 2005.

[27] J.L Traff and A. Ripke, “An Optimal Broadcast Algorithm Adapted to SMP-Clusters,” *EURO PVM/MPI*, pages 48-56, 2005.

[28] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra, “Automatically Tuned Collective Communications,” In *Proceedings of SC’00: High Performance Networking and Computing* (CDROM proceeding), 2000.

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

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