Banana Tree Protocol, an End-host Multicast Protocol

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

Abstract— **Multicast is a network technology that allows a host to efficiently send data to a group of hosts. IP multicast is one example of multicast that relies on Internet routers to forward multicast packets to multicast group members. "End-host multicast" is another approach where the hosts in the multicast group form a virtual network and route multicast data through the graph themselves without router cooperation.**

This paper describes Banana Tree Protocol (BTP), an end-host multicast protocol we designed and implemented. We have simulated BTP along with other multicast protocols and theoretically optimal virtual networks. We find BTP performs well in optimal conditions, but performs poorly in more realistic conditions. We analyze this behavior and find BTP to be too limited in its allowed graph transformations. We conclude that an end-host multicast protocol must allow nodes to make a wide range of graph transformations in order to effectively optimize the graph.

Keywords— **end-host multicast, multicast**

I. Introduction

A. Motivation

Distributed file sharing (DFS) programs have recently become very popular. Programs like Napster [1] and Jungle Monkey [2] allow users to download files from each other in a peer-to-peer fashion. Often these programs use meta-servers to find other users or search servers to search for files, but most notably there is no central file server. The advantage of DFS is that users can download popular files from nearby hosts. They need not connect to an overloaded central server. Unfortunately, copyright issues aside, network administrators are isolating their networks against Napster because of the large amount of bandwidth it requires. Multicast would be an ideal solution to this problem.

Multicast is a network technology that allows a host to efficiently send data to a group of hosts. IP Multicast is an implementation of multicast for IPv4. IP Multicast uses special addresses which the senders send to and the receivers join to receive the sent packets. Routers maintain the group lists and perform the necessary routing.

IP Multicast has several problems: it is not widely deployed, has a small address space, and can have long join latencies. Solutions have been proposed to solve some of these problems, however, for reasons technical and administrative, 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 use the virtual network to route multicast data so that it reaches all members. End-host Multicast does not require support from routers beyond regular unicast forwarding.

The disadvantages of EM are that it requires a bootstrap process to join the group, may not scale well beyond a few hundred users, and will likely use more network resources than IP Multicast. However, it does solve most of the problems with IP Multicast: it can be easily deployed since it can be implemented at the user level, the address space can be unlimited, and there is often no join latency problems. EM would be suitable for many applications that require small multicast groups such as video conferencing, network games, and distributed file sharing.

B. Banana Tree Protocol

We have designed and implemented an end-host multicast protocol called the Banana Tree Protocol (BTP). BTP was designed for our own distributed file sharing program, Jungle Monkey (JM) [2]. The latest version of JM uses BTP. BTP builds multicast trees and primarily provides tree maintenance and optimization functions. There are two protocols built on top of BTP that provide additional features that JM requires: Banana Tree Simple Multicast

Protocol (BTSMP) and Banana Tree File Transfer Protocol (BTFTP). BTSMP provides many-to-many group communication for sending and receiving packets. JM uses BTSMP to announce files that the user has available for download. BTFTP provides one-to-many file distribution. Ideally, BTP connects the hosts together in a way that does not waste network resources. For example, if two hosts are close together in the network, they should be close together in the virtual network. Using BTP for peer-to-peer file transfer allows a member to transfer file from a nearby neighbor in the virtual network (ideally, the file is transferred from the closest host possible). JM uses BTFTP for file transfer.

II. Design issues

In end-host multicast, the hosts in the multicast group form a virtual network and route multicast data through the graph themselves without router cooperation. We will often refer the virtual network as the "graph" and hosts in the virtual network as "nodes".

A. Design goals

We make the following assumptions to simply discussion of end-host multicast protocols.

Packets. The unit of data is a packet which contains an application-level frame. We do not specify a transport layer. The transport layer may be stream-based or datagram-based. Hence the protocol does not need to be reliable. Packets may be lost, duplicated, or corrupted. Protocols layered above an end-host multicast protocol must provide reliability if it is desired.

Distributed. The protocol must be distributed. The hosts are solely responsible for maintaining the graph and fault tolerance. Using a centralized server is not desirable because it is a single point of failure and can be a bottleneck.

Dynamic membership. Group membership must be dynamic. Hosts must be able to join and leave the group at any time.

Self-correcting, self-optimizing topology. The protocol must allow hosts to dynamicly modify the graph to optimize it and to repair graph partitions. Hosts optimize the graph by adding and removing links to improve its performance. Performance is measured by latency, cost, degree, or other metrics. These metrics are defined and discussed in the next section. Hosts add links to repair graph partitions. Graph partitions can occur if a host leaves or fails.

