End-host multicast communication using switch-trees protocols

David A. Helder Sugih Jamin
University of Michigan
{dhelder, jamin}@eecs.umich.edu

Abstract— **Switch-trees are peer-to-peer algorithms for building and improving end-host multicast trees. Nodes switch parents to reduce tree cost or lower source-member latency. A node switches parents by disconnecting from its parent and reconnecting to a new parent. If the new parent is well chosen, the performance of the tree is improved overall. We look at the performance of switch-trees using the following metrics: cost, latency, link stress and number of switches. Simulations show switch-tree algorithms can build trees of hundreds of nodes at less than twice the optimal cost. In addition, we describe our implementation of a switch-tree protocol. Experiments show that our protocol builds low-cost trees in practice.**

Keywords: end-host multicast, peer-to-peer

I. Introduction

Many peer-to-peer (P2P) systems and network applications require the ability to send data to multiple destinations. For example, a multimedia conferencing application sends audio and video data from one participant to the others. A network game sends position updates and user actions between players. A P2P file sharing program sends search queries from one user to others. As these sort of applications become more and more popular, the ability to *efficiently* send data to multiple destinations is essential.

Multicast is a network technology that allows a host to efficiently send data to a group of other hosts. IP Multicast is an implementation of multicast for IPv4. IP Multicast uses special addresses to which the sources send data and which the receivers join to receive the sent data. Routers maintain group state and perform the necessary routing. However, for technical and administrative reasons, IP Multicast has not been globally deployed on the Internet.

Another approach to multicast is *end-host multicast* (EM). The idea is that hosts in the multicast group connect to each other to form a virtual network (i.e., a connected graph). The hosts route multicast data themselves over the virtual network so that it reaches all members. EM does not require support from routers beyond regular unicast forwarding. Examples of EM include Overcast [7], Narada [1], ALMI [10], and Yoid [2].

Figure 1 shows an example end-host multicast graph (solid lines) on top a physical network (dotted lines). In this example, nodes A, B, C, D, and E (but not s) are members of the multicast group and connect over the physical network to form a virtual network. Consider how node D routes multicast data. When node D receives data from node B, it unicasts it to nodes C and E. When node D sends data (i.e., D is the originator), it unicasts it to B, C, and E.

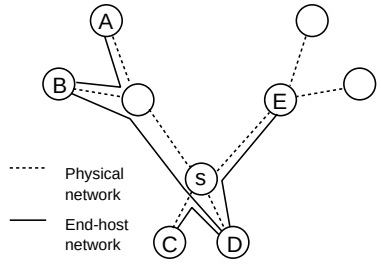


Fig. 1. Network and end-host multicast tree

End-host multicast has several problems: it will likely use more network resources than IP Multicast (e.g., in the previous example, three multicast packets would cross node s in EM while in IP multicast only one would), it may not scale well beyond a few hundred users, and it requires a bootstrap process for members to join the group. However, it can be easily and immediately deployed since it can be implemented at the user level. EM is suitable for many applications that require small multicast groups (tens to hundreds of members) such as video conferencing, network games, and P2P file sharing. Until IP multicast is widely deployed, EM can be used to provide multicast service.

Multimedia conferencing applications and network games require low latency delivery. Data sent by a participant should reach all other participants as soon as possible. Ideally, the EM topology is a shortest-path-first (SPF) tree, which has the lowest possible source-member latency. P2P file sharing applications do not require low latency so may use low cost groups. Low cost groups attempt to use as few network resources as possible. Ideally, the topology is a minimum-spanning tree (MST), which has the lowest possible cost. In Section II, we show building SPF trees and MSTs for EM is impractical because of bandwidth constraints and protocol complexity.

