A Stable Broadcast Algorithm

Kei Takahashi, Hideo Saito, Takeshi Shibata and Kenjiro Taura
The University of Tokyo
Email: {kay, h_saito, shibata, tau}@logos.ic.i.u-tokyo.ac.jp

Abstract

Distributing large data to many nodes, known as a broadcast or a multicast, is an important operation in parallel and distributed computing. Most previous broadcast algorithms explicitly or implicitly try to deliver data to all nodes in the same rate. This assumption is reasonable for homogeneous environments where all nodes have similar receiving capabilities. However, when nodes have various receiving capabilities, nodes with slow-receiving capabilities slow down the entire receiving bandwidth in these algorithms. In such settings, each node desires to receive data at its largest possible bandwidth and to start computation as soon as it receives the data.

In this paper, we propose to say a broadcast is stable *when the bandwidth to a node is never sacrificed by the presence of other, possibly slow, receiving nodes, and proposes the stability as a desired property of broadcast algorithms. In addition, we show a simple and efficient stable broadcast when the topology among nodes is a tree and each link has a symmetric bandwidth. This work improves upon previously proposed algorithms such as FPFR and Balanced Multicasting. For general graphs, it outperforms them when the network is heterogeneous and for trees, our algorithm is proved to be stable and optimal.*

In a real environment with 100 machines in 4 clusters, our scheme achieved 2.1 to 2.6 times aggregate bandwidth compared to the best result in the other algorithms. We also demonstrated the stability by adding a slow node to a broadcast. Some simulations also showed that our algorithm also performs well in many bandwidth distributions.

1 Introduction

There are growing demands to effectively execute data intensive applications in distributed, wide-area environments. Since data transfers occupy a considerable part of the total processing time in the execution of such applications, efficient data transfer techniques are required.

One of the most common, practical transfer problems is broadcast. In a broadcast, one or more *source* nodes have data that need to be copied to some *destination* nodes. Various algorithms have been proposed to optimize broadcasts according to completion time and aggregate bandwidth (the sum of the bandwidths of all nodes) [1, 3, 4, 5, 6, 2].

Such existing broadcast algorithms have a few shortcomings, especially for data intensive applications executed in distributed, wide-area environments. One is that they are not *stable*, in that the addition of a new node to an existing broadcast can potentially lower the bandwidth of a node already participating in the broadcast. Another shortcoming of existing broadcast algorithms is that they are not *optimal* in terms of aggregate bandwidth. Even though many existing algorithms do optimize for aggregate bandwidth, they do so under the assumption that all nodes have the same incoming bandwidth. As a result, they do not explore the possibility of different nodes receiving data at different rates, missing the opportunity of having nodes that receive data early start computation without waiting for other nodes. In wide-area environments where link speeds can vary a great amount, it is important to prevent nodes with narrow links from slowing down other nodes.

In this paper, we propose a broadcast algorithm that uses bandwidth-annotated topology information to optimize for aggregate bandwidth. Our algorithm improves an existing algorithm called FPFR [4], and performs broadcasts with higher aggregate bandwidth than FPFR for any graph topology. Moreover, for tree topologies, we prove that our algorithm satisfies the following two properties:

- *Stability:* Adding a node to an existing broadcast does not lower the incoming bandwidth of any node already participating in the broadcast.
- *Optimality:* A broadcast performed using our algorithm achieves the maximum possible aggregate incoming bandwidth.

While our algorithm works well with any graph topology, we use tree topologies for simplicity in the exposition. To evaluate our algorithm, we used the topology inference

tool developed by Shirai et al. [7] to obtain a tree topology, and performed both simulations and real-machine experiments.

The rest of this paper is organized as follows. In section 2, some previous broadcast techniques are explained. Our algorithm as well as proof of optimality and stability is shown in Section 3. Section 4 depicts the experiments and evaluation of our algorithm in the real environment, and we conclude the paper in Section 5.

2 Related Work

In this section, we show some existing broadcast techniques for large messages. Although much research has been done to improve aggregate bandwidth, none has been evaluated in terms of stability.

2.1 Topology-unaware Broadcast

The simplest broadcast algorithm is *flat-tree*, in which a source node directly sends data to all the destinations. As this algorithm does not perform well because of the bottleneck at the source, some broadcast algorithms have been invented.

In *FastReplica* [1], the source splits data into small pieces and sends each piece to a different destination. Then, the destinations construct a ring connection, exchange pieces and finally obtain the whole data. While the bottleneck at the source is solved in this method, congestions are possibly induced by using the same link many times. For example, when a certain link is used by N transfers, the bandwidth is reduced to $1/N$.

BitTorrent [6, 9] adaptively improve the transfer graph, by changing the relaying network. First, data are divided into small pieces so that each node is allowed to receive them in an arbitrary order. Each mode randomly chooses its parent to construct a transfer network to exchange pirces between them, and the parent is periodically changed. When a node is chosen as a parent by more than five nodes, five nodes that offers more bandwidth to other nodes are chosen. As a result, nodes with a broad link are more likely to relay data so that the aggregate bandwidth is increased.