Multiple senders. There are multiple senders in the multicast group. Alternatively, we could have assumed that there is a single-sender and use multiple groups when there are multiple senders. However, this may be inefficient if there are more than a few senders. If one of the sender sends more than the others, e.g. in a distant lecture application where participants may ask an occasional question, we call the sender that sends more the *dominant* sender.

Multicast routing only. The protocol provides multicast routing only. Unicast routing may be added to run conventional routing algorithms over the topology. However, this is not a requirement.

Links are bidirectional. Links in the graph are bidirectional. That is, if node A can forward packets to node B, node B can forward packets to node A. We will also assume there is only connection between two connected hosts.

Scaling. The group does not need to scale to more than a few hundred members.

B. Topology issues

The topology for an end-host multicast protocol is important because it affects how later design issues are addressed. These issues include routing, graph optimization, and partition avoidance, detection, and repair. We will first discuss these issues in section II-B.1 and then discuss how two topologies—trees and meshes—could address these issues in section II-B.2.

B.1 Topology design issues

Routing. When a node receives a packet, it must decide which neighbor(s) to forward the packet to. Ideally, the packet will reach each host once and only once.

Optimization. Nodes can modify their connectivities to optimize the performance of the group. In this section, we consider three performance metrics: cost, latency, and degree. These metrics are presented in [3]. As usual, these performance metrics represent trade-offs in the design space.

1. *Latency* is the average distance between two nodes in the virtual network. We define the distance between two adjacent nodes to be the network

round-trip-time between them. We will also call this distance *link cost*. We define the distance between two non-adjacent nodes to be the sum of the link costs of the edges in the path through the graph between the nodes.

Latency matters in interactive applications such as video conferencing and network games. Some applications, such as collaborative work applications, require a low node-to-node latency. In these applications, a complete graph (i.e., the graph formed by connecting every node to every other node) would provide the theoretical best performance. Other application require a low sender-to-node latency when a single sender dominates, such as in live video broadcasts. In these applications, a tree formed by a shortest-path-first algorithm based at the sender would provide the theoretical best performance.

2. *Tree Cost* is the sum of the link costs a packet travel across. Edges on the graph that a packet does not cross are not counted. Theoretically, the graph with the lowest possible cost is a minimum spanning tree (MST) built over the complete graph of nodes (or, an MST with extra, unused edges). Although cost does not easily translate into an easily-measurable, real-world metric, such as latency, it does indicate how efficiently a graph uses network resources. Note that a graph with a low cost may have a high latency and vice versa.

3. *Degree* is the number of nodes a node is connected to in the graph (virtual network). In contrast to physical network where the maximum degree of a node is determined by the number of network interfaces it has, the maximum degree of a node on the virtual network may determined by the bandwidth available to the node. There is a trade off between low latency and a smaller node degree, up to the capacity of its links. If all nodes have a high degree, the average number of hops between two nodes is low and node-to-node latency is low; but a high-degree node increases the stress on nearby links and thus increases the chance of congestion. If all nodes have a low degree, the average number of hops between two nodes is high. This means node-to-node latency is high, but a low-degree node reduces the stress on nearby links and thus lowers the chance of congestion.

Partitions. A graph that is not connected is said to have one or more *partitions*. An EM protocol is said to perform partition avoidance if it ensures that graph transformations preserve the connectedness of the graph. The protocol performs partition detection and repair when the protocol identifies a partition and then adds edges to repair the partition. The protocol must perform partition detection-repair because a node may leave or fail and create a partition. Partition avoidance is not necessary, but may reduce the need for detection-repair, which may be slow or costly.

The greatest practical constraint in building an end-host multicast delivery network is the node degree. A node cannot have hundreds of connections due to bandwidth limit and the stress placed on nearby links. Ten connections is a more realistic number. This then excludes star topologies, complete graphs, and graphs with high degree requirement on some or all nodes. In addition, the graph should not have extremely poor latency. A graph will have poor latency if it includes long chains. This means the graph must have some nodes of a degree greater than two. This then excludes chains and rings.

B.2 Trees and meshes

We now discuss trees and meshes in relation to the topology design issues—routing, optimization, and partitions. A tree is a natural choice for end-host multicast delivery network because it has the optimal cost and latency. Nodes in a tree do not need a large degree. Theoretically, the average degree of a node in a tree is just less than two. We will also consider a mesh. A mesh is a graph, possibly with loops, with nodes of low degree.

Routing is simple on a tree because a tree is, by definition, loop-free. To route a packet, a node only needs to know who its neighbors are. When the node receives a packet from a neighbor it forwards the packet to its other neighbors. The packet cannot loop and will eventually reach all nodes.

