Optimal Distribution Tree for Internet Streaming Media*

Min Sik Kim Simon S. Lam Dong-Young Lee

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712–1188, USA
{minskim,lam,dylee}@cs.utexas.edu

TR–02–48
September 2002

Abstract

Internet radio and television stations require significant bandwidth to support delivery of high quality audio and video streams to a large number of receivers. IP multicast is an appropriate delivery model for these applications. However, widespread deployment of IP multicast on the Internet is unlikely in the near future. An alternative is to build a multicast tree in the application layer. Previous studies have addressed tree construction in the application layer. However, most of them focus on reducing delay. Few systems have been designed to achieve a high throughput for bandwidth-intensive applications. In this paper, we present a distributed algorithm to build an application-layer tree. We prove that our algorithm finds a tree such that the average incoming rate of receivers in the tree is maximized (under certain network model assumptions). We also describe protocols that implement the algorithm. For implementation on the Internet, there is a tradeoff between the overhead of available bandwidth measurements and fast convergence to the optimal tree. This tradeoff can be controlled by tuning some parameters in our protocols. Our protocols are also designed to maintain a small number, $O(\log n)$, of soft states per node to adapt to network changes and node failures.

Keywords: optimal distribution tree, streaming media, application-layer multicast, peer-to-peer

1 Introduction

Internet radio and television stations have, in the past, been operated by companies with high-performance dedicated servers. The availability of broadband access and increasing computing performance of PCs have made it feasible for individuals to run their own radio stations. As a result, thousands of channels are serving multimedia on the Internet.[1]

These applications require one-way data transmission to a large number of receivers, for which IP multicast is an appropriate delivery model. The availability of IP multicast is, however, extremely limited, and unlikely to improve much in the near future. An alternative to IP multicast is end-system multicast. In end-system multicast, participants form an overlay distribution tree in the application layer and perform multicasting among themselves. The main advantage is that it does not require multicast support from the underlying network. The overlay multicast tree can be constructed on top of any network that provides a unicast transport service.

Many end-system multicast systems have been proposed for different target applications. Each of them has its own way to create a distribution tree. Of the ones that try to perform tree optimization, they

*Research sponsored by National Science Foundation grant no. ANI–9977267 and Texas Advanced Research Program grant no. 003658–0439–2001.

[1] See Icecast (http://yp.icecast.org/) and SHOUTcast (http://www.shoutcast.com/).

generally fall into one of two categories depending on which metric they emphasize in tree construction, i.e., reducing delay or increasing throughput.[2]

Consider a set of nodes (end systems) that form an overlay on the Internet. In systems with the goal of reducing delay [1, 3, 4, 14], a mesh consisting of all nodes and a subset of logical links connecting them are first constructed. Then the nodes measure Internet delays of the logical links, and run a routing algorithm, such as the distance vector algorithm, to find best paths from each node to others.

In one system with the goal of increasing throughput [11], logical links with high (available) bandwidth[3] are first chosen as edges of the distribution tree. Then the system keeps trying to increase the bandwidth between each pair of nodes by modifying the tree topology. Unlike systems with the goal of reducing delay, for which the distance vector algorithm is proven to lead to an optimal state, the proposal in [11] lacks an algorithmic method to achieve an optimal solution. In another proposal [5], a centralized algorithm was presented to compute, for a given graph, a "maximal bottleneck" spanning tree rooted at a given vertex.

Since increasing throughput is more important than reducing delay in one-way multimedia delivery, it is desirable to have a distributed algorithm that finds a tree with "maximal throughput." However, this is not a straightforward task due to the difficulties described below.

The first is the result of a fundamental limitation of today's Internet, namely: there is no simple mechanism to measure the bandwidth available to a flow between two nodes. Generally, many packets need to be sent to detect the congestion status of a path as well as how much bandwidth a flow can use without adversely affecting other flows. In other words, bandwidth measurement requires a lot more traffic than delay measurement in the Internet. Therefore, in designing the distributed algorithm, we should avoid measuring the bandwidth of too many logical links. Thus, the first difficulty we encounter is how to choose logical links that need to be measured. If we choose too few, we may be unable to find an optimal tree due to insufficient information. On the other hand, if we choose too many, there would be substantial measurement overhead on the network.

Another difficulty is node failures. Because end-system multicast depends on participating nodes, which are user machines, rather than routers, it is likely that many nodes leave the multicast group during a session. Losing some nodes would definitely change the optimal tree; thus the algorithm should be designed to be adaptive, with the ability to re-compute a new optimal tree without too much additional overhead.

In this paper, we first present a distributed algorithm that builds a tree in which the average receiving rate, computed over all receivers in the tree, is maximized. Convergence of the tree to an optimal tree is proved under certain network model assumptions. Protocols that implement our distributed algorithm are then designed to address the difficulties discussed above. In our protocols, each node measures bandwidth from at most $O(\log n)$ nodes. The distribution tree is continuously updated as it converges towards an optimal tree. When there is a node failure, our protocols will adapt and the distribution tree will start converging towards a new optimal tree.

We evaluated our algorithm experimentally by simulation. Our simulation results show that significant bandwidth gain is obtained within a relatively short time duration. The optimal tree derived achieves an average receiving rate (per receiver) as much as 30 times that of a random tree depending on the network configuration. The simulation results also demonstrate how the average receiving rate increases as the distribution tree evolves. For a topology consisting of 51 end hosts and 100 routers, it takes about eighty seconds to get close to the maximum. Considering the usual playback time of audio and video streams, we believe this is reasonably fast.

The remainder of this paper is organized as follows. We introduce our network model and assumptions in Section 2. In Section 3, we first present a centralized algorithm to find an optimal tree. We then present a distributed algorithm that is guaranteed to converge to a tree as good as the one found by the centralized algorithm. These results are stated as two theorems. In Section 4, we present protocols implementing the distributed algorithm and address various implementation issues in the Internet. An experimental evaluation of our algorithm is presented in Section 5. We conclude in Section 6.

[2] The throughput of a distribution tree is a notion we will make more precise later.

[3] For simplicity, we will use *bandwidth* and *available bandwidth* interchangeably.

2 Network Model

It is difficult to find a simple model capturing all aspects of the Internet. In building a streaming media distribution tree, however, our main concern is bandwidth. In other words, our goal is to find a tree that provides the largest (available) bandwidth we can utilize. Accordingly, we develop a network model to focus on that aspect.

Even when we limit our concern to bandwidth only, there are still many factors to be considered. Available bandwidth is determined by many parameters. In particular, the available bandwidth between two nodes is a function of the underlying Internet topology and existing traffic. Based upon the following observations, we abstract away detailed topology and traffic information in our network model.[4]

- Usually access links are bottlenecks causing congestion while backbone links are loss-free [16].
- An access link has incoming and outgoing bandwidths that do not affect each other.

An access link means a link that connects a host or its local area network to the network of its ISP. We use these observations to simplify our model. Since congestion occurs mainly on access links, we assume that the bandwidth available to a flow between two nodes is determined by the congestion status of the access links of the nodes. The links in between add delay, but do not limit the bandwidth of the flow. Based on these observations, we propose an abstract model.