MOB [2] improves BitTorrent for multi cluster environments. In MOB, chunks are not transferred through a global link many times. Since it avoids transferred global network many times, transfers become more efficient.

While these algorithms achieves good performance when topology and bandwidth are unknown, the schedule obtained by this algorithm cannot always avoid link sharings. Since each node relays data to multiple (typically five) nodes, the bandwidth may be diminished in these branches. In addition, since it randomly changes the relaying structure, a good schedule is obtained only after long time.

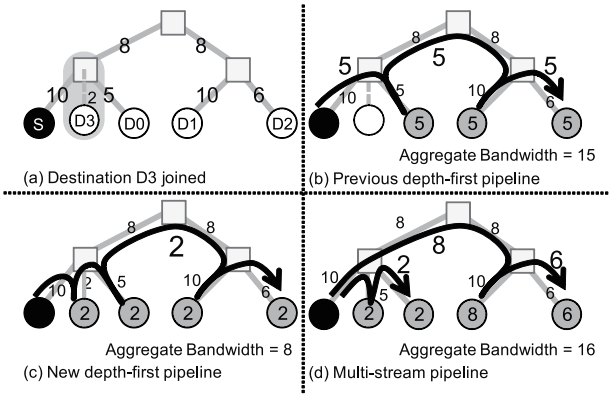

Figure 1. Effect of adding slow nodes

2.2 Topology-aware Broadcast

Some methods obtain a good schedule from the beginning by using network topology information. Karonis et al. proposed to use a hierarchical tree that describes the real network topology [5]. A broadcast is performed by tracing the tree: a node relays data to all its children. While it can avoid self-induced congestions, many branches in the relaying plan diminish the aggregate bandwidth. If a node has N children, these children can only receive $1/N$ bandwidth each.

In a depth-first pipeline method [7], each node relays data to only one node. Thie method constructs a pipeline by tracing a topology in a depth-first manner. In this pipeline, every node can receive as much bandwidth as the source sends so the slowest node can receive the largest possible bandwidth.

However, the depth-first pipeline cannot achieve good performance in terms of aggregate bandwidth for a heterogeneous network. Figure 1 illustrates a situation in which new node D_3 with small bandwidth joins the broadcast. One source S holds the original data, three destinations $D_0, D_1 and D_2$ need them. The label on each link shows the bidirectional bandwidth. Originally the aggregate bandwidth of D_0, D_1 and D_2 is 15. In the new pipeline, however, the aggregate bandwidth has dropped to 6. This deterioration can be avoided by concurrently using multiple pipelines. In *Multi-stream Pipeline* in Figure 1, the aggregate bandwidth is improved to 16.

Another approach uses *Dijkstra* method to construct a relaying structure [8]. In this method, two node sets are maintained: one is *reachable-set*, a set of nodes that are already connected to the source, and another is *unrechable-set*, a set of nodes that are not already connected to the source. Based on bandwidth-annotated topology, one node that is reachable with the largest bandwidth from the *reachable-*

set is selected and added to the *reachable-set* at a time. This process is repeated until all the destinations are connected to the source.

In this algorithm, fast nodes are more likely to be added in earlier phases. As a result, adding a slow node to a broadcast does not deteiorate aggregate bandwidth. However, this algorithm cannot avoid link sharings. Experiments show that depth-first pipeline achieves larger aggregate bandwidth in many settings [4].

2.3 Multistream Broadcast

The *Fast Parallel File Replication* (FPFR) tool [4] uses bandwidth-annotated network topology information to perform efficient broadcasts. It iteratively constructs multiple spanning trees, and uses them concurrently. By utilizing available link bandwidth more effectively, FPFR achieves higher aggregate bandwidth than methods that only use a single stream.

Let the network topology and available bandwidth of each link be given in advance. The first tree is built based on the original bandwidth values. Several ways are tried to construct spanning trees, and *depth-first* method, which traces the destinations from the source in a depth-first manner, achieved the best performance. After the first tree was formed, the bandwidth value for the tree is subtracted from the original bandwidth of all the links. The following trees are repeatedly built with the subtracted bandwidth values until no spanning tree can be created. These multiple spanning trees are concurrently used with the planned bandwidth. Each tree sends different pieces of the file, and finally every destination receives the whole file.

FPFR has achieved better performance compared to a method that uses only one tree. *Balanced Multicasting* [3] has further improved FPFR by using linear programming to maximize the aggregate bandwidth. However, we think FPFR has problems with stability. Since FPFR only uses spanning trees, a node with narrow bandwidth limits the aggregate bandwidth of the other nodes. Especially in a case of a tree topology, it only outputs one pipeline. Since the throughput of a pipeline is limited by the narrowest link in it, the stability is not achieved.