In Section III, we present practical algorithms for building and improving EM trees. We call these algorithms *switch-trees* because the nodes switch parents to reduce tree cost or lower source-member latency. Parent switching is where nodes disconnect from their parent and reconnect to a new parent. If the new parent is well chosen, the tree can have low cost, like an MST, or low source-member latency, like an SPF, while obeying any bandwidth constraints. We present four parent-switching algorithms of varying complexity and performance. Sim-

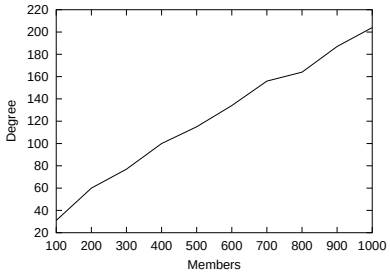


Fig. 2. MST maximum degree on various topologies

ulation results are presented in Section IV. In Section V, we present the Banana Tree Protocol (BTP), a switch-tree protocol we designed and implemented. Experimental results show BTP can create low-cost trees for small groups.

II. Problems with optimal solutions

An end-host multicast node resides on a host on a network. A node's *degree* is the number of virtual links between it and other nodes in the EM tree. Each host has at least one *interface*, a physical link between it and another host on the network. Degree affects *link stress*, the number of virtual links that map onto an interface. If link stress is high, there is greater contention for an interface and congestion and packet loss are more likely. Link stress should be minimized to avoid poor performance. To minimize link stress, we limit the maximum degree each EM node can have.

Constructing shortest-path-first (SPF) trees and minimum-spanning trees (MSTs) is impractical because of degree constraints. A SPF tree rooted at the source would provide the theoretical best performance. The shortest path between two hosts is a direct unicast connection, hence in this tree all non-source hosts will be directly connected to the source. Practically, any given host may not have the bandwidth to support more than tens of concurrent connections. A node in an MST can have high degree also, as shown in Figure 2 (experiment details are described in Section IV). Each point plotted is the median maximum node degree of 100 MSTs. Each MST was built over a complete graph connecting nodes randomly selected from a topology of 8000 hosts. Even with small multicast groups, at least one node was connected to many nodes.

Constructing a SPF tree is simple — members can simply connect to the root. Constructing a MST, however, is impractical for a large multicast group due to protocol complexity. All inter-node distances must be known in advance to build the MST. Stoica et al. [3] describes a completely distributed algorithm for building an MST, but construction takes $O(n)$ time where n is the number of nodes. In addition, MSTs cannot be incrementally updated. If a new node joins or an existing member leaves, the tree must be rebuilt.

III. Switch-trees

Switch-trees are practical algorithms for building and improving EM trees while obeying any given degree limits. Our strategy is to perform local transformations to improve overall tree performance. A complex algorithm that allows a wide range of tree transformations produces a near-optimal tree, but we show that our relatively simple algorithms produce trees with good performance.

Each tree has a single root. If there is a single multicast source, the source is the root. Otherwise, the root is arbitrarily chosen. In practice, the root may be the member that initially created the group. The root determines a parent-child relationship between nodes: the root has no parent and a non-root node's parent is the first node on the path to the root through the tree. We do not specify a method for a node to join or leave the tree. In our protocol, described in section V, new nodes initially connect to the root. When a node leaves the group its children reconnect to the root.

We call our algorithms *switch-trees* because nodes switch parents to reduce tree cost or latency. That is, nodes disconnect from their parent and reconnect to a new parent. Theoretically, a node can switch to any node that is not a descendant at any time. If a node were to switch to a descendant, it would create both a partition and a loop. In addition to forbidding switches to descendants, we forbid switches that would cause a node to exceed the degree limit.

We now define a family of switch-tree algorithms which allow switching to different sets of nodes. Each algorithm subsumes the previous algorithm. The switch-trees are illustrated in Figure 3. In the figure, the gray node is the node performing a switch and the black nodes are the nodes that it can switch to (i.e., make its parent). In *switch-sibling* a node can switch to any of its siblings. Two nodes are siblings if they share the same parent. In this algorithm, nodes move away from the root as they switch to nearby siblings. In *switch-one-hop* a node can switch to any node within one hop of its parent, namely its grandparent and any of its siblings. In *switch-two-hop* a node can switch to any node within two hops of its parent, except to one of its own children. In terms of relationships, this includes its grandparent, great-grandparent, uncles, siblings, and nephews. In *switch-any* a node can switch to any node that is not its descendant.

