In *Proceedings of the 23rd IEEE ICDCS*, May 2003. .

Optimal Distribution Tree for Internet Streaming Media[*]

Min Sik Kim, Simon S. Lam, and Dong-Young Lee
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712–1188, USA
{*minskim,lam,dylee*}*@cs.utexas.edu*

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.

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] IP multicast would be a highly efficient delivery model for these applications. However, in the absence of widespread deployment of IP multicast, end-system multicast has emerged as an attractive alternative.

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 optimize a tree, they 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 [10], 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 proved to lead to an optimal state, the proposal in [10] 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

[*]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/).

[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.

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 a 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 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, the distribution tree is continuously updated as it converges toward an optimal tree. When there is a node failure, our protocols will adapt and the distribution tree will start converging toward 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.

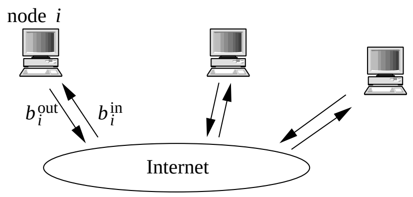

Figure 1. Network model

2. Network Model

Our goal in building a streaming media distribution tree is to find a tree that provides the largest available bandwidth. Available bandwidth is determined by many factors. In particular, the available bandwidth between two nodes is a function of the underlying Internet topology and existing traffic. To simplify our model, we use the following observations to 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 [15].
- An access link has incoming and outgoing bandwidths that do not affect each other.

An access link is a link that connects a host or its local area network to its ISP. 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 each node's access link. The other links in between add delay, but do not limit the bandwidth of the flow.

2.1. Abstract 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 $b_{i,j}$,

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

Table 1. Variables

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

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 *incoming (receiving) rate* of node i is the minimum of edge bandwidths on the path from the root node to node i.

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

where $(0 = i_0, i_1, \ldots, i_l = i)$ is a path from the root 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.

2.2. Fair Contribution Requirement

The centralized and distributed algorithms to be 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 (close 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. On the other hand, if a node, say i, has a very large b_i^{out} relative to b_i^{in}, 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 $M = (N, B)$ and a tree consisting of the nodes in N, we can compute r_i^{in} for every receiver node i. A list of these rates is called a *rate vector*: $R = (r_1^{\text{in}}, r_2^{\text{in}}, \ldots, r_n^{\text{in}})$. 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 a chance to increase a large amount of rate for one receiver by sacrificing a little for another. To take the overall rate increase into account, we evaluate a distribution tree by its average incoming rate $\frac{1}{n}\sum_{i=1}^{n} r_i^{\text{in}}$.

3. Optimal Algorithms

We define an optimal distribution tree to be a tree that maximizes the average incoming rate of a receiver. In this section, we first present an efficient centralized algorithm and prove that it computes an optimal distribution tree. Next we present a distributed version of the algorithm and prove

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

```
CENTRALIZED-OPTIMAL-TREE
 1 $T \leftarrow \varnothing$, $X \leftarrow \{0\}$, $Y \leftarrow N - \{0\}$, $r_0^{\mathrm{in}} \leftarrow \infty$
 2 while $Y \neq \varnothing$
 3     do $v \leftarrow$ a node in $X$ such that $r_v^{\mathrm{out}} = \max_{i \in X} r_i^{\mathrm{out}}$
 4        $w \leftarrow$ a node in $Y$ such that $b_w^{\mathrm{in}} = \max_{i \in Y} b_i^{\mathrm{in}}$
 5        $T \leftarrow T \cup \{(v, w)\}$
 6        $X \leftarrow X \cup \{w\}$
 7        $Y \leftarrow Y - \{w\}$
 8        if $|\{x | (v, x) \in T\}| = c_v$
 9           $X \leftarrow X - \{v\}$
10 return $T$
```

Figure 2. Centralized algorithm

that it converges to a tree that has the same rate vector as the optimal tree computed by the centralized algorithm.

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 the root node is in X, and the others are in Y. In each iteration, the algorithm selects a node with the highest outgoing rate in X, and a node with the highest incoming access link bandwidth in Y. The edge connecting them is then added to the tree T. Nodes that cannot accept a child any more are deleted from X.

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

The following theorem is proved in Appendix A.