2.1 Abstract Model

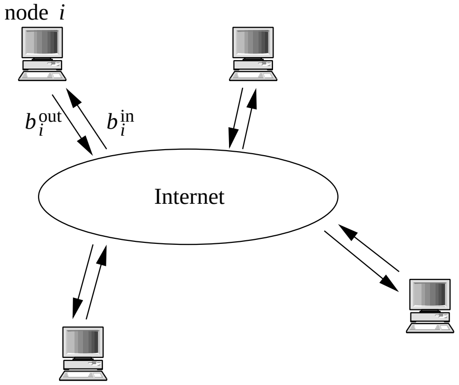

Figure 1: Network Model

A visual representation of our model is shown in Figure 1. A node is connected to the Internet through an access link, which has a pair of parameters: incoming and outgoing bandwidths. The incoming bandwidth of a node is the bandwidth from the ISP to the node, and the outgoing bandwidth is the bandwidth from the node to its ISP. In Figure 1, b_i^{in} represents the incoming bandwidth of the access link of node i, and b_i^{out} the outgoing bandwidth. A configuration of our network model is defined to be $M = (N, B)$, where N is a set of nodes and B is the set, $\{(b_i^{\text{in}}, b_i^{\text{out}}), i \in N\}$. N has $n+1$ elements: a sender and n receivers. For convenience in presenting algorithms, we assume $N = \{0, 1, 2, \ldots, n\}$, where 0 represents the sender, and $\{1, 2, \ldots, n\}$ receivers.

Consider a distribution tree consisting of the nodes in N. The root of the tree is node 0, the sender. An intermediate node in the tree has one incoming connection from its parent and one or more outgoing connections to its children. We assume that the outgoing link bandwidth is allocated equally to its children. Let c_i denote the number of children of node i. We make the following assumption on $b_{i,j}$, the *edge bandwidth* from node i to a child node j, for every edge in the distribution tree.

Edge Bandwidth Assumption *Each node i is characterized by b_i^{in} and b_i^{out} such that if node j is a child of node i in the tree, then $b_{i,j} = \min\left(\frac{1}{c_i} b_i^{\text{out}}, b_j^{\text{in}}\right)$, where $i = 0, 1, \ldots, n$, $j = 1, 2, \ldots, n$, and $i \neq j$.*

If backbone links are not congested, then the bottleneck between two nodes should be one of the access links at either end. Therefore, we abstract away Internet topology and traffic by this assumption, and consider only access link bandwidths in our abstract model. (This abstraction is used by our theorems in Section 3. In our protocol implementation, described in Section 4, $b_{i,j}$ is obtained by measuring the available bandwidth from node i to node j.)

The three quantities defined above are determined by access link characteristics. We define two more quantities in the context of a distribution tree. The *incoming (receiving) rate* of node i is defined to be the minimum of edge bandwidths on the path from the root node to node i:

[4] This abstraction is needed by our theorems in Section 3, but not by our protocol implementation in Section 4.

Variable	Description
b_i^{in}	incoming access link bandwidth of node i
b_i^{out}	outgoing access link bandwidth of node i
$b_{i,j}$	edge bandwidth from node i to node j
r_i^{in}	incoming rate of node i
r_i^{out}	outgoing rate of node i
c_i	number of children of node i

Table 1: Variables

$$r_i^{\text{in}} = \min_{k=1,\ldots,l} b_{i_{k-1},i_k} \tag{1}$$

where $(0 = i_0, i_1, \ldots, i_l = i)$ is a path from the root node 0 to node i. The *outgoing (sending) rate* of node i is defined as follows.

$$r_i^{\text{out}} = \min\left(r_i^{\text{in}}, \frac{1}{c_i} b_i^{\text{out}}\right) \tag{2}$$

Table 1 summarizes the variables we have defined in this section.

2.2 Fair Contribution Requirement

The centralized and distributed algorithms presented in Section 3 are "greedy" algorithms. For these algorithms, in order for the distribution tree to converge to a global optimum, rather than a local optimum, the following condition is needed.[5]

Fair Contribution Requirement *If* $b_i^{\text{in}} > b_j^{\text{in}}$*, then* $\frac{1}{c_i} b_i^{\text{out}} > \frac{1}{c_j} b_j^{\text{out}}$, for $i, j \in \{1, 2, \ldots, n\}$, $i \neq j$.

This requirement states that a node that receives more should provide more to each of its children. Suppose this requirement is not satisfied by a node that has a large incoming access link bandwidth and, relatively, a very small outgoing access link bandwidth. (This is typical of an ADSL access link.) If this node is placed high (closer to the root) in the distribution tree, selected by the greedy approach on the basis of its large incoming bandwidth without regard to its small outgoing bandwidth, then it is possible that the tree would fail to converge to the global optimum. Thus, before using one of the algorithms in Section 3 to find a distribution tree, the values of b_i^{in} and b_i^{out}, for $i = 1, 2, \ldots, n$ should be chosen such that the Fair Contribution Requirement is satisfied.

In particular, for a node with an ADSL access link, the incoming bandwidth should be reduced to a value such that the node's incoming and outgoing bandwidth values conform to the Fair Contribution Requirement. On the other hand, if a node, say i, has a very large outgoing access link bandwidth relative to its incoming access link bandwidth, it would be desirable to choose a large value for c_i so long as the Fair Contribution Requirement is not violated.

We name this requirement "Fair Contribution" because, assuming that c_i is the same, for all i, the requirement states that a node that receives more from the system should provide more to the system. We consider this to be a basic fairness principle for peer-to-peer networks.

2.3 Tree Evaluation

The incoming rate of each receiver is a good measure for evaluating a distribution tree, because it represents the amount of data that can be delivered from the root to the receiver per unit time. Given a network model

[5] See proof of Theorem 1 in Appendix A.

```
Centralized-Optimal-Tree($M$)
$T \leftarrow \varnothing$
$X \leftarrow \{0\}$
$Y \leftarrow N - \{0\}$
$r_0^{\text{in}} \leftarrow \infty$
while $Y \neq \varnothing$
    do $v \leftarrow$ a node in $X$ such that $r_v^{\text{out}} = \max_{i \in X} r_i^{\text{out}}$
       $w \leftarrow$ a node in $Y$ such that $b_w^{\text{in}} = \max_{i \in Y} b_i^{\text{in}}$
       $T \leftarrow T \cup \{(v, w)\}$
       $X \leftarrow X \cup \{w\}$
       $Y \leftarrow Y - \{w\}$
       if $|\{x|(v, x) \in T\}| = c_v$
          then $X \leftarrow X - \{v\}$
return $T$
```

Figure 2: Centralized Algorithm

$M = (N, B)$ and a tree consisting of the nodes in N, we can compute the incoming rate for every node except the root. A list of these rates is called a *rate vector*:

$$R = (r_1^{\text{in}}, r_2^{\text{in}}, \ldots, r_n^{\text{in}})\,. \tag{3}$$

Note that each tree has an associated rate vector.

We can compare distribution trees by comparing their rate vectors. However, it is difficult to determine which vector is better. The best vector for one receiver is not necessarily the best for another. We can define a partial order as follows: For rate vectors, $R_1 = (r_1^1, r_2^1, \ldots, r_n^1)$ and $R_2 = (r_1^2, r_2^2, \ldots, r_n^2)$, $R_1 \geq R_2$ if and only if $r_i^1 \geq r_i^2$ for all i, $1 \leq i \leq n$. With the partial order, although we do not know in general which rate vector is "best," it should be clear that if there is a best vector, it must be a rate vector that is not less than any other rate vector. However, for a given network model M, there are usually more than one such "locally optimum" rate vectors. Trying to find one of these is too conservative a strategy. If we stop after finding a rate vector that is not less than any other, we may overlook another that increases a large amount of rate for one receiver by sacrificing a little for another. To take the overall rate increase into account, we will evaluate a distribution tree by its average incoming rate $\frac{1}{n}\sum_{i=1}^{n} r_i^{\text{in}}$. In the next section, we present a centralized algorithm and then a distributed algorithm to find a distribution tree that maximizes the average incoming rate of receivers.

3 Optimal Algorithms

We define an optimal distribution tree to be a tree that maximizes the average incoming rate of a receiver. Given an abstract network model, $M = (N, B)$, we can find an optimal distribution tree by enumerating all trees. However, it is an infeasible approach even with a reasonable size N since there are exponentially many trees. We need more efficient algorithms to find an optimal tree.