3 Algorithm

3.1 Basic Idea

Like many broadcast algorithms referred in section 2, our algorithm takes as input a bandwidth-annotated network topology, a single source node, and multiple destination nodes. From this information, it generates *transfer trees* labeled with its throughput (the bandwidth it consumes). In this model, a set of transfer trees is said *feasible* when the total throughput used on each link is within its capacity. In practical terms, we model the problem assuming that each switch has a sufficiently strong backplane so the transfer rate is only limited by capacities of hosts or individual ports of switches, not by internal switching capacities. In addition, we assume each link is full-duplex: link traffics in two directions do not affect each other.

A broadcast algorithm is defined to be *stable* when adding more nodes to destinations never decrease bandwidths delivered to the original nodes. Formally, for any pair of destination sets D and E such that $D \subset E$, and for any node $n \in D$, the bandwidth to n generated with destinations E is never smaller than that generated with destinations D.

We propose a broadcast algorithm which achieves more aggregate bandwidth than *FPFR* and *Balanced Multicasting*. Besides, the algorithm is shown to be stable and optimal with respect to aggregate bandwidth when the network satisfies the following conditions.

- The topology is a tree.
- Each link is bidirectional and has a symmetric capacity in both directions.

While many multi-cluster environments and ISP's networks satisfy these conditions, some WAN environments do not. However, some graph topologies are possibly approximated to trees for a broadcast problem. For example, when a loop exists in a WAN but it consists of links with larger bandwidth than bandwidth of LAN links, we consider the loop as one switch with large-enough backplane.

The basic idea of the algorithm is similar to FPFR and Balanced Multicasting, in that it builds multiple transfer trees and sends data fragments through them. The main difference is that while FPFR stops building transfer trees as soon as the narrowest link saturates, our algorithm continues to make *partial trees* that involve only a subset of the destinations. Such trees play an important role to guarantee that our algorithm is stable, or that a node connected to the source with higher bandwidth will receive data with its highest bandwidth. Transfer scheduling becomes slightly more complex than FPFR to guarantee that all nodes receive all data despite that some transfer trees only contain a subset of the destinations. Details are shown in Section 3.3.

3.2 Constructing Transfer Trees

The algorithm to create transfer trees is shown in Algorithm 1. The network topology $\mathcal{T}$ is given in advance as a directed graph, which consists of computational nodes as leaf nodes and switches as intermediate nodes. Each link is bidirectional and modeled as two separate edges of the graph. In addition, the bandwidth (capacity) $B_0(e)$ is given for each link e.

Algorithm 1 Constructing transfer trees

Require: $\mathcal{T}$ is a network topology. $B_0(e)$ is the bandwidth for link e. s is the source node and D_0 is a set of destinations.

```
B := B_0; T := ∅;
while TRUE do
  t := the tree obtained by tracing destinations in depth-first manner from s;
  if t contains no destination then
    break
  end if
  u := the bandwidth of the narrowest link in t;
  T := T ∪ {⟨t, u⟩};
  for each link e in t do
    B(e) := B(e) − u;
  end for
end while
return T
```

Algorithm 2 Sending data via transfer trees

Require: $t_1, \ldots, t_n$ are the transfer trees returned by the construction algorithm (t_i is made by the i-th iteration). u_i is the throughput of t_i. D is the data to broadcast.

```
for k := n downto 1 do
  R := data nodes in t_k have not yet received;
  send R through t_1, ..., t_k, allocating to t_i the amount of data proportional to u_i;
end for
```

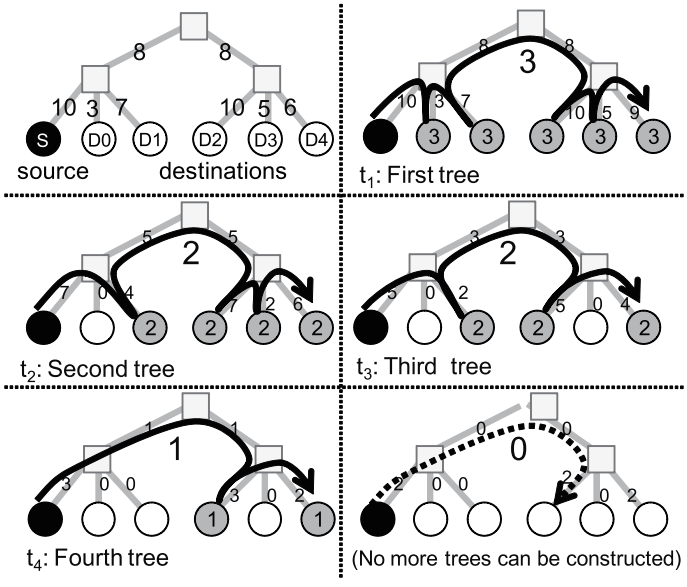

Figure 2. The algorithm to build transfer trees

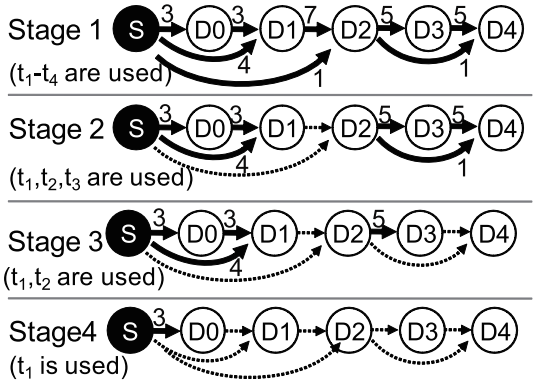

Figure 3. Execution of the broadcast

Like FPFR, the algorithm repeats building transfer trees. The first tree is constructed by visiting all the destinations from the source node in a depth-first manner. As a result, it creates one tree that connects the source and all the destinations. The throughput of this tree is set to the capacity of the narrowest link in it.