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

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 an arbitrary tree rooted at node 0. The variables, p and C, are updated as shown 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

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

 5  upon receiving ⟨probe; $y$, $r_y^{\mathrm{in}}$, $b_y^{\mathrm{in}}$, $b_y^{\mathrm{out}}$, $c_y$⟩:
 6    if $y \notin C$ and $r_y^{\mathrm{in}} < r_x^{\mathrm{out}}$
 7       if $|C| < c_x$ or $\min_{v \in C} r_v^{\mathrm{out}} < \min\left(r_x^{\mathrm{out}}, b_y^{\mathrm{in}}, \frac{1}{c_y} b_y^{\mathrm{out}}\right)$
 8          $NewChild \leftarrow y$
 9       else if $\min_{v \in C} b_{x,v} > r_y^{\mathrm{in}}$
10          $m \leftarrow$ a random child
11          send ⟨probe; $y$, $r_y^{\mathrm{in}}$, $b_y^{\mathrm{in}}$, $b_y^{\mathrm{out}}$, $c_y$⟩ 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       $y \leftarrow NewChild$
17       $NewChild \leftarrow$ NIL
18    $C \leftarrow C \cup \{y\}$
19    if $|C| > c_x$
20       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          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

Section 4, we describe several protocols that provide node x with up-to-date values of its variables.)

In 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 ⟨*probe*⟩ message to the ancestor. Lines 5–13 describe 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 next part of the algorithm. Otherwise,

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

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 2. State variables of node x

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 ($a \in A$)

the $\langle$*probe*$\rangle$ message is forwarded to a node chosen randomly among its children. When Lines 14–26 are triggered by $NewChild$ or a $\langle$*child*$\rangle$ message, the new node is added to the children set and the worst (lowest edge bandwidth) child is cut and forwarded to the best child with a $\langle$*child*$\rangle$ message. Lines 27–29 handle the reception of an $\langle$*accept*$\rangle$ message from a new parent, and Lines 30–31 handle the reception of a $\langle$*leave*$\rangle$ message from a child.

The following theorem is proved in Appendix B.

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.

4. Protocol Implementation

To implement DISTRIBUTED-OPTIMAL-TREE, 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

Each joining node knows the root (sender) address through an out-of-band channel, such as WWW. When the root receives a join request from node x, $x \neq 0$, the root accepts x as a child if the root has fewer children than c_0. Otherwise, the root returns the address of one of its children, say node i. Then x sends a join request to i. This procedure repeats until x is accepted by some tree node. With this protocol, the processing overhead of joins is distributed over all nodes. Because every node in the tree is capable of handling join requests, the sender's load is further reduced by announcing addresses of other tree nodes, in addition to the sender, over the out-of-band channel.

When the join request of x is accepted by node y, y sends to x a range of sequence numbers indicating the part of the data stream available from y. Node x notifies y of a chosen starting sequence number, and y starts data transmission. After joining the tree, the state variables of x 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 $r_0^{\text{in}} = \infty$.

4.2. Tree information update

Edge bandwidth $b_{x,c}$ The Edge Bandwidth Measurement protocol measures $b_{x,c}$ from actual data transmission to avoid introducing extra traffic. Node x forwards data packets to c 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 do not change. They contain the sequence numbers of the data packet following this marker, and the data packet preceding the next marker.

A marker initiates throughput measurement. Node c records the time when it receives a marker. When the data packet whose sequence number is `seq_to` arrives, c calculates throughput from the amount of received data and the elapsed time since it received the marker. In a case when the data packet `seq_to` is lost in transmission, c calculates throughput when the next marker or a data packet whose sequence number is larger than `seq_to` arrives. The smaller of the measured throughput and b_x^{in} is an estimate of $b_{x,c}$, and sent to x. (Note that in x, until it receives $b_{x,c}$ from c for the first time, c is excluded whenever 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 [10, 8]. Other avail-

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

able bandwidth estimation methods [7, 9] can also be used instead. A disadvantage of using throughput to estimate bandwidth is that x must receive all of the data packets in 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 let x wait 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 $\frac{c_x}{|C|}\sum_{c\in C} b_{x,c}$ if $C \neq \varnothing$, and ∞ otherwise. When $|C| = c_x$, this is simply the total edge bandwidth and might be inaccurate if the outgoing access link is not saturated. However, an intermediate node in a distribution tree usually has 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 the 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 the 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 the Fair Contribution Requirement so long as the following condition is satisfied: $b_x^{\text{in}} = \frac{1}{c_x}b_x^{\text{out}} > r_x^{\text{in}}$. 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 the Fair Contribution Requirement. The reduced b_x^{in} might cause x to be moved to a new optimal position by the algorithm. However, r_x^{in} remains unchanged, because otherwise there must be a node y on the path from the root to x that has a smaller incoming bandwidth and it means x is not at an optimal position.