By parent switching, switch-tree algorithms can build a tree that has either low tree cost, like an MST, or low source-member latency, like an SPF, while observing any degree constraints. To reduce tree cost, a node attempts to switch to the closest node possible, where closeness is determined by the round-trip time. Consider the bottom right tree in Figure 4. Node A is the root, nodes B and C are initially its children, and tree cost is 6. If node C switches from node A to node B (bottom left tree), then

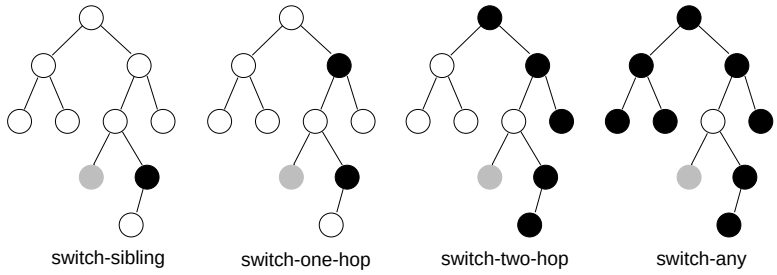


Fig. 3. Switch trees. The gray node can make any of the black nodes its new parent.

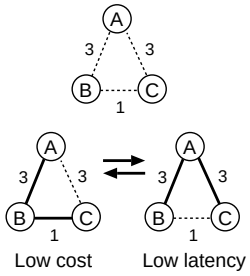


Fig. 4. Switching example

tree cost is reduced to 4, though the latency from A to C increases from 3 to 4.

To lower source-member latency, a node attempts to switch to a nearby node that already has a low latency to the root. Specifically, a node attempts to find a potential parent through which the node's distance to the root is minimized. We call this distance the *root-path latency*. Consider the bottom left tree in Figure 4. Node A is the root, node B is A's child, node C is B's child, and the latency from A to C is 4. If node C switches from node B to node A (bottom right tree), then the latency from A to C is reduced to 3, though the tree cost increases from 4 to 6.

If in an attempt to lower cost (or latency), a node finds that two potential parents are equally close (or have equal root-path latencies), the tie is broken arbitrarily. To avoid oscillation, a node will not switch if a potential parent is equally as close (or has an equal root-path latency) as its current parent. Ultimately, if the network remains stable and distances do not change, no node will be able to switch to a better parent and the tree will reach a fix-point.

While we are primarily concerned with tree cost and latency, the metrics most visible to the end user, we also aim to reduce protocol overhead and minimize impact on the network. Protocol overhead is measured by total switches. Network impact is measured by maximum link stress, previously discussed in Section II.

Total switches is the total number of switches made by all nodes over the lifetime of a group. We would like to minimize total switches for the following reasons. First, a switch incurs some overhead in processing and bandwidth. Second, a switch can cause a shift in network traffic if the added connection adds traffic through an interface or the removed connection removes traffic. Other hosts using the interface must adapt to any change (e.g., a loss in throughput). Third, multicast data can be lost or duplicated during a switch. A switching node may not receive a packet sent in the period between the time it removes the link to its old parent and the time it adds a link to its new parent. Alternately, if it adds a link to its new parent before removing the link to its old parent, it may receive a duplicate packet. While the application could handle duplicated data or recover from loss, it adds additional processing and communication overhead.