After the first tree is obtained, bandwidth of each link is updated for the construction of the next tree. The throughput allocated to the first tree is subtracted from the original lik bandwidth, and an edge is removed when its bandwidth becomes zero.

The second tree is constructed with this reduced bandwidth map using the same depth-first traversal. In this time, it may not be possible to reach every destination from the source as some edges have been removed. If that happens, we connect only destinations reachable from the source. For a tree network in particular, at least one node becomes unreachable. The throughput of the second tree is again set to the capacity of its narrowest link, and is then subtracted from each link used by the tree. We repeat this procedure until no nodes become reachable from the source. Our algorithm can also utilize multiple replicas by creating a set of trees from each replica one by one.

Figure 2 illustrates a process to build transfer trees. One source S and destinations $D_0 \ldots D_4$ are engaged in this broadcast. The first tree t_1 connects all the destinations $D_0 \ldots D_4$, and its throughput is limited to 3 by the link connected to D_0. The second tree t_2 is built after subtracting the throughput 3 from each link used by the first tree. Tree t_2 connects $D_1 \ldots D_4$ with the throughput of 2. After the construction of the third and the fourth trees, the available bandwidth map does not allow to construct any transfer from the source. Consequently, four trees are obtained.

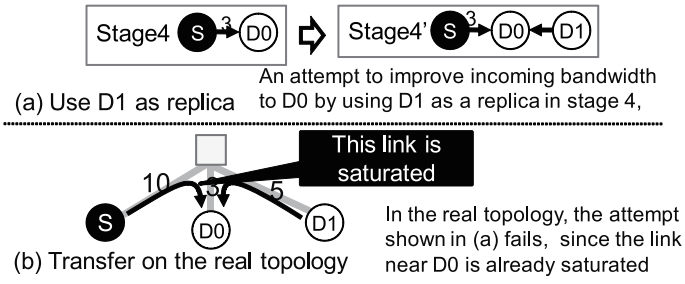

Figure 4. Use a fast node as replica

3.3 Transfer Using Multiple Trees

Once the transfer trees have been constructed, data fragments are sent through them. To achieve the claimed stability and optimality, all the trees must be effectively utilized. Algorithm 2 shows how the transfer is done.

Data are sent in n *stages*, with each stage using a different set of transfer trees. Let t_i be the transfer tree made by the i-th iteration of the construction algorithm ($i = 1, \ldots, n$) and $N(t)$ be the set of destinations involved in tree t. Note that the relation $N(t_n) \subset N(t_{n-1}) \subset \cdots \subset N(t_1)$ holds.

In the first stage, the entire data are partitioned into fragments and sent using *all* transfer trees. Data must be allocated to trees so that they finish the transfer at the same time. In principle, this can be achieved by allocating to each tree the amount of data proportional to its throughput. In the actual implementation, we achieve this by dividing data into small chunks, then allocating data dynamically to trees. Each transfer edge is implemented by a TCP connection and we send a chunk via a connection only when the connection is writable (i.e., writing to it does not block). At the end of the first stage, nodes in $N(t_n)$ will have obtained the entire data and will not have to receive any more data. Other nodes only get a part of the data; the remaining will be delivered in later stages.

In the second stage, data, that the nodes in $N(t_{n-1})$ have just missed in the first stage (i.e., data sent through t_n), are sent by using $t_1, \ldots, t_{n-1}$, with the same policy for allocating data to individual trees. The second stage will deliver all the data to nodes involved in t_{n-1}. Similarly in the third stage, data that the nodes in $N(t_{n-2})$ have not yet received will be sent via $t_1, \ldots, t_{n-2}$ so they will have all data in the end of the stage. We repeat this sequence until all the destination nodes receive all the data.

Although data are sent using multiple trees, the trees utilize the same TCP connections. In addition, nodes will not send any duplicate chunks that have already received by the peer in the real implementation. Figure 3 illustrates how the transfers are performed with trees constructed in the example of Figure 2. The number written beside each arrow shows the bandwidth of each flow. For example, node D_1 receives bandwidth 3 from D_0 and 4 from S. Data delivered from D_0 correspond to t_1, and data from S corresponds to t_1 and t_2. D_1 receives the combined bandwidth 7, which is the maximum possible bandwidth for D_1 to receive data from S.