In this section, we will first present a centralized algorithm and prove that it computes an optimal tree. Next we present a distributed version of the algorithm and prove that it converges to a tree that has the same rate vector as the optimal tree computed by the centralized algorithm. That is, the tree obtained by the distributed algorithm also maximizes the average incoming rate of a receiver.

3.1 Centralized Algorithm

Figure 2 shows the centralized algorithm to find an optimal distribution tree. X is a set of nodes that can accommodate more children, and Y a set of nodes that are not added to the tree yet. Initially, only node 0,

Variable	Description
p	parent
C	set of children
A	set of ancestors
b_x^{in}	incoming access link bandwidth of node x
b_x^{out}	outgoing access link bandwidth of node x
$b_{x,c}$	bandwidth from node x to a child c ($c \in C$)
c_x	maximum number of children
r_x^{in}	incoming rate of node x
r_x^{out}	outgoing rate of node x
$b_{a,x}$	bandwidth from an ancestor a to node x ($a \in A$)
r_a^{in}	incoming rate of an ancestor a ($a \in A$)

Table 2: State variables of node x

Message	Sender	Meaning
$\langle$*probe; i*, r_i^{in}, b_i^{in}, b_i^{out}, $c_y\rangle$	i or receiver's parent	The receiver is asked to be a new parent of node i.
$\langle$*child; i*$\rangle$	receiver's parent	The receiver is asked to accept node i as a child.
$\langle$*accept; i*$\rangle$	i	Node i has accepted the receiver as its child.
$\langle$*leave; i*$\rangle$	i	Node i is no longer a child of the receiver.

Table 3: Messages of DISTRIBUTED-OPTIMAL-TREE ($0 \leq i \leq n$)

the root node, is in X, and all other nodes are in Y. In each iteration, the algorithm selects a node that can provide the highest outgoing rate in X, and a node that has the highest incoming access link bandwidth in Y. The edge connecting them is then added to the tree T. If the node selected in X cannot accept a child any more, it is deleted from X.

This algorithm is similar to the centralized algorithm in [5] in that both algorithms are based upon the greedy method [6]. However, both our abstract model and objective function for optimization are different from the ones in [5].

A proof of the following theorem about our centralized algorithm is given in Appendix A.

Theorem 1 *With Edge Bandwidth Assumption and Fair Contribution Requirement,* CENTRALIZED-OPTIMAL-TREE(M) *yields a tree* T *that maximizes the average incoming rate* $\frac{1}{n}\sum_{i=1}^{n} r_i^{\text{in}}$.

3.2 Distributed Algorithm

In a distributed version of our algorithm, each node maintains $O(\log n)$ states about its ancestors in the tree. The distributed algorithm is specified by the actions of each node, presented in Figure 3, where node x denotes some node in N. State variables maintained by node x are shown in Table 2. Protocol messages sent and received between nodes are shown in Table 3.

Initially, we assume that the state variables, p and C, in each node have been assigned values such that the nodes in N form a random tree rooted at node 0. The variables, p and C, are updated as shown in code for node x in Figure 3. In our abstract network model, b_i^{in}, b_i^{out}, and c_i, are known constants, for all $i \in N$, and they satisfy the Fair Contribution Requirement. Also, $b_{i,j}$, for all $i, j \in N$, are known constants, and they satisfy the Edge Bandwidth Assumption. (In our protocol implementation of the distributed algorithm, presented in Section 4, we describe several protocols that provide node x with up-to-date values of its variables.)

The code for node x in Figure 3 consists of five parts. In the first part (Lines 1–4), node x chooses an ancestor randomly. Random choice does not compromise algorithm correctness as long as the root node has nonzero probability to be chosen. It only affects how fast a tree converges to an optimal distribution tree. If the chosen ancestor can be a better parent than its current one, node x sends a $\langle$*probe*$\rangle$ message to the

```
DISTRIBUTED-OPTIMAL-TREE
    ▷ Code for node $x$ $(0 \le x \le n)$.
 1  periodic probe:
 2      choose a random ancestor $a \in A$
 3      if $\min(r_a^{\text{in}}, b_{a,x}) > r_x^{\text{in}}$
 4        then send ⟨probe; $x$, $r_x^{\text{in}}$, $b_x^{\text{in}}$, $b_x^{\text{out}}$, $c_x$⟩ to $a$

 5  upon receiving ⟨probe; $y$, $r_y^{\text{in}}$, $b_y^{\text{in}}$, $b_y^{\text{out}}$, $c_y$⟩:
 6      if $y \notin C$ and $r_y^{\text{in}} < r_x^{\text{out}}$
 7        then if $|C| < c_x$ or $\min_{v \in C} r_v^{\text{out}} < \min\left(r_x^{\text{out}}, b_y^{\text{in}}, \frac{1}{c_y} b_y^{\text{out}}\right)$
 8                then $NewChild \leftarrow y$
 9                else  if $\min_{v \in C} b_{x,v} > r_y^{\text{in}}$
10                        then $m \leftarrow$ a random child
11                             send ⟨probe; $y$, $r_y^{\text{in}}$, $b_y^{\text{in}}$, $b_y^{\text{out}}$, $c_y$⟩ to node $m$
12                        else ignore the ⟨probe⟩ message
13        else ignore the ⟨probe⟩ message

14  upon receiving ⟨child, $y$⟩ or $NewChild \neq$ NIL:
15      if $NewChild \neq$ NIL
16        then $y \leftarrow NewChild$
17             $NewChild \leftarrow$ NIL
18      $C \leftarrow C \cup \{y\}$
19      if $|C| > c_x$
20        then find $l$ such that $b_{x,l} = \min_{v \in C} b_{x,v}$
21             $C \leftarrow C - \{l\}$
22             if $y \neq l$
23               then send ⟨accept; $x$⟩ to $y$
24             find $i$ such that $b_{x,i} = \max_{v \in C} b_{x,v}$
25             send ⟨child; $l$⟩ to node $i$
26        else send ⟨accept; $x$⟩ to $y$

27  upon receiving ⟨accept; $y$⟩:
28      send ⟨leave; $x$⟩ to node $p$
29      $p \leftarrow y$

30  upon receiving ⟨leave; $y$⟩:
31      $C \leftarrow C - \{y\}$
```

Figure 3: Distributed Algorithm

ancestor. The second part (Lines 5–13) describes the actions taken when a node receives a ⟨*probe*⟩ message. If the node cannot provide a higher rate than the current incoming rate of the probing node, the message is discarded. If it has room for a new child or the probing node is able to provide a higher rate to child nodes than one of the children of x, it accepts the probing node by setting *NewChild* to the probing node, which activates the third part of its code. Otherwise, the ⟨*probe*⟩ message is forwarded to a node chosen randomly among its children. The reception of a ⟨*child*⟩ message is handled in the third part (Lines 14–26). The new node is added to the children set, and the worst child (lowest edge bandwidth) is cut and forwarded to the best child. The fourth part (Lines 27–29) handles the reception of an ⟨*accept*⟩ message from a new parent, and the last part (Lines 30–31) handles the reception of a ⟨*leave*⟩ message from a child.

Theorem 2 *With Edge Bandwidth Assumption and Fair Contribution Requirement,* DISTRIBUTED-OPTIMAL-TREE *makes the distribution tree converge to a tree that has the same rate vector as the one obtained with* CENTRALIZED-OPTIMAL-TREE(M)*.*

A proof of Theorem 2 is presented in Appendix B.