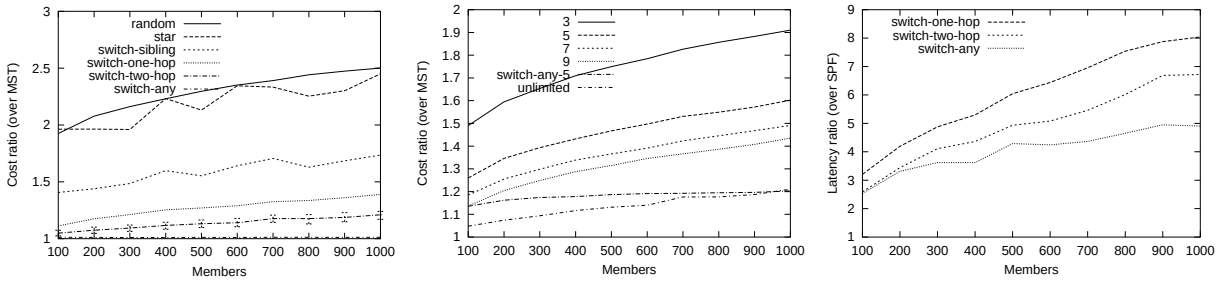
IV. Experiments

In this section, we evaluate switch-trees through simulation. We developed an end-host multicast tree simulator that simulates the building and improvement of various trees and computes statistics for each tree. We find that the switch-one-hop and switch-two-hop algorithms best balance our performance metrics.

Trees are built over simulated Internets generated using the topology generator *Inet* [8]. The simulator was run on three different Inet-generated networks. Each run produced qualitatively similar results; for legibility of the graphs, we present results from only one topology. The network we use has 8000 hosts. The host with the most interfaces has 1502 interfaces. Inet embeds the network in a plane of 10,000 by 10,000 distance units. The tree cost of a link is the Euclidean distance between the end hosts.

In our simulations, we vary the number of group members from 100 to 1000, stepping by 100 members. For each group size, we run 100 tree building and transformation simulations. In each simulation, a group and its root are selected randomly. Time is divided into discrete rounds. At the end of each round, each node is allowed to switch once. If no nodes switch, the tree is in equilibrium. Nodes are added incrementally. At the beginning of each round, a single node is added if there are more nodes to be added and the tree is in equilibrium. Each added node is initially connected to the root.

In the first experiment, we show that the switch-tree algorithms can create trees with low cost. The ratio of switch-tree cost to the MST cost, the theoretical lower bound, is shown in Figure 5.a. The more potential parents a node can switch to, the more likely it will find a nearby parent and reduce the tree cost. Two extreme tree building algorithms are included in this experiment—*random*, where nodes switch to random non-descendants, and *star*, where nodes connect directly to the source and do not switch. All switch-trees perform better than random and star, demonstrating that informed parent choice is required for low cost. The most limited switch-tree, switch-sibling, has the highest tree cost of all switch-trees. The least limited switch-tree, switch-any, has the



a. Ratio of tree cost to MST cost

b. Tree cost ratio of switch-two-hop with varying degree

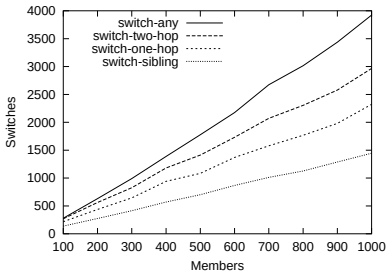
c. Ratio of average latency to net-SPF latency

Fig. 5. Cost and Latency

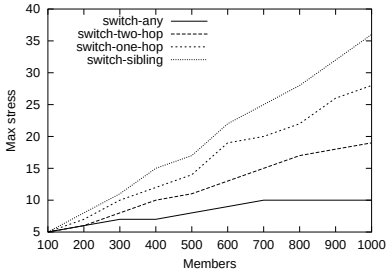
lowest tree cost, which is only slightly more than MST's. Switch-any is a good approximation of MST. All switch-trees scale well—the added cost of adding another member is small relative the cost of the tree. Error bars (at the 25th and 75th percentile) are shown for switch-two-hop. The range is small, so we do not include them in latter graphs.