Incoming rate r_x^{in} The Bottleneck Discovery protocol provides r_x^{in}. The root sets `r_in` in each marker to "infinity." When a node i receives a marker from its parent p, it compares `r_in` in the marker and $b_{p,i}$. If $b_{p,i}$ is smaller, i overwrites `r_in` with $b_{p,i}$. The marker is then forwarded to i's children. Thus, when 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.

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}, $b_{a,x}$ Since Edge Bandwidth Measurement protocol is run only between a parent and a child, $b_{a,x}$ needs to be measured separately. A concern is that measuring $b_{a,x}$ may overwhelm a if many descendants of a try to measure simultaneously. So, instead of letting x choose a arbitrarily, we design the Ancestor Token protocol which takes care of Lines 2–3 in the algorithm.

In the Ancestor Token protocol, node a issues a token (packet) containing r_a^{in}, whenever a has one or more children. The token is passed to a randomly chosen child. When node x receives a token from a, it passes the token to a random child if a is its parent. Otherwise, it either keeps the token with probability p, or forwards the token to a random child with probability $1 - p$. If x is a leaf node, it always keeps the token. Keeping the token means that x chooses 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 Edge Bandwidth Measurement protocol. Each node stores 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. Then a transmits two markers with data between them. $b_{a,x}$ 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 last data packet and the second marker are 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. The Ancestor Token protocol removes the need for keeping information on ancestors. A is no longer needed to run the algorithm. Therefore the amount of information kept by node x 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 of the Ancestor Token Protocol is propagating a node's address to descendants. 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. These are soft states to help recovery from node failures. 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 may need to make the data stream forwarded to its child have a lower rate than the rate of the data stream it receives, if its outgoing rate is smaller than the incoming rate. A straightforward way to deal with this situation is to transcode the data stream [13]. However, it may impose too much processing overhead on nodes. A better solution is to use hierarchical (or layered) encoding. A concern with this approach is that a tree topology change may not lead to quality improvement if the new incoming rate of a node is less than the cumulative rate of the next layer. However, Yang et al. have shown that 80% of the average incoming rate of an optimal tree can be utilized with a few (4 or 5) layers if the rates of layers are chosen carefully [16]. This indicates that available bandwidth increase is likely to improve quality for receivers when hierarchical encoding is used.

5. Evaluation

We use the following access link bandwidth distributions that include both slow (.05 Mbps) and fast (5 Mbps) links: a uniform distribution over the interval $[.05, 5)$, a normal distribution with mean 2 and standard deviation 2, and a bimodal distribution consisting of two normal distributions, of which the means are .05 and 2.5, and the standard deviations are .02 and 2, respectively. In the bimodal distribution, 20% of the receivers are selected from the first normal distribution. Similar distributions have been used in previous multicast studies [11, 16].

5.1. How good is the optimal tree?

To show that an optimal tree increases the average incoming rate significantly, optimal trees are compared with random trees. A random tree is 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. We plot the average incoming rates of both trees in Figure 4, with the number of nodes varied from 100 to 800. Each point represents the mean over ten simulations.

For all distributions, an optimal tree has a much higher average incoming rate than 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 indicates that a small fraction of low bandwidth users can significantly slow down a large part of the tree. In such a case, tree improvement is critical.

Another thing to notice is that the rate decreases (moves toward the origin) as the number of nodes increases. It is more noticeable for random trees. The decrease for optimal trees is, however, relatively small. Therefore, a tree with more nodes gets more benefit from our algorithm.