4 Protocol Implementation

We have proved that Distributed-Optimal-Tree finds an optimal tree for the abstract network model. To implement the algorithm, however, several protocols are needed to initialize state variables in each node and measure up-to-date values of these variables, namely: the Join protocol, the Edge Bandwidth Measurement protocol, the Bottleneck Discovery protocol, and the Ancestor Token protocol.

4.1 Joins

The distributed algorithm is assumed to begin with a tree consisting of all participating nodes, which is unrealistic. For implementation, we provide the Join protocol which specifies how a joining node finds an existing tree node to which it attaches as a child.

For streaming media distribution, we assume that each joining node knows the root (sender) address, which can be obtained through an out-of-band channel, such as WWW. When the root receives a join request from a node, say x, the root accepts x as a child if the root has fewer children than c_0. Otherwise, the root replies to the request with the address of one of its children, say node i. Then the joining node sends a join request to i. The above procedure repeats until the joining node is accepted by some node in the tree. With this protocol, the processing overhead of a join is distributed over all nodes and the sender's load is much reduced. Note that this protocol allows a joining node to join the tree if it knows the address of any existing node in the tree. Therefore, the sender's load can be further reduced by simply announcing addresses of other tree nodes, in addition to the sender, over the out-of-band channel.

When the request of a joining node, say x, is accepted by a tree node, say y, y sends to x a range of sequence numbers indicating the part of the data stream currently available from y. Then x sends to y a chosen starting sequence number in the range, and y starts data transmission. After joining the tree, the state variables of x ($1 \leq x \leq n$) are initialized as follows: $p = y$, $c_x = 2$, $C = \varnothing$, $A = \varnothing$, $b_x^{\text{in}} = b_x^{\text{out}} = \infty$, and $r_x^{\text{in}} = r_x^{\text{out}} = 0$. The root node has the same initial values except one: $r_0^{\text{in}} = \infty$. After initialization, node x can begin executing the algorithm in Figure 3 to try to find its optimal position in the tree.

4.2 Tree information update

To run Distributed-Optimal-Tree, state variables in node x that were assumed to be up-to-date in the algorithm should be explicitly measured or calculated. We describe several more protocols and explain below how to estimate these variables.

Edge bandwidth $b_{x,c}$ The edge bandwidth from a node x to its child node c is measured with the Edge Bandwidth Measurement protocol. To avoid introducing extra traffic, this protocol measures bandwidth from actual data transmission. When the data stream is forwarded on the distribution tree from node x to node c, x transmits data packets using the congestion control mechanism of TCP.[6] In the data stream, there are marker packets, or *markers*, inserted by the root. In between two consecutive markers, 32 kB of data are transmitted. A marker has three fields: `seq_from`, `seq_to`, and `r_in`. The last field, `r_in`, is the incoming rate of the node who sends the marker; this field, updated at every node, is used by the Bottleneck Discovery protocol to be described below. `seq_from` and `seq_to` are set by the root and they do not change. They contain the sequence numbers of the data packet following this marker, and the data packet preceding the next marker.

When node c receives a marker, the time is recorded. Then node c tries to determine when it finishes receiving the 32 kB of data packets that follow this marker. The finishing time is detected either by the arrival of the next marker or a data packet whose sequence number is larger than or equal to `seq_to`. Node c calculates throughput from the amount of data received divided by the elapsed time from receiving the marker to receiving the last data packet. Node c then sends to x a protocol message containing the smaller of this throughput value and b_x^{in}. This smaller value is used as an estimate of $b_{x,c}$ at both nodes. Edge

[6] Our data transport protocol does not use other features of TCP, such as reliability.

bandwidth measurements are carried out by node c for every data interval in between consecutive markers. (Note that in node x, until it has receive $b_{x,c}$ from c for the first time, c is excluded whenever node x compares its children to select one of them in the distributed algorithm.)

Throughput is a convenient metric for available bandwidth, used in some previous studies [11, 9]. Other available bandwidth estimation methods [8, 10] can also be used instead in our protocols. A disadvantage of using throughput to estimate bandwidth is that x should have received all of the data packets between two markers before it forwards the first marker to c. Otherwise, data transmission rate may be limited by the receiving rate of x, rather than the bandwidth between x and c. It certainly increases latency. Although we can avoid this latency by using dummy data to measure $b_{x,c}$, we choose to use the actual data stream because our protocols are designed for bandwidth-intensive applications.

Outgoing access link bandwidth b_x^{out} b_x^{out} is estimated as follows.

$$b_x^{\text{out}} = \begin{cases} \frac{c_x}{|C|} \sum_{c \in C} b_{x,c} & \text{if } C \neq \varnothing \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

where C is x's set of children. When $|C| = c_x$, the above estimate is simply the total edge bandwidth and might be inaccurate if the outgoing access link is not saturated. At an intermediate node in a distribution tree, there is usually more outgoing traffic than incoming traffic because the node has more than one child. Besides, an access link with more outgoing bandwidth than incoming bandwidth is rare. Therefore outgoing links are likely to be congested and the total edge bandwidth would be a good estimate for the outgoing access link bandwidth. When $|C| < c_x$, the above formula tends to overestimate b_x^{out} and accordingly gives an advantage to x in finding its position in the tree. However, in the case that x is located higher in the tree than it should be, x has a higher probability to get a new child. Eventually C of x becomes full and the inaccuracy is corrected.

Number of children c_x and incoming access link bandwidth b_x^{in} Initially c_x is set to 2. To satisfy Fair Contribution Requirement, b_x^{in} is assigned to be $\frac{1}{c_x} b_x^{\text{out}}$. Although this is a stronger condition than that in Fair Contribution Requirement, it is simple and easy to implement. In this case, if node x is willing to support more children without reducing its current incoming rate, it can increase c_x while not violating Fair Contribution Requirement so long as the following condition is satisfied.

$$b_x^{\text{in}} = \frac{1}{c_x} b_x^{\text{out}} > r_x^{\text{in}} \tag{5}$$

The reason is as follows. When x increases c_x, it should decrease b_x^{in} to the new value of $\frac{1}{c_x} b_x^{\text{out}}$ to satisfy Fair Contribution Requirement. The reduced b_x^{in} might cause the optimal position of x to be moved to another position by the algorithm. However, r_x^{in} remains unchanged, since any node y on the path from the root to x has a higher or equal incoming bandwidth, i.e. $b_y^{\text{in}} \geq b_x^{\text{in}}$. Because $\frac{1}{c_y} b_y^{\text{out}} = b_y^{\text{in}} \geq b_x^{\text{in}}$, the new incoming rate of x is limited only by its own incoming bandwidth, b_x^{in}, which by Eq. 5 is not smaller than the previous incoming rate of x.

Incoming rate r_x^{in} The incoming rate of node x is provided by our Bottleneck Discovery protocol as follows. As mentioned in the presentation of edge bandwidth measurements, the root node sends a marker packet periodically. The last field of the marker, `r_in`, is set to "infinity" by the root. When a node, say i, receives the marker from its parent p, it compares `r_in` in the marker and $b_{p,i}$ measured by i. If $b_{p,i}$ is smaller, i overwrites `r_in` with $b_{p,i}$; otherwise, it is left unchanged. The updated marker is then forwarded by i to its child nodes. Thus, after the marker reaches node x and has been updated by x, `r_in` contains the minimum edge bandwidth on the path from the root to node x, which is r_x^{in}.

Outgoing rate r_x^{out} When node x has r_x^{in}, b_x^{out}, and c_x, r_x^{out} is obtained directly from Eq. 2.