Common practice has been to receive different chunks from multiple replicas. For example, Figure 4(a) depicts an attempt to improve incoming bandwidth to D_0 by transferring data both from S and from D_1. However, in a tree topology whose links have the same bandwidth in both directions, it is not possible to improve the incoming bandwidth of remaining nodes by using these replicas. Figure 4(b) shows that in the real topology in Figure 3, the bottleneck lies on the link connected to D_0. Otherwise, D_0 should have been included in the other trees such as t_2 and t_3.

3.4 Stability and Optimality for Tree Topologies

We show that the algorithm introduced in the Section 3 is stable and optimal in the sense of aggregate bandwidth for a tree topology whose links have the same bandwidth in two directions. The proof consists of three parts. First, we show that we can treat each link as an undirected link during tree construction. Then, we introduce some properties about pipeline transfers, and finally, we prove that the set of trees deliver to a node the maximum amount of data that can be received by the node.

3.4.1 Preparation: Property from Symmetric Link

Basically, we need to treat each link as a directed link during the construction of the trees. The bandwidths of the two directed links on the same path are separately calculated, and the link is added separately. However, when each link has the same bandwidth in the two directions, we can treat the two directed links as one link, whose bandwidth is given by the smaller bandwidth of the two directional links. The reason is shown as follows.

We have a tree topology that contains a source s and some destinations. A bidirectional link e in the network has two subtrees connected to both ends. Since the network is a tree, the source node s is contained in either of the two subtrees. Assume that the subtree T does not contain the source node s.

The link e can be split into two directional links: one heading for T and the other away from it. Let the term *forward link* e_f and *backward link* e_b denote the former and the latter link, respectively. This is shown in Figure 5(a). In a tree network, the removal of e from the topology discon-

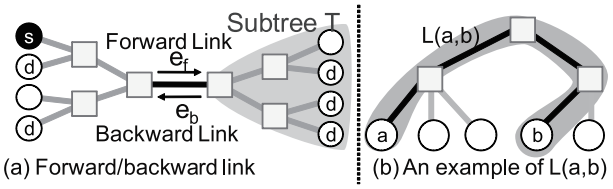

Figure 5. Links in a tree symmetric network

nects every node in T and s. Therefore, if e_f is removed, we can create no route from s to any node in T.

Assume that the backward link e_b is adopted by a tree. Since the tree uses e_b, it has reached one of the destinations in T. Without e_f, we cannot create a route from s to any node in T. Thus, when a tree contains e_b, it also employs the corresponding forward link e_f. Therefore, $B_k(e_b) - B_{k-1}(e_b) \leq B_k(e_f) - B_{k-1}(e_f)$ holds, where $B_k(e)$ denotes the available bandwidth used in the construction of the (k)th tree. Since $B_0(e_b) = B_0(e_f)$ holds from the assumption, $B_k(e_b) \leq B_k(e_f)$ is lead.

Since the bottleneck in the tree always lies on the forward link, during the construction of the trees, we only need to use the bandwidth of the forward link.

3.4.2 Properties for Single Transfer

For a tree topology, there is exactly one route that connects two nodes a and b using the minimum number of links. Let $L(a, b)$ denote this set of links, which is shown in Figure 5(b). Since the removal of any link in $L(a, b)$ from the tree disconnects a and b, any route from a to b contains all the links in $L(a, b)$. Although some route may use a link multiple times, the throughput is maximized when each link in $L(a, b)$ is only used once Let $v_k(a, b)$ denote this throughput when the bandwidth is B_k:

$$v_k(a, b) = \min_{e \in L(a,b)} (B_k(e)) \tag{1}$$

With a set of links $(L(a, b) \cup L(b, c))$, we can construct a path from a to c via b. Therefore, the following relation holds:

$$(L(a, b) \cup L(b, c)) \supset L(a, c) \tag{2}$$

Assume that we iteratively construct a total of n trees. From the source s, a tree traces some destinations in a depth-first path. Let D_k denote the set of destinations included in the (k)th depth-first tree, and P_k denote the set of links used in the (k)th tree.

Using the above notations, the following equations (3) and (4) hold. Because the graph is a tree,

$$P_k = \bigcup_{d \in D_k} L(s, d). \tag{3}$$

Because the algorithm traces the path from s,

$$d \notin D_k \text{ iff } v_k(s, d) = 0. \tag{4}$$

3.4.3 Proof of Stability and Optimality

Let $d_{k,0} \ldots d_{k,n-1}$ denote the nodes in the destination set D_k, which are ordered to satisfy $v_k(s, d_{k,i}) \leq v_k(s, d_{k,i+1})$. Let u_k denote the throughput of the tree P_k. We get the following:

$$u_k = \min_{e \in P_k}(B_k(e)) = \min_{1 \leq i \leq n} (v_k(s, d_{k,i})) = v_k(s, d_{k,1}). \tag{5}$$

The following equation holds.

$$D_{k+1} = D_k \setminus A_k, \tag{6}$$

where $A_k = \{d \in D_k | v_k(s, d) = v_k(s, d_{k,1})\}$.