5.2. Convergence speed

In practice, how fast a random tree converges to an optimal tree is also important. The convergence speed is heavily dependent on how tokens are distributed, because they trigger relocation of nodes. Figure 5 shows how long it takes to achieve 80% of the maximum average incoming rate with different p, the probability to keep a token. Each point represents an average over ten runs. Elapsed time is measured in *rounds*. 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. The effect of inaccurate measurements will be discussed in Section 5.3.

As shown in Figure 5, p should be large for fast convergence. With a small p, most tokens are wasted by leaf nodes. In simulations with p larger than 0.9, the speed gain becomes negligible, so we use $p = 0.9$ in later simulations.

Figure 6 demonstrates how the average incoming rate changes over time. A tree has 500 nodes, and the average incoming rate is normalized with respect to the maximum average incoming rate. Though convergence to the maximum takes hundreds of rounds, most rate increase happens 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 rates both at the beginning and after 50 rounds in Figure 7. Each point is obtained by taking the average of 10 runs. Though the average incoming rates after 50 rounds decrease as the number of nodes increases from 100 to 800, the decrease speed is slow. Besides, the initial average incoming rates decrease more as the number of nodes increases. Therefore, the convergence speed is actually higher for a larger group.

5.3. Bandwidth measurement errors

In Figure 8, we investigate the impact of inaccurate bandwidth measurements on the average incoming rate.

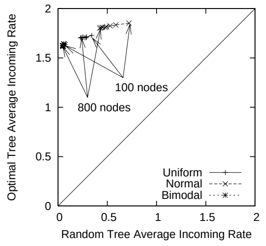

Figure 4. Optimal trees vs. random trees

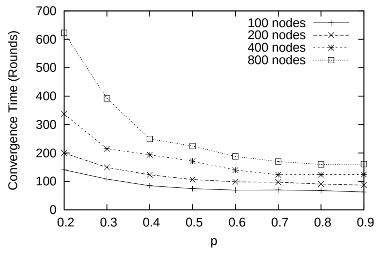

Figure 5. Convergence time vs. p

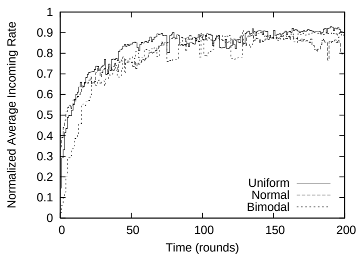

Figure 6. Evolution of average incoming rate

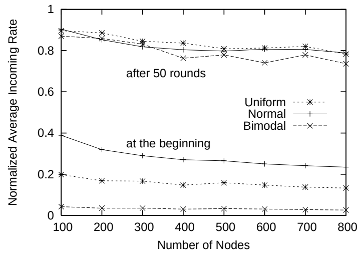

Figure 7. Average incoming rates in 50 rounds

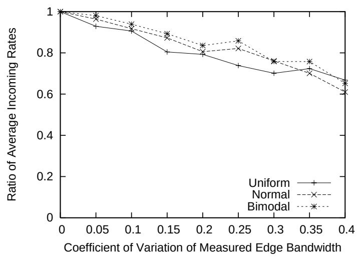

Figure 8. Effect of measurement errors

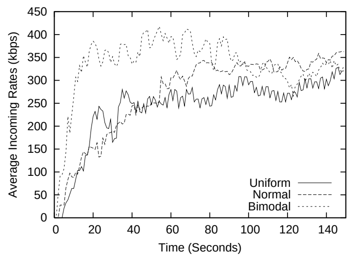

Figure 9. Evolution with measured bandwidth

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.

The ratio 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 [17]; therefore, the throughput of one of these protocols would be suitable for edge bandwidth estimation in our algorithm implementation. Protocols with larger CoV such as the AIMD (additive increase/multiplicative decrease) protocol of TCP can also be used by having a sufficiently large measurement timescale to decrease CoV [6].

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] on a topology generated with the Transit-Stub model of Georgia Tech Internetwork Topology Models (GT-ITM) [2]. The topology contains 75 stub and 25 transit routers. one sender node and 50 receiver nodes are added to the topology. Access link bandwidths are drawn from the three 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 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 that this is acceptable for such applications.

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 toward 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.

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

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 [17] by adopting a more fair and smoother protocol such as TFRC [6]. Because a basic assumption of our algorithm is that a node can measure the bandwidth from another node to itself, we expect that a more accurate and stable estimation technique will lead to better performance. This is a topic of our future study.