Ancestor information A, r_a^{in}, $b_{a,x}$ State variables containing information about ancestors are used only in the first part (Lines 1–4) of DISTRIBUTED-OPTIMAL-TREE, where node x finds an ancestor, say a, to probe. The edge bandwidth $b_{a,x}$ should be known in Line 3 for x to decide if ancestor a can provide a higher rate. Since each node knows edge bandwidths from its parent to itself and from itself to its children only from the Edge Bandwidth Measurement protocol, $b_{a,x}$ needs to be measured separately. A concern is that measuring $b_{a,x}$ may overwhelm node a if many descendants ask a to perform measurement simultaneously. So, instead of letting x choose a arbitrarily, we design the Ancestor Token protocol which takes care of choosing an ancestor in Line 2 and measuring $b_{a,x}$ in Line 3 of the algorithm.

In the Ancestor Token protocol, node a sends out a token (packet) whenever a has one or more children. The token contains r_a^{in}. The token is passed to a's descendants as follows. When node a issues a token, it selects a child randomly, and passes the token to the child. When node x receives a token from a, it also passes the token to a randomly selected child if a is its parent. Otherwise, it either keeps the token with probability p, or forwards the token to a randomly selected child with probability $1 - p$. If x is a leaf node, it always keeps the token. Keeping the token means that x choose a in Line 2. While x has the token from a, it is entitled to measure $b_{a,x}$. Note that x retrieves r_a^{in} from the token, which is needed in Line 3.

The measurement procedure is similar to the one in the Edge Bandwidth Measurement protocol. Each node is expected to store in its buffer at least two consecutive marker packets and all data packets in between them. Node x sends a protocol message to a requesting measurement and data transmission to x. Then a transmits the first marker, data packets, and last marker. (Note that the markers carry r_a^{in} needed by x.) $b_{a,x}$ is estimated as in the Edge Bandwidth Measurement protocol. One difference is that the end of data transmission is detected by timeout in case the second marker is lost.

After $b_{a,x}$ has been measured or the token is lost (detected by timeout), a is ready to issue a new one. By adjusting how often tokens are issued, each node can control the amount of traffic used for bandwidth measurement from itself to descendants.

After getting r_a^{in} and $b_{a,x}$, x runs the remaining part (Lines 3–4) of the algorithm. Note that the Ancestor Token protocol removes the need for keeping information on ancestors. That is, A is no longer needed to run the algorithm, and r_a^{in} and $b_{a,x}$ are provided or measured when needed. Therefore the amount of information kept by each node is $O(c_x)$.

4.3 Node leaves and failures

In end-system multicast, we should pay more attention to node failures, because end systems are less reliable than routers in IP multicast. Therefore, it is critical to have address information about ancestor nodes. In our implementation, an important side effect when a node issues a token packet is propagating the node's address to descendant nodes. When a node has lost its parent, it is desirable for the node to contact its closest ancestor in the tree. We add a field called `distance` into the token packet to enable each node to construct a path from the root to itself. `distance` is initially set to 0 by the node issuing a token, and incremented by one by every node receiving it. Each node caches a list of ancestors containing their addresses and distances. Note that these are soft states to help recovery from node failures; with the Ancestor Token protocol, there is no longer any need for A in our algorithm implementation. If a node detects the loss of its parent by timeout, it sends a join message to nodes in its ancestor cache starting from the closest one. In the case of a voluntary leave, a leaving node sends its parent's address to all its children, so that they can send join messages to their grandparent.

4.4 Rate adaptation

In an optimal distribution tree, a node farther from the root has a lower incoming rate. Thus it may be necessary for a node to make the data stream forwarded to its child have a lower rate than the rate of the data stream it receives. A straightforward way to deal with this situation is to transcode the data stream whenever its rate should be lowered [13]. However, it may impose too much processing overhead on nodes. A better solution is to use hierarchical encoding.

Multimedia data are often encoded in layers, such that the sender provides a base layer and many enhancement layers. A receiver then subscribes to the base layer and upper layers to the extent allowed by its incoming rate. If a server makes as many layers as receivers, then every receiver can fully utilize its available bandwidth. On the other hand, with a small number of layers, a tree topology change might not lead to quality improvement if the new incoming rate of a node does not exceed the cumulative rate of the next layer. However, Yang et al. have shown that 80% of the average incoming rate can be utilized with a few (4 or 5) layers if the rates of layers are chosen carefully [17]. This indicates that available bandwidth increase is likely to improve quality for receivers when layered encoding is used.

5 Evaluation

To evaluate our protocols, we run simulations using several distributions of access link bandwidths. There are various types of access links ranging from 56 kbps telephone lines to dedicated high-speed lines with bandwidth higher than 1 Mbps. Distribution of access link bandwidths also varies. In the simulations, we use the following distributions that include both slow (< 56 kbps ≈ 0.05 Mbps) and fast (≥ 5 Mbps) links. Similar distributions have been used in previous multicast studies [12, 17].

- A uniform distribution over the interval $[0.05, 5)$.
- A normal distribution with mean 2 and standard deviation 2.
- A bimodal distribution consisting of two normal distributions. The means are 0.05 and 2.5, and the standard deviations are 0.02 and 2, respectively. In our simulations, twenty percent of the receivers are selected from the first normal distribution.

5.1 How good is the optimal tree?

The first question to investigate is whether it is worthwhile to compute an optimal tree. Randomly-constructed trees are compared with optimal trees to show that an optimal tree actually increases the average incoming rate significantly.

A random tree is a tree built with a given number of nodes, whose access link bandwidths are drawn from one of the three distributions described above. An optimal tree with the same set of nodes is computed using Centralized-Optimal-Tree(M). We plot the average incoming rates of both trees in Figure 4, with the number of nodes varied from 100 to 800. Each point in the figure represents the mean over ten simulations.

Though the actual values depend on distributions, an optimal tree has a much higher average incoming rate than a random tree. With the bimodal distribution, an optimal tree achieves a rate 30 times higher than the rate of a random tree. Note that random trees with the bimodal distribution have lower average incoming rates than those with the normal distribution, even though the mean of the bimodal distribution is larger than that of the normal distribution. The reason is that twenty percent of the nodes drawn from the bimodal distribution have very small bandwidths. It means that a small fraction of low bandwidth users can significantly slow down a large part of the tree. In this case, tree improvement is critical for the performance of bandwidth-intensive multicast applications.

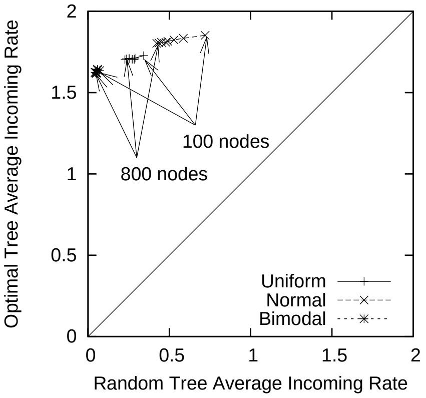

Figure 4: Optimal Trees vs. Random Trees

Another thing to notice in Figure 4 is that the average incoming rate decreases (moves toward the origin) as the number of nodes increases. Such decrease is more noticeable for random trees. The corresponding decrease for optimal trees is, however, relatively small. Therefore, a tree with more nodes gets more benefit by computing an optimal tree.

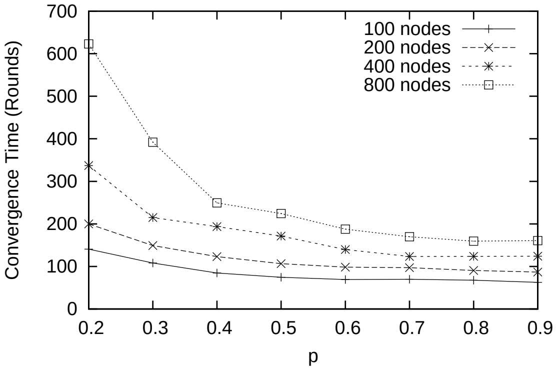

Figure 5: Convergence Time vs. p

5.2 Convergence speed

Even when the average incoming rate of an optimal tree is much higher than that of a random tree, how fast a random tree converges to an optimal tree is more important in practice. In this section, we investigate factors related to the convergence speed, especially the token keeping probability p and the number of nodes.