Proof. (Proof of (6))
From the definition of B_k,

$$B_{k+1}(e) = \begin{cases} B_k(e) - u_k & \text{if } e \in P_k, \\ B_k(e) & \text{otherwise.} \end{cases}$$

Thus, from (4),

$$v_{k+1}(s, d) = \begin{cases} v_k(s, d) - u_k & \text{if } d \in D_k, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

If $d \in A_k$, namely $v_k(s, d) = v_k(s, d_{k,1})$, then $v_{k+1}(s, d) = u_k - u_k = 0$ from (5). If $d \in D_k \setminus A_k$, namely $v_k(s, d) > v_k(s, d_{k,1})$, then $v_{k+1}(s, d) = v_k(s, d) - u_k > 0$.

From (4), $D_{k+1} = \{d \in D_1 | v_{k+1}(s, d) \neq 0\} = D_k \setminus A_k$. □

From (6), a destination d in A_k receives data from the (1)st to the (k)th trees. Let $w(d)$ denote the total bandwidth d receives from the trees, that is,

$$w(d) = \sum_{1 \leq i \leq k} u_k. \tag{8}$$

From (7), the following holds:

$$\begin{aligned} u_k = v_k(s, d_{k,1}) &= v_{k-1}(s, d_{k,1}) - u_{k-1} \\ &= v_1(s, d_{k,1}) - \sum_{1 \leq i < k} u_i. \end{aligned}$$

Thus

$$v_1(s, d_{k,1}) = \sum_{1 \leq i \leq k} u_i. \tag{9}$$

From (8) and (9), we get the following:

$$\forall d \in A_k, w(d) = v_1(s, d_{k,1}) \tag{10}$$

This means that the trees deliver to d the same amount of data as in the direct transfer from s to d in the condition that there is no other traffic. From 1, this is the maximum amount of data d can receive.

In addition, from the definition of A_k and (6), every destination is included in one of $A_1, \ldots, A_n$, where n is the number of the trees.

Consequently, it is proved that our trees deliver to a node the maximum amount of data that can be received by the node. It is also stable because the receiving bandwidth does not depend on other destinations.

3.5 Improvement for Graph Topologies

In a graph topology, no algorithms can achieve both stability and optimality together. However, our broadcast algorithm improves aggregate bandwidth than previously proposed algorithm such as FPFR. While FPFR and Balanced Multicasting create only spanning trees that connect all the destinations, our algorithm constructs the same set of spanning trees and a set of partial trees that connect part of the destinations. Since the partial trees use surplus bandwidth that was not consumed by the spanning trees, the aggregate bandwidth of our algorithm is never less than that of FPFR. By using linear programming for these trees to optimize the aggregate bandwidth, our algorithm also improves the aggregate bandwidth of Balanced Multicasting.

4 Evaluation

We implemented the algorithm, and evaluated it in both a simulation and a real environment. Five algorithms are compared:

Ours: This is the algorithm we proposed. Multiple spanning and partial trees are iteratively constructed in a depth-first manner, and transfers are performed by using them in parallel.

Depth-First (FPFR [4]-like): The algorithm constructs multiple spanning trees in a depth-first manner [3, 4].

Dijkstra: This algorithm iteratively builds a tree in a greedy manner. Pick one unreached destination that can be reached in the maximum possible bandwidth from reached nodes. The method is explained in [4], and a similar method is proposed in [8].

Random Pipeline: This algorithm randomly creates a tree. One node is randomly chosen from all the unreached destinations at a time. A total of 100 candidates are generated, and one with the largest aggregate bandwidth is chosen.

Flat Tree: The source directly sends the data to all the destinations.

Except for *Flat-tree*, every algorithm requires a bandwidth-annotated topology.

4.1 Simulation

A simulator has been implemented to evaluate broadcast algorithms. We make the throughputs of the machines and switches large enough compared to the link bandwidths, so that they do not become bottlenecks. We used a tree topology with 400 nodes, taken from the real environment. During the experiment, three different bandwidth distributions were tested:

Uniform Random: A uniform random value is assigned to each link to see general behavior of these algorithms. Both *low-variance* (from 500 to 1000) and *high-variance* (from 100 to 1000) conditions are tested.

Mixed Fast and Slow Links: While 80% of the links has 1000Mbps bandwidth, the other 20% has 100Mbps bandwidth. It describes a situation that fast and slow devices are mixed.

Random Inter-Cluster: In this condition, inter-switch links are assigned random distribution (from 100 to 1000).

The simulation is performed 10 times for the same algorithm, bandwidth variance and number of destinations. For each conditions, we tested two settings with each link: one has the same bandwidth in two directions (symmetric), the other does not (assymmetric).