Routing is more difficult on a mesh because a mesh may have loops. Nodes must ensure that packets do not loop and must avoid sending packets that would be duplicates. That is, routing should be such that each packet is received by each node once and only once. Nodes could run a conventional routing protocol, such as distance-vector or link-state. With the resulting routing information,

they could then use reverse path broadcast or other broadcast routing algorithms. Nodes could also flood the mesh, but allow a neighbor who receives a duplicate packet to quench future packets. Any solution requires the node to store and maintain some state for routing.

Optimization can be difficult when using a tree. A node cannot simply add a link to a new neighbor because it will form a loop. A node cannot remove a link because it will create a partition. Instead, a node must add a link and remove a link simultaneously and ensure that this action does not create a loop or partition. This may require coordination with several other nodes or knowledge of the topology.

While a node on a mesh can easily add links because loops are allowed, it cannot easily remove them without the risk of creating a partition. A node may need to coordinate with several nodes or have knowledge of the topology. Another strategy would be to only remove links that are unlikely to create a partition and perform partition detection and repair when this fails.

Partition detection is easy on a tree. If a node leaves or fails, there must be a partition. Repair is more difficult because a node must add a connection, but if it adds one to a node in its own partition, it will only create a loop and not fix the partition. If more than one node attempts to repair the partition concurrently, they may create a loop.

Partition detection is more difficult on a mesh. A node may leave or fail and the graph may still be connected. A node could act conservatively and add additional links when a neighbor leaves or fails. Nodes could also rely on heartbeats to determine if there is still a path to an arbitrary set of other nodes. If it does not receive a heartbeat from any node in the set it attempts to add a link to the node or replaces the node with another one.

Each topologies has some difficult issues. We ultimately choose a tree because routing and partition detection in a tree are simple. In the next section we describe our protocol and how we address the most difficult issue, optimization.

C. Other issues

There are some other issues that must also be considered in end-host multicast protocol design whose detailed treatments are outside the scope of this paper.

Denial of service. We define a *denial-of-service attack (DSA)* against a multicast group to be a action made by a malicious host intended to disrupt the correct functioning of the multicast group or any of its participants. Some DSA apply to multicast in general. Examples of attacks against multicast groups in general include flooding the group with packets, forging the sender address of packets, eavesdropping on a group, sending invalid data.

Because end-host multicast group members also act as routers, EM is prone to additional attacks. Examples of these attacks include modifying data, deliberately creating loops or partitions, and selectively forwarding data. Some of these attacks can be prevented using encryption or signatures.

Bootstrap. To join an EM group, the node must connect to the graph. To connect to the graph, the host must learn of some nodes in the graph that it can connect to. In the remainder of this paper, we assume this functionality is provided by another protocol.

III. The Banana Tree Protocol

Our end-host multicast protocol, Banana Tree Protocol (BTP), is based on a tree topology. Each host in the group is a node in the tree. The node may be a parent or a child. The host that created the tree is the root node and has no parent. All other nodes have a parent. The *parent* is the next node on the path to the root. A node may have multiple children. A *child* is the next node on a path away from the root. A node may be a parent, a child, or both. By definition, each parent is also another node's child. A node's parent and children are also its *neighbors*. When present, other children of a node's parent are its siblings. Nodes can change parent in some situations. If node A changes its parent from node B to node C, we say that node A *switches* to node C. By switching to siblings that are nearby, the tree minimizes the network resources used.

A host joins a group by becoming the child of a node currently in the tree, e.g. the root node. We assume the existence of a bootstrapping protocol by which a host can learn of a node in a multicast group. A node that joins a multicast group with no member becomes the root node.

To send a multicast packet, a node sends the packet to its neighbors. When a node receives a multicast packet from its neighbor it forwards the packet to its other neighbors.

If a node's parent leaves or fails, a partition is formed. The node then reconnects to the root. Note that 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. While there may be a more efficient way to repair a partition, nodes should not leave or fail frequently so this is adequate.

Optimization is performed through parent switching, discussed in the next subsection.

A. BTP and optimization

Nodes can switch parents to optimize the tree. Since the tree must remain loop-free at all times, we do not allow a node to switch to an arbitrary node. Otherwise, the node could switch to one of its descendants and create a loop and partition. In the previous section, we said that a node can switch to the root without creating a loop. A node can also switch to a sibling without creating a loop because a sibling cannot be a descendant of the node.