The convergence speed is heavily dependent on how tokens are distributed, because they give each node chances to relocate itself. Token distribution is governed by the Ancestor Token protocol with parameter p, the probability for a node to keep a token. Figure 5 shows how long it takes to achieve 80% of the maximum average incoming rate with different p values. Each point represents an average over ten runs. To measure elapsed time in the simulations, we use a *round* as a time unit. A round is the period during which each node issues a token once. We assume that every node issues tokens periodically. One round should be long enough for token propagation and edge bandwidth measurement. We also assume that edge bandwidth measurements are accurate in this section. The effect of inaccurate measurements will be discussed in Section 5.3.

As shown in Figure 5, p should be large in order for fast increase of average incoming rate. With a small p, most tokens are used by leaf nodes, and the majority of the probe messages caused by those tokens are discarded in the second part (Lines 5–13) of Distributed-Optimal-Tree. In simulations with p larger than 0.9, the speed gain in achieving 80% of the maximum becomes negligible. So we use $p = 0.9$ in later simulations.

Figure 6 demonstrates how the average incoming rate changes over time when $p = 0.9$. A tree has 500 nodes, and the average incoming rate of the tree is normalized with respect to the maximum average incoming

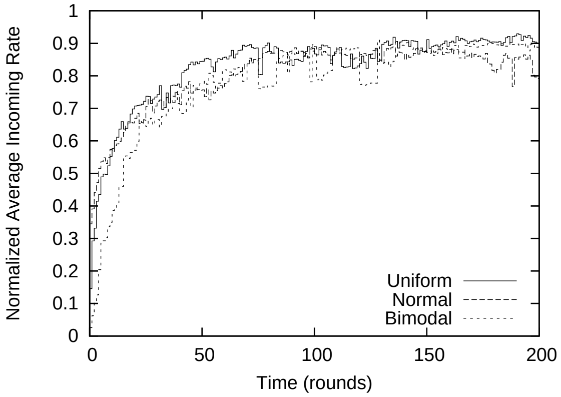

Figure 6: Evolution of Average Incoming Rate

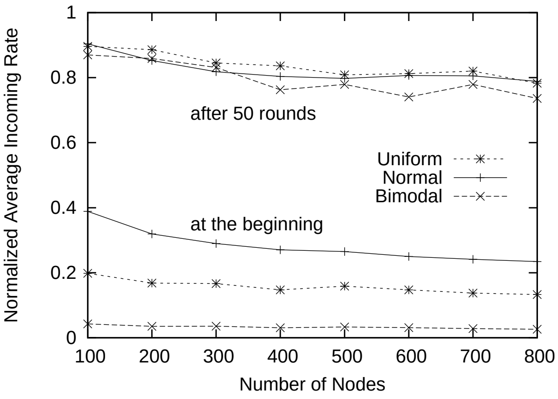

Figure 7: Average Incoming Rates at the Beginning and After 50 Rounds

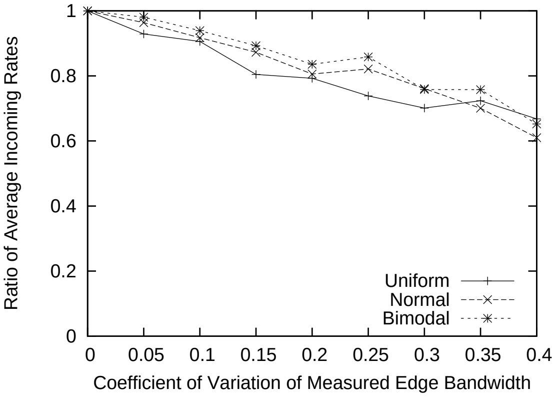

Figure 8: Measurement Errors and Achieved Average Incoming Rate

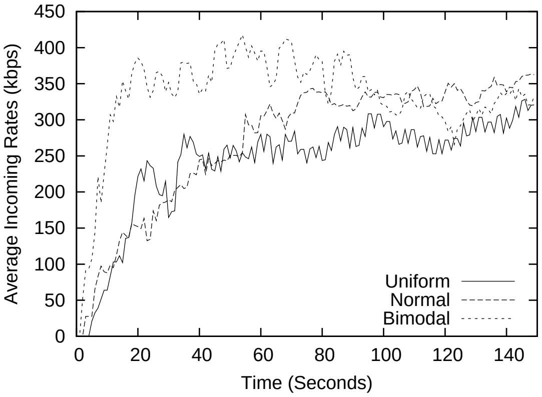

Figure 9: Tree Improvement with Measured Bandwidth

rate. The evolution of average incoming rate looks similar for all bandwidth distributions. Convergence to the maximum value takes hundreds of rounds. However, most benefits of the algorithm can be achieved within a short duration, about 50 rounds. To show that convergence time is not sensitive to the number of nodes, we plot the normalized average incoming rate both at the beginning and after 50 rounds in Figure 7. The normalized average incoming rate of each point is obtained by taking the average of 10 runs.

Again, all three trees with different bandwidth distributions show similar behaviors. Note that the average incoming rates after 50 rounds decreases as the number of nodes increases from 100 to 800. However, the decrease speed is slow. The average incoming rate for 800 nodes is 10% less than that for 100 nodes. Besides, the initial average incoming rates also decrease as the number of nodes increases; in fact, the amount of decrease is more than 30% from 100 nodes to 800 nodes. Therefore, the convergence speed is actually higher for a larger group.

These simulations show that the benefits of an optimal tree are significant and that most of them are achievable within a relatively short time.

5.3 Bandwidth measurement errors

We assumed that edge bandwidth measurements are accurate in the simulations presented in Section 5.2. In practice, however, edge bandwidth measurements may contain errors. These errors would adversely affect our protocols and lead to a sub-optimal tree.

In Figure 8, we investigate the impact of inaccurate bandwidth measurements on the average incoming rate. The tree has 500 nodes. Whenever a node measures an edge bandwidth, the value is drawn from the normal distribution with a mean value equal to the accurate edge bandwidth. We change the coefficient of variation (CoV) of the normal distribution to vary the degree of errors. The ratio of the average incoming rates (after 50 rounds) for trees with inaccurate and accurate measurement is plotted in Figure 8.

The ratio of the average incoming rates decreases linearly as CoV increases. In order to achieve a ratio higher than 0.8, CoV should not exceed 0.3. Some congestion control protocols designed to avoid sending rate fluctuations have sending rate CoV lower than 0.3 [18]; therefore, the throughput of one of these protocols would be suitable for edge bandwidth estimation in our algorithm implementation. Protocols with larger CoV like the AIMD (additive increase/multiplicative decrease) protocol of TCP can also be used by having sufficiently large measurement timescale to decrease CoV [7].

Figure 9 shows the average incoming rate traces using AIMD throughput to estimate edge bandwidths. The simulations are run using the ns–2.1b9 simulator,[7] for a topology generated with the Transit-Stub model of Georgia Tech Internetwork Topology Models (GT-ITM) [2]. The topology contains 100 routers: 75 stub routers and 25 transit routers. 51 nodes are added to the topology. One of them is the sender, and the other

[7] http://www.isi.edu/nsnam/ns/

nodes are receivers. Access link bandwidths are drawn from the uniform, normal, and bimodal distributions described at the beginning of this section. Due to large variation in throughput measurements, the average incoming rate curves show large fluctuations. One thing to notice is that the average incoming rate is much lower than the average of the bandwidth distribution. The first reason is, as we have mentioned before, that measurement errors result in a low average incoming rate. The second is that throughput measurements with 32 kB blocks give a significantly lower value than the actual edge bandwidth, especially for those with high bandwidth; a 32 kB block may fail to saturate such a high bandwidth edge. Due to low link utilization, the measured edge bandwidth becomes much lower than the actual value, and the average incoming rate is also lower than it should be. However, the algorithm is still effective because all it needs is relative comparison among edge bandwidths.