We now consider the affect of degree on tree cost. Degree can be high in an MST, as previously shown in Figure 2. This suggests there are nodes that are natural hubs. If the degree limit is raised, these nodes can have more children. In addition, a node with the maximum number of children prevents other nodes from switching to it. A node may not be able to switch to its best potential parent because it would have to first switch to a "full" node. If the degree limit is raised, nodes are less likely to block other nodes from switching. For these two reasons, switch-tree algorithms will build trees with lower cost if the degree limit is raised. This is demonstrated in Figure 5.b, which shows the switch-tree cost to MST cost ratio of switch-two-hop trees of varying degree limits. For comparison, the figure includes switch-two-hop with no degree limit and switch-any with a degree limit of 5. Degree-unlimited switch-two-hop outperforms degree-limited switch-any. From this, we conclude that degree limit has a more significant affect on tree cost than the particular switch-tree algorithm.

Next, we show that switch-trees can create trees with low source-member latency. A node reduces latency by switching to a nearby node that already has a low latency to the root. In this experiment, the SPF tree is built over the *physical* network (denoted *net-SPF* henceforth). A net-SPF tree is similar to a tree generated by a source-rooted IP multicast protocol. A star switch-tree has an average latency equal to a net-SPF. If degree were unlimited, any switch-tree algorithm would simply form a star, thus we limit degree to 5 in this experiment. As shown in Figure 5.c, switch-any performs best, at the expense of between 2.6 and 5 times greater average latency than net-SPF. Switch-one-hop and switch-two-hop perform slightly worse, with between 3 and 9 times greater average latency than net-SPF. Switch-sibling and random



a. Total number of switches



b. Maximum link stress

Fig. 6. Number of switches and link stress

perform the worst, between 5 and 50 times worse than net-SPF, and are not included in the figure. In switch-sibling, nodes can only move away from the root, so the tree has a greater depth. In the other switch-tree algorithms, nodes can move both up and down the tree.

In our final experiment, we consider number of switches and maximum link stress in switch-trees built for low tree cost. Ideally, the total number of switches required for a switch-tree to reach equilibrium is low because a switch incurs some control overhead, causes a shift in network traffic, and may cause data loss or duplication, as explained in Section III. Switch-tree algorithms with more switch candidates switch more often, as shown in Figure 6.a. Switch-any switches the most; switch-sibling switches the least. However, switch-sibling has the greatest maximum link stress and switch-any the least, as shown in 6.b.

In conclusion, while switch-any has the lowest cost (or latency) and stress, it requires the most switches. Switch-sibling switches less frequently, but has the highest cost (or latency) and network stress. Switch-one-hop

and switch-two-hop provide a good balance of the metrics overall. In the next section, we describe a switch-one-hop protocol we have implemented and deployed.

V. The BTP Protocol

In this section we describe the Banana Tree Protocol (BTP), a peer-to-peer, fault-tolerant, switch-one-hop protocol. We have implemented BTP and currently use it in a peer-to-peer file sharing program and in a distributed Internet distance measurement system. Our purpose is to illustrate some of the issues that would be considered in EM protocol design, especially when using switch-trees. This is a high-level description that explains how nodes join, detect partitions, and switch parents to improve the tree. We do not include low-level details such as packet formats.

To send a multicast packet in BTP, as well as other tree-based EM protocols, a node forwards the packet to all of its neighbors. When a node receives a multicast packet from one of its neighbors it forwards the packet to its other neighbors. A host joins a group by connecting to the root and becoming its child. A node that joins a group with no current member becomes the root node.

If a node's parent leaves the tree or fails, a partition is formed. The node then reconnects to the root. This cannot create a loop because the root cannot be the node's descendant. If a node's child fails, the node does nothing. If the child had children, they will reconnect to the root themselves. Unfortunately, the root will be flooded with connections if many nodes fail. One solution is to have nodes connect to their grandparent, or some other non-descendant. However, if we assume nodes will not leave or fail frequently, connecting to the root is adequate. This is what is done in the current implementation.

A node improves the tree by switching to a nearby sibling or grandparent. Because the applications that use BTP do not require low latency, we attempt to reduce tree cost. Each node will occasionally request from its parent a list of the parent's neighbors. It will then ping its parent and each node in the list. If it finds a node that's closer to it than its parent, it will initiate switching to that node. We will call this node the *potential parent*. When switching, care must be taken to ensure when the switch occurs that 1) the potential parent is not simultaneously attempting to switch to another node and 2) the potential parent is still the node's sibling or grandparent. We now discuss these two cases in detail.

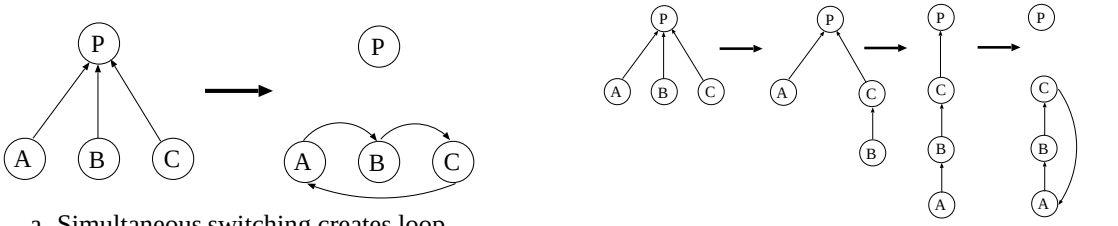
When a node wants to switch to a potential parent, it must first send a switch request to the potential parent and wait for an acceptance or rejection message. If two siblings switch to one another at the same time they would clearly form a loop. To ensure this does not happen, we could require that a node trying to switch to a potential parent will always reject a switch request from the potential parent. However, this policy is not sufficient to ensure loop freedom when the switch involves more than two nodes. Consider the case in Figure 7.a where node A attempts to switch to its sibling B. If simultaneously C switches to A and B switches to C a loop forms. To prevent such loops from forming, we adopted the policy that a node will reject *all* switching attempts if it is itself in the process of switching parents. This is a conservative policy in that there are cases where simultaneous switching can take place without forming a loop.

In addition, when a node attempts to switch to a potential parent, it must ensure that the potential parent is still its sibling or grandparent. For example, consider the case in Figure 7.b where nodes A, B, and C are siblings in the tree. Subsequently B switches to C and A switches to B. If C then switches to A, a loop is formed. This happened because A and C acted on out-of-date information—A believed B was its sibling and C believed A was its sibling. Hence when a node requests to switch to another, it must include its current parent information in the switch request. Before accepting the switch, the potential parent checks that either the node and itself are actually sharing the same parent, and therefore siblings, or the node's parent is its child, and therefore its grandchild. This is again a conservative policy. For example, in the above scenario, after B switched to C, A will be prevented from switching to B even though no loop would be formed by the switch.

We now describe an experiment that shows BTP produces a low cost tree in practice. In our experiment, we compare the cost of a MST to the cost of a BTP tree. The MST is calculated from the round-trip-time measurements between each pair of hosts. The cost of the BTP tree is determined by traversing the tree to discover the tree's edges and then summing the corresponding round-trip-time measurements.

In this experiment, the end-host multicast group consisted of nineteen members. Each member was located on one of nineteen hosts. Seven hosts were at U.S. educational sites, six at European educational sites, three at U.S. commercial sites, and two were in U.S. homes. Two of the US educational sites were on the west coast and the other five on the east cost. The six European educational sites were in England, France, Germany, Greece, Sweden, and Switzerland. The three commercial sites were in Washington, D.C., California, and Michigan. The two in homes were both in Ann Arbor, Michigan and connected to the Internet by cable modem.

Round-trip-time was measured using the TCP Ping utility [11], based on the technique described in [4]. Each host made eight round-trip-time measurements to every other host. Therefore, for each pair of hosts there were sixteen round-trip-time measurements in total. Of the sixteen measurements, only the minimum is used in our cal-



a. Simultaneous switching creates loop

b. Switching with outdated information creates loop

Fig. 7. BTP switching examples

culations.