The purpose of being able to switch to a sibling is to optimize the tree for low cost. The node does this by switching to a sibling closer than its parent if such a sibling exists. Figure 1 shows an example. Part (a) shows three nodes and the distances between them. Node R is the root and nodes A and B are initially its children. The cost of the tree is 6. Part (b) shows node A switching to node B. The cost of the new tree is 4. Note that node B could have switched to node A instead. To measure the effectiveness of various switching mechanisms presented in Section IV-B, we define a *closeness* metric: $\mathcal{C} = d_s/d_c$, where d_s is the distance from a node to the closest node found by switching, and d_c is the distance from the node to the closest possible node in the tree. Distance is the estimated round trip time between two hosts.

Alternatively, nodes could optimize for low latency by not switching frequently in order remain close to the root. Because the application we had in mind when we designed BTP does not require low latency multicast groups, we choose to optimize the tree for cost.

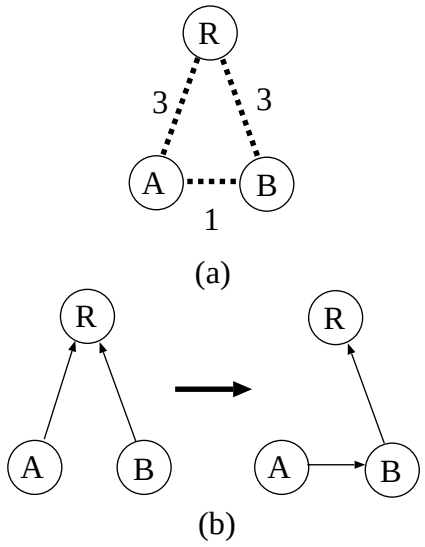

Fig. 1. Sibling switching in order to lower tree cost

When switching to a sibling, care must be taken to ensure that when the switch occurs that 1) the sibling is still the sibling and that 2) the sibling is not trying to switch to another node at the same time. We now discuss these two cases in detail.

Assume the node has received a list of its siblings from its parent and has found a sibling closer than its parent. For example, it may find this sibling by pinging each sibling or use an Internet distance service like IDMaps [4]. We will call this sibling the *potential parent*. 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. To ensure that siblings do not try to switch to each other at the same time, a node trying to switch to a potential parent will always reject a request from the potential parent to switch to it. Note, however, this policy is not sufficient to ensure loop freedom. Consider the case when node A tries to switch to its sibling B. There could be a third sibling C to which B is switching at the same time A is switching to B and C is switching to A (see Figure 2). To prevent such loops from happening, we adopted the policy that a node will reject *all* attempts at switching 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.

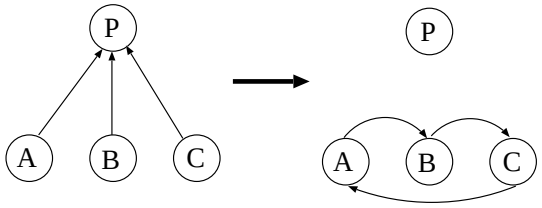

Fig. 2. Simultaneous switching creates loop

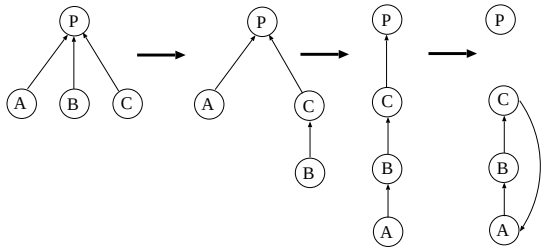

Fig. 3. Switching with outdated information creates loop

In addition, when a node attempts to switch to a potential parent, it must ensure that the potential parent is still its siblings. For example, suppose that 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 (see Figure 3). The cause is that A and C acted on out-of-date information. 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 the node and itself are actually sharing the same parent, and therefore siblings. 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.

B. BTFTP and BTSMP

Banana Tree Simple Multicast Protocol (BTSMP) and Banana Tree File Transfer Protocol (BTFTP) are built on top of BTP.

BTSMP provides many-to-many group communication for sending and receiving packets. BTSMP is mostly a wrapper around BTP provided to make BTP easier to use for the programmer. The only feature BTSMP adds to BTP is caching. Each node can keep a cache of the last few packets sent or received. When a new node joins the tree, it can request the contents of the cache from its parent. Caching improves a host's perceived join latency which is useful for some applications.

BTFTP provides reliable, one-to-many file transfer. It is assumed that the root has the file. A host that wants the file joins the tree and switches parents to find a nearby host. It then downloads the file from its parent.

IV. Evaluation

In this section we evaluate the performance of BTP through simulations. BTP provides two functions using two higher-level protocols: many-to-many multicast packet communication from BTSMP