The result is shown in Figure 6. The vertical axis shows the relative aggregate bandwidth of the *FlatTree algorithm*. Our algorithm took only 2 milliseconds to induce the schedule. Our method achieved the best performance in every symmetric condition. As shown in Figure 6(c), the improvement is especially notable when fast and slow links are mixed. In this case, the performance of the other algorithms is dropped because of the lack of stability. In an asymmetric network, the performance of our algorithm is the best except for one case, Figure 6(d). Even in this case, the difference is only 3%. As a result, the superiority of our algorithm is confirmed.

4.2 Real Machine Experiment

We also performed some experiments using a real machine environment. This environment had the tree topology shown in Figure 8, with 105 nodes in 4 clusters. The bandwidth of the links is also shown in Figure 8. We performed broadcasts among 10, 47 and 105 nodes. Table 1

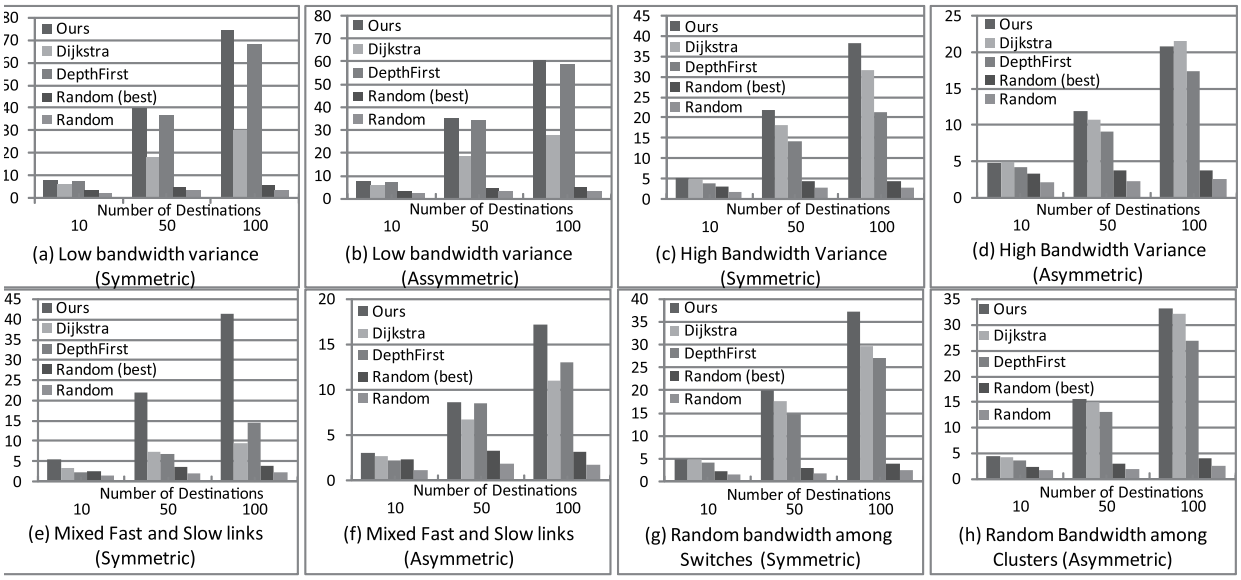

Figure 6. Aggregate bandwidth of each algorithm (Vertical axis: relative aggregate bandwidth comapring to the *FlatTree* algorithm)

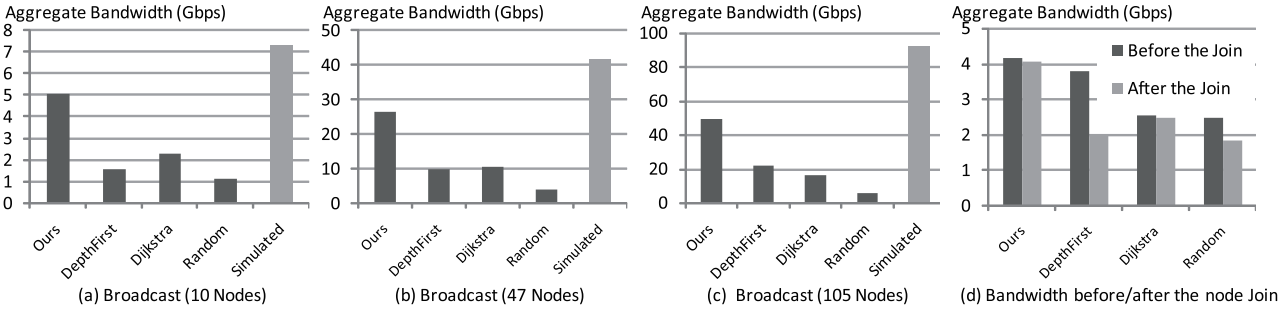

Figure 7. Real machine experiments

Table 1. Number of nodes in each clusters

Number of Destinations	A	B	C	D
105	59	9	35	2
47	30	1	16	1
10	6	1	2	1

shows the number of nodes from each cluster. Each condition was tested four times, and the best aggregate bandwidth was taken.