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM 2002*, Aug. 2002.

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

[3] Y. Chawathe and M. Seshadri. Broadcast Federation: an application layer broadcast internetwork. In *Proceedings of NOSSDAV 2002*, May 2002.

[4] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM 2001*, Aug. 2001.

[5] R. Cohen and G. Kaempfer. A unicast-based approach for streaming multicast. In *Proceedings of IEEE INFOCOM 2001*, Apr. 2001.

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM 2000*, Aug. 2000.

[7] M. Goyal, R. Guerin, and R. Rajan. Predicting TCP throughput from non-invasive network sampling. In *Proceedings of IEEE INFOCOM 2002*, June 2002.

[8] K. M. Hanna, N. Natarajan, and B. N. Levine. Evaluation of a novel two-step server selection metric. In *Proceedings of the 9th IEEE ICNP*, Nov. 2001.

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

[10] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. James W. O'Toole. Overcast: reliable multicasting with an overlay network. In *Proceedings of OSDI 2000*. USENIX, Oct. 2000.

[11] T. Jiang, M. H. Ammar, and E. W. Zegura. Inter-receiver fairness: a novel performance measure for multicast ABR sessions. In *Proceedings of ACM SIGMETRICS '98*.

[12] M. S. Kim, S. S. Lam, and D.-Y. Lee. Optimal distribution tree for Internet streaming media. Technical Report TR–02–48, Department of Computer Sciences, University of Texas at Austin, Sept. 2002; available at http://www.cs.utexas.edu/users/lam/NRL/.

[13] Z. Lei. Media transcoding for pervasive computing. In *Proceedings of the 9th ACM Multimedia*, Sept. 2001.

[14] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: an application level multicast infrastructure. In *Proceedings of USITS '01*. USENIX, Mar. 2001.

[15] M. Yajnik, J. Kurose, and D. Towsley. Packet loss correlation in the MBone multicast network. In *Proceedings of IEEE Global Internet 1996*, Nov. 1996.

[16] Y. R. Yang, M. S. Kim, and S. S. Lam. Optimal partitioning of multicast receivers. In *Proceedings of the 8th IEEE ICNP*, Nov. 2000.

[17] Y. R. Yang, M. S. Kim, and S. S. Lam. Transient behaviors of TCP-friendly congestion control protocols. *Computer Networks*, 41(2):193–210, Feb. 2003; an abbreviated version in *Proceedings of IEEE INFOCOM 2001*, Apr. 2001.

A. Proof of Theorem 1

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

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. The subgraph consisting of nodes $0, 1, \dots, i$ and edges between them in T_i is a tree and equal to the corresponding subgraph in T.
2. The average incoming rate of T_i is equal to that of T^*.

The base case is trivial, because after zeroth step, the transformed tree T_0 is T^* itself. Suppose as induction hypothesis that both properties hold when $i = k-1$. It suffices to show that we can construct T_k with both properties.

Let the rate vector of T_{k-1} be $R' = ({r_1^{\text{in}}}', {r_2^{\text{in}}}', \dots, {r_n^{\text{in}}}')$. 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.

Assuming (i), let node j be the first node on the path from the root to node k in T_{k-1} that is not in $\{0, 1, 2, \dots, k-1\}$. If $j = k$, k's parent in T_{k-1} must be in $\{0, \dots, k-1\}$, and have an outgoing rate larger than k's parent in T to satisfy (i). This is impossible because of Line 3. 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 (ii) should hold. If k's position in T is empty in T_{k-1}, then we transform T_{k-1} into T_k by moving k's subtree (a tree rooted at k) to that position. It does not decrease

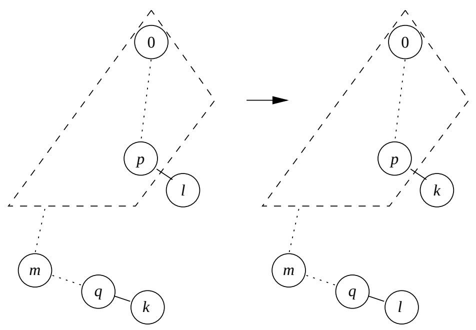

Figure 10. Converted Trees

any incoming rate of nodes in k's subtree, because k's position in T is chosen to maximize k's incoming rate when the Edge Bandwidth Assumption holds.