Even with the inaccurate bandwidth measurements, the curves in Figure 9 look similar to those in Figure 6. In Figure 9, the average incoming rate increases for about eighty seconds and stays at a relatively stable level. Since the usual playback time of audio and video streams exceeds minutes and even hours, we believe this is acceptable.

6 Conclusion

Finding a good tree topology is critical for the performance of bandwidth-intensive multicast applications. We have proposed a distributed algorithm to build a tree in the application layer, and proved that it finds an optimal tree, which maximizes the average incoming rate of receivers under certain network model assumptions. Unlike other approaches using heuristics to find a local optimum, our algorithm is always heading towards the global optimum. We have described protocols to implement the algorithm on the Internet. Since a node does not keep any hard state in our implementation, it is resilient to membership changes and failures. Any node can take care of join requests in the same way as the root does, and can easily recover from leaves or failures of other nodes.

Our protocol implementation has room for improvement, especially in bandwidth measurement. The AIMD throughput has large variations, caused in part by short-term unfairness of the protocol and in part by interference from other flows. The former is avoidable by adopting a more fair and smoother protocol such as TFRC [7] and TEAR [15]. Because a basic assumption of our algorithm is that a node can measure the bandwidth between another node and itself, we expect that a more accurate and stable estimation technique will lead to better algorithm performance. This is a topic of our future study.

References

[1] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM 2002*, August 2002.

[2] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[3] Yatin Chawathe and Mukund Seshadri. Broadcast Federation: an application layer broadcast internetwork. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, May 2002.

[4] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[5] Reuven Cohen and Gideon Kaempfer. A unicast-based approach for streaming multicast. In *Proceedings of IEEE INFOCOM 2001*, April 2001.

[6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, chapter 17. MIT Press, 1990.

[7] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

[8] Mukul Goyal, Roch Guerin, and Raju Rajan. Predicting TCP throughput from non-invasive network sampling. In *Proceedings of IEEE INFOCOM 2002*, June 2002.

[9] Katrina M. Hanna, Nandini Natarajan, and Brian Neil Levine. Evaluation of a novel two-step server selection metric. In *Proceedings of the 9th IEEE International Conference on Network Protocols*, November 2001.

[10] Manish Jain and Constantionos Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. In *Proceedings of ACM SIGCOMM 2002*, August 2002.

[11] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and Jr. James W. O'Toole. Overcast: reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*. USENIX, October 2000.

[12] Tianji Jiang, Mostafa H. Ammar, and Ellen W. Zegura. Inter-receiver fairness: a novel performance measure for multicast ABR sessions. In *Proceedings of ACM SIGMETRICS '98*, June 1998.

[13] Zhijun Lei. Media transcoding for pervasive computing. In *Proceedings of the 9th ACM International Conference on Multimedia*, September 2001.

[14] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: an application level multicast infrastructure. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[15] Injong Rhee, Volkan Ozdemir, and Yung Yi. TEAR: TCP emulation at receivers — flow control for multimedia streaming. Technical report, Department of Computer Science, North Carolina State University, April 2000.

[16] Maya Yajnik, Jim Kurose, and Don Towsley. Packet loss correlation in the MBone multicast network. In *Proceedings of IEEE Global Internet 1996*, November 1996.

[17] Y. Richard Yang, Min Sik Kim, and Simon S. Lam. Optimal partitioning of multicast receivers. In *Proceedings of the 8th IEEE International Conference on Network Protocols*, Osaka, Japan, November 2000.

[18] Y. Richard Yang, Min Sik Kim, and Simon S. Lam. Transient behaviors of TCP-friendly congestion control protocols. In *Proceedings of IEEE INFOCOM 2001*, April 2001.

A Proof of Theorem 1

Proof Let T be the tree built with Centralized-Optimal-Tree(M) and $R = (r_1^{\text{in}}, r_2^{\text{in}}, \ldots, r_n^{\text{in}})$ its rate vector. Suppose that T^* is a tree that maximizes the average incoming rate and that its rate vector is $R^* = (r_1^{\text{in}^*}, r_2^{\text{in}^*}, \ldots, r_n^{\text{in}^*})$. Without loss of generality, we assume that $(1, 2, \ldots, n)$ is the order in which receiver nodes are added to the tree T by Line 8 of the algorithm. We will show by induction that T^* can be transformed into T without reducing any incoming rate, which proves that T also maximizes the average incoming rate.

The transformation procedure consists of n steps, where n is the number of receivers. At ith step, node i is relocated in the tree so that nodes $0, 1, \ldots, i$ have the same topology as in T. For example, we start with T^* whose root is node 0. Though node 1 is a child of node 0 in T by the algorithm, it may not be the case in T^*. Then, at the first step, we make node 1 a child of node 0, and adjust other child nodes of node 0 not to exceed the number of children c_0. We will explain how to adjust other nodes later. After this step, we have a tree that has node 0 as its root and node 1 as 0's child. At the second step, node 2 is relocated so that a subtree containing node 0, 1, and 2 have the same topology as in T. This transformation is continued until node n is relocated. At each step, we make sure that the average incoming rate of the transformed tree is equal to that of T^*.

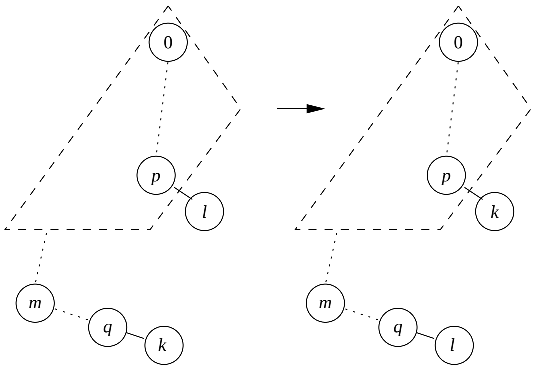

Figure 10: Converted Trees

We use induction on the number of steps in transforming T^* into T. Let T_i denote the transformed tree after i steps. Then we are to prove that T_i has the following properties for all i, where $0 \leq i \leq n$.

1. Nodes $0, 1, \ldots, i$ in T_i have the same topology as in T.
2. The average incoming rate of T_i is equal to that of T^*.

Note that T_i maximizes the average incoming rate if the second property holds.

The base case is trivial, because after zeroth step, the transformed tree T_0 is T^* itself, and node 0 is located at the root of the tree both in T_0 and T^*.

Suppose as induction hypothesis that both properties hold when $i = k - 1$.

Let the rate vector of T_{k-1} be $R' = ({r_1^{\text{in}}}', {r_2^{\text{in}}}', \ldots, {r_n^{\text{in}}}')$. By the induction hypothesis, ${r_i^{\text{in}}}' = r_i^{\text{in}}$ for all i, $1 \leq i \leq k-1$. The comparison of r_k^{in} and ${r_k^{\text{in}}}'$ gives two cases: (i) $r_k^{\text{in}} < {r_k^{\text{in}}}'$ and (ii) $r_k^{\text{in}} \geq {r_k^{\text{in}}}'$. We first show that (i) leads to a contradiction.