The experiment was conducted on November 13, 2001 and consisted of ten runs. There was a two-hour interval between each run. In each run, we measured the round-trip-times between the hosts and then ran our BTP client on each host. Node degree was limited to five. Each node requested a neighbor list from its parents every twenty seconds, pinged each node in the list, and attempted to switch if it found a better potential parent. The root of the BTP trees was at the University of Michigan. After starting the clients, the nodes were allowed to switch for ten minutes and then the tree was traversed to discover the edges. An MST was calculated from the round-trip-time measurements. The cost of the BTP tree was calculated from the round-trip-time measurments and the edges discovered during the traversal.

Ideally, the ratio of the cost of the BTP tree to the cost of the MST is 1. The median cost ratio in our experiment was 1.19. The mean ratio was 1.27, the minimum 1.11, and the maximum 1.83. In the case of the maximum ratio, over half of the BTP cost came from a single link between Greece and Britain. In this run, round-trip-times to Greece were much higher than in other runs, indicating its network was temporarily congested. In all ten runs, the BTP tree had only one link between the west and east coasts of the U.S. and only one link between the U.S. and Europe.

VI. Related Work

Yoid builds both a tree and a mesh [2]. The tree is used for normal data transfer and the mesh is used for control data, fault tolerance, and partition detection. *Overcast* builds a single-source EM tree that maximizes bandwidth available from the source to members through the tree [7]. Nodes perform switch-one-hop-like optimizations to find good positions in the tree. *ALMI* builds an MST in a centralized fashion [10]. A controller ensures member connectivity, handles failures, and periodically recalculates and rebuilds the MST. *Hypercast* builds a hypercube over which source-specific trees are built for each member [5]. Hosts join the hypercube in a strict order and are assigned Grey-code node IDs, used for routing. *Narada* creates a mesh and then builds delivery trees over the mesh using a DVMRP algorithm [1]. Nodes periodically exchange distance vectors which are used for routing, optimization, and partition detection.

VII. Conclusion

We have presented switch-trees, a family of algorithms for building and improving end-host multicast trees. Switch-any switch-trees have the lowest cost or latency but they require many switches to achieve this. Switch-sibling trees have highest cost or latency and require very few switches. A good balance is provided by switch-one-hop and switch-two-hop trees. We have implemented BTP, a switch-one-hop protocol, and shown that it produces low cost trees. BTP is currently used in Jungle Monkey (JM) [9], our P2P file sharing program, and IDMaps [6], our distributed Internet distance measurement system. The source code for BTP is available for download as part of our open-source EM toolkit included in JM.

References

[1] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang. A case for EndSystem multicast. *Proc. of ACM SIGMETRICS'00*, pages 1–12, June 2000.

[2] Paul Francis. Yoid: extending the internet multicast architecture. Technical report, NTT, April 2000.

[3] I. Stoica H. Abdel-Wahab and F. Sultan. A Simple and Fast Distributed Algorithm to Compute a Minimum Spanning Tree in the Internet. *Proc. of Joint Conference on Information Sciences '95*, pages 429–433, 1995.

[4] Martin Horneffer. Assessing Internet Performance Metrics Using Large-Scale TCP-SYN Based Measurement. *Passive and Active Measurement Workshop*, 2000.

[5] Tyler K. Beam J. Liebeherr. A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology. *Proc. First International Workshop on Networked Group Communication (NGC '99)*, pages 72–89, 1999.

[6] Sugih Jamin, Cheng Jin, Yixin Jin, Dan Raz, Yuval Shavitt, and Lixia Zhang. "On the Placement of Internet Instrumentation". *Proc. of IEEE INFOCOM*, March 2000.

[7] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, pages 197–212, October 2000.

[8] C. Jin, Q. Chen, and S. Jamin. Inet: Internet Topology Generator. Technical Report CSE-TR-433-00, EECS Department, University of Michigan, 2000.

[9] Jungle Monkey homepage, November 2001. http://www.junglemonkey.net.

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

[11] Amgad Zeitoun. TCP Ping homepage, November 2001. http://www.eecs.umich.edu/ azeitoun/tools.html.