If k's position in T is occupied by node l in T_{k-1}, there are two possibilities depending on whether k is l's descendant or not. If it is, T_k is obtained by exchanging k and l. By (ii), k's incoming rate does not decrease. Since k has been chosen in Line 4, we know the following inequality holds.

$$r_k^{\text{in}} \geq r_l^{\text{in}\prime} \tag{3}$$

Therefore, by Eq. 3 and the Fair Contribution Requirement, the incoming rates of the nodes on the path from l to k in T_{k-1} do not decrease. There is also no change to the incoming rates of k's descendants in T_{k-1} because their ancestors remain same. The only concern is node l.

To calculate l's new incoming rate, suppose that p and q are parents of l and k in T_{k-1}, respectively. The left tree in Figure 10 represents T_{k-1}, and the right T_k. The area surrounded with a dotted line is the common part of T and T_{k-1}, and contains nodes $1, 2, \ldots, k-1$.

Then $r_p^{\text{out}} = r_p^{\text{out}\prime} \geq r_q^{\text{out}\prime}$ by the algorithm. Because the new incoming rate of l is $\min\left(r_q^{\text{out}\prime}, b_l^{\text{in}}\right)$ by the 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 not less than the previous value because it is the maximum q can get. 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 4. In this case the net effect for k's and l's incoming rates is as follows.

$$\begin{aligned}&(k\text{'s rate change}) + (l\text{'s rate change}) \\ &\quad = (r_k^{\text{in}} - r_k^{\text{in}\prime}) + \left(r_k^{\text{in}\prime} - r_l^{\text{in}\prime}\right) \geq 0 \qquad \text{(by Eq. 3)}\end{aligned}$$

Therefore T_k satisfies both properties.

If k is not l's descendant, T_k is obtained by exchanging k's subtree and l's subtree. Since we have shown that neither k's incoming rate nor the sum of k's and l's incoming rates doesn't decrease by exchange, it suffices to show that the incoming rates of l's descendants do not decrease.

Before calculating 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 q 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 4, and accordingly $\frac{1}{c_l} b_l^{\text{out}} \leq \frac{1}{c_k} b_k^{\text{out}}$ by the Fair Contribution Requirement. By this and the previous claim, we get $r_q^{\text{out}\prime} \geq \min\left(b_l^{\text{in}}, \frac{1}{c_l} b_l^{\text{out}}\right)$. Since q provides a higher rate than l can forward to its children, the incoming rates of l's descendants do not decrease. Therefore, we can obtain T_k with both properties.

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

B. Proof of Theorem 2

Proof We first prove a stronger version of the theorem under an assumption that all incoming and outgoing bandwidths are distinct. The stronger version is that, by DISTRIBUTED-OPTIMAL-TREE, a tree converges to the one obtained by CENTRALIZED-OPTIMAL-TREE. 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. We use induction on nodes. The base case is trivial, because node 0 is fixed.

Suppose as induction hypothesis that, while running DISTRIBUTED-OPTIMAL-TREE, nodes $1, 2, \ldots, k-1$ forms the same topology as they have in T. No node among them will move because their $\langle$*probe*$\rangle$ messages are discarded in Lines 6–7 of the distributed algorithm.

Consider node k. If k is already in the same position as in T, we are done. Otherwise, k's incoming rate must be lower than k's incoming rate in T because T is an optimal tree and all bandwidths are distinct (no tie). Eventually k sends a $\langle$*probe*$\rangle$ message to 0 because 0 clearly satisfies the condition in Line 3 if k is not at an optimal position. (Sending a $\langle$*probe*$\rangle$ message to a non-root ancestor can accelerate the convergence, without compromising this proof.) k cannot receive more than it does in T because all such positions are filled out by nodes $1, 2, \ldots, k-1$. However, it keeps sending $\langle$*probe*$\rangle$ messages until it reaches k's parent in T. Since k is the best node among the remaining ones, k beats any node in Line 7 and moves to its optimal position.

By induction, all nodes move into their optimal positions and result in forming a tree equal to T.

If bandwidths are not distinct, we may encounter ties, but they do not affect incoming rates. Therefore, DISTRIBUTED-OPTIMAL-TREE converges to a tree with the same rate vector as T. □