Figure 7 illustrates the results. Since the environment is heterogeneous, the performance was improved significantly by our algorithm. The aggregate bandwidth increased by 2.1 to 2.6 times compared to the best result in the other algorithms. However, the bandwidth was 30% to 45% worse than the optimal value predicted by simulation. An investigation revealed that our assumption that each link is bidirectional and thus has an independent (full-duplex) bandwidth in each direction was subtly violated, because transferring data in both directions saturated CPUs. While each node could send or receive 900Mbps independently, it could only send and receive 750Mbps when it was simultaneously sending and receiving. In the simulation, the link adjoining such a node was modeled as having 900Mbps in each direction, but in reality each only had 750Mbps when it was used to relay data.

To demonstrate the stability of our broadcast, another test was performed. In this experiment, a node with small bandwidth joined a 9-node broadcast. Figure 7 (d) shows

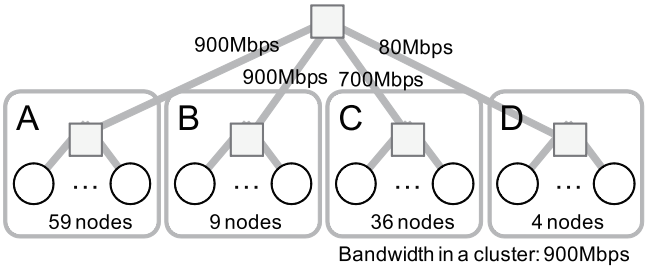

Figure 8. The real environment topology

the change in the aggregate bandwidth of the older 9 nodes for four broadcast algorithms. While the aggregate bandwidth dropped to 52% in *depth-first*, which works the same as FPFR for a tree, the deterioration in our algorithm was only 2.5%. Although the drop in the other two algorithms were also small, our algorithm yielded 1.5 times aggregate bandwidth of them.

5 Conclusion

In this paper, we introduced the notion of *stability* in broadcasts, and proposed a simple and efficient broadcast for heterogeneous environments.

Our broadcast improves a previously proposed class of broadcast algorithms that include FPFR and Balanced Multicasting, focusing on the fact that each node should receive data as fast as possible and start computation without waiting for other nodes. Like FPFR and Balanced Multicasting, our broadcast achieves high bandwidth by forwarding data along multiple spanning trees. In addition, our broadcast avoids the effect of nodes with narrow bandwidth by forwarding data along multiple partial trees. While the slowest node only receives data from the spanning trees, faster nodes receive data both from the spanning trees and from the partial trees.

For general graphs, our broadcast will always outperform FPFR and Balanced Multicasting, and for trees, we proved that it is *stable* as well as *optimal*. In a real environment with 100 machines in 4 clusters, our scheme achieved 2.1 to 2.6 times aggregate bandwidth compared to the best result in the other algorithms. We also demonstrated the stability by adding a slow node to a broadcast. While the aggregate bandwidth dropped to 52% in *depth-first*, which works the same as FPFR for a tree, the deterioration in our algorithm was only 2.5%. Some simulations also showed that our algorithm also performs well in many bandwidth distributions.

In the future, we would like to invent an algorithm that maximizes aggregate bandwidth in arbitrary graph networks. Although no algorithm can achieve both stability and optimality in a graph topology, it is still important to deliver as much data as possible to each host in a broadcast, in order to effectively execute data-intensive applications.

Another future work is about a method that can adaptively improve the transfer schedule. In a WAN environment, bandwidth sometimes fluctuates because of other traffic. To cope with this fluctuation, we will utilize dynamically measured bandwidth during a transfer, and plan the transfer schedule again.

Acknowledgment

This project is partly supported by the Grant-for-Aid for Scientific Research on Priority Areas from Society for the Promotion of Science (JSPS).

References

[1] L. Cherkasova and J. Lee. FastReplica: Efficient Large File Distribution within Content Delivery Networks. In *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2003.

[2] M. den Burger and T. Kielmann. MOB: Zero-configuration, High-throughput Multicasting for Grid Applications. In *IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 159–168, June 2007.

[3] M. den Burger, T. Kielmann, and H. E. Bal. Balanced Multicasting: High-throughput Communication for Grid Applications. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Nov. 2005.

[4] R. Izmailov, S. Ganguly, and N. Tu. Fast Parallel File Replication in Data Grid. In *Future of Grid Data Environments Workshop (GGF-10)*, Mar. 2004.

[5] N. T. Karonis, B. R. de Supinski, W. G. I. Foster, E. Lusk, and J. Bresnahan. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. pages 377–384, May 2000.

[6] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 367–378, Aug. 2004.

[7] T. Shirai, H. Saito, and K. Taura. A Fast Topology Inference — A building block for network-aware parallel computing. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 11–21, June 2007.

[8] R.-C. Wang, S.-L. Wu, and R.-S. Chang. A novel data grid coherence protocol using pipeline-based aggressive copy method. In *GPC*, pages 484–495, 2007.

[9] B. Wei, G. Fedak, and F. Cappello. Scheduling independent tasks sharing large data distributed with bittorrent. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 219–226, Washington, DC, USA, 2005. IEEE Computer Society.