Assume that (*i*) is true. Let node j be the first node on the path from the root to node k in T_{k-1} that is not in $\{0, 1, 2, \ldots, k-1\}$. If $j = k$, k's parent in T_{k-1} must be in $\{0, \ldots, k-1\}$ and have an outgoing rate than k's parent in T, because $r_k^{\text{in}} < {r_k^{\text{in}}}'$. This is impossible because of Line 6. If $j > k$, then ${r_j^{\text{in}}}' \geq {r_k^{\text{in}}}'$ because k is j's descendant. From this and (i), we conclude ${r_j^{\text{in}}}' > r_k^{\text{in}}$, which means j should have been chosen instead of k in building T. It contradicts the assumption that T is obtained by the algorithm.

Now only (ii) can hold.

$$r_k^{\text{in}} \geq {r_k^{\text{in}}}' \tag{6}$$

Consider k's position in T. If the position is empty in T_{k-1}, then moving k's subtree to that position does not decreases any incoming rate of nodes in k's subtree, because k's position in T is chosen in Line 6 to maximize k's incoming rate when Edge Bandwidth Assumption holds.

If the position is not empty in T_{k-1}, let l be the node occupying the position. The left tree in Figure 10 represents T_{k-1}, and the right represents T_k. The area surrounded with a dotted line is the common part of T and T_{k-1}, and contains nodes $1, 2, \ldots, k-1$.

We know

$$r_k^{\text{in}} \geq {r_l^{\text{in}}}' \tag{7}$$

because k has been chosen in Line 7 of the algorithm. There are two possibilities depending on whether k is l's descendant or not. If it is, we exchange k and l, and if it is not, we exchange k's subtree and l's subtree. We will show that neither case decreases any incoming rate.

As for node k, it will have the incoming rate r_k^{in} after exchange since the new position is the position of node k in T. By Eq. 6, k's incoming rate does not decrease and neither of its descendants if we exchange the subtrees.

To calculate l's new incoming rate, suppose that p and q are parents of l and k in T_{k-1}, respectively. Then $r_p^{\text{out}} = r_p^{\text{out}\prime} \geq r_q^{\text{out}\prime}$ by the algorithm and Fair Contribution Requirement. Because the new incoming rate of l is to be the minimum of $r_q^{\text{out}\prime}$ and b_l^{in} by Edge Bandwidth Assumption, there are two cases depending on which value is the smaller. If $r_q^{\text{out}\prime} \geq b_l^{\text{in}}$, l's new incoming rate after exchange will be b_l^{in}, which is equal to $b_l^{\text{in}\prime}$. If $r_q^{\text{out}\prime} < b_l^{\text{in}}$, l's new incoming rate will be $r_q^{\text{out}\prime}$, which is equal to $r_k^{\text{in}\prime}$, since $b_l^{\text{in}} \leq b_k^{\text{in}}$ by Line 7. Therefore, l's new incoming rate is at least $\min(b_l^{\text{in}\prime}, b_k^{\text{in}\prime})$.

Then the lower bound of the net effect for k's and l's incoming rates is as follows.

$$(k\text{'s rate change}) - (l\text{'s rate change}) \geq (b_k^{\text{in}} - b_k^{\text{in}\prime}) + \Big(\min(b_l^{\text{in}\prime}, b_k^{\text{in}\prime}) - b_l^{\text{in}\prime}\Big) \geq 0 \tag{8}$$

In the case where k is l's descendant, any incoming rate of the nodes on the path from l to k does not decrease after the exchange of k and l, because k's incoming rate does not decrease. Incoming rates of k's descendants also do not decrease because the nodes on the path from the root remain same and thus they have the same bottleneck. By the arguments above and Eq. 8, the average incoming rate does not decrease after the exchange of k and l.

When the subtrees of k and l are exchanged, it is clear that incoming rates of k's descendants do not decrease because k's incoming rate does not. Consider l's descendants. The incoming rate of l decreases if the new value is $r_k^{\text{in}\prime}$, which happens to be smaller than the previous value $r_l^{\text{in}\prime}$. Before deriving the incoming rate changes of l's descendants, we claim $r_q^{\text{out}\prime} \geq \min\left(b_k^{\text{in}}, \frac{1}{c_k} b_k^{\text{out}}\right)$. If not, there exists a bottleneck node m on the path from 0 to k such that $r_m^{\text{out}\prime}$ is equal to $r_q^{\text{out}\prime}$. It means we can achieve a higher average incoming rate by exchanging node m and node k, which contradicts that T_{k-1} maximizes the average incoming rate.

We know $b_l^{\text{in}} \leq b_k^{\text{in}}$ by Line 7, and accordingly $\frac{1}{c_l} b_l^{\text{out}} \leq \frac{1}{c_k} b_k^{\text{out}}$ by Fair Contribution Requirement. By this and the previous claim, we get $r_q^{\text{out}\prime} \geq \min\left(b_l^{\text{in}}, \frac{1}{c_l} b_l^{\text{out}}\right)$. It means that q provides a higher rate than l can forward to its children. Hence the incoming rates of l's descendants do not decrease.

Therefore, by exchange, nodes $0, 1, \ldots, k$ have the same topology as in T, and the average incoming rate of T_{k-1} is not less than that of T^*. It means the average incoming rate should be equal to that of T^* because T^* is an optimal tree. Thus two properties hold when $i = k$.

By induction, T_n maximizes the average incoming rate. Since $T_n = T$ by the first property, T is also a tree that maximizes the average bandwidth. □

B Proof of Theorem 2

Proof We will show that Distributed-Optimal-Tree makes a tree converge to an optimal tree by proving (i) that nodes that are already in their optimal positions do not move any more and (ii) that nodes not in optimal positions move to their optimal positions by the algorithm. We mean by an optimal position the position where a node is located by Centralized-Optimal-Tree(M).

Without loss of generality, we assume that $(1, 2, \ldots, n)$ is the order in which receiver nodes are added to the tree T by Centralized-Optimal-Tree(M). Then Distributed-Optimal-Tree locates nodes in their optimal positions in this order.

We use induction on nodes. The base case is trivial. Node 0 is always in its optimal position and never moves.

Suppose as induction hypothesis that nodes $1, 2, \ldots, k-1$ are in their optimal positions, and will not move any more. For any node $i < k$, every $\langle probe \rangle$ message from i is discarded either in Line 6 or in Line 7 of Distributed-Optimal-Tree because of the following reasons. Assume i sends a $\langle probe \rangle$ message and let p denote i's parent. Note that $r_i^{\text{in}} = r_p^{\text{out}}$ because i sends a $\langle probe \rangle$ message only when b_i^{in} is not the bottleneck. Suppose node j receives the $\langle probe \rangle$ message. If the message is not discarded in Line 6, $r_p^{\text{out}} = r_i^{\text{in}} < r_j^{\text{out}}$, which means j must be chosen earlier than p in running Centralized-Optimal-Tree(M). Therefore j doesn't have any available slot for a child and all its children have better outgoing rate than i could have if i were j's child. Hence the $\langle probe \rangle$ message is discarded in Line 7, and node i stays at its optimal position.

If k is already in its optimal position, then it also does not move by the same argument. If k is not in an optimal position and the incoming rate is less than that in the optimal position, there exist one or more ancestors (including 0) that is an ancestor of k in the optimal tree and satisfies the conditions in Line 3. Since k executes the probe routine periodically, it chooses 0 as a with probability 1. (Choosing a non-root ancestor can accelerate the convergence, without compromising this proof.) Then k sends a $\langle probe \rangle$ message to a. Since k's optimal position is in a's subtree, the $\langle probe \rangle$ message eventually reaches the node that is k's parent in the optimal tree. Because of the way k is chosen in Line 7 of Centralized-Optimal-Tree(M), k is accepted as a child in the second part of Distributed-Optimal-Tree. One exception is the case where k is not accepted in Line 9 due to other nodes tying with k. However, exchanging nodes of the tie does not change any incoming rate.

By repeating the procedure above, an optimal tree is obtained by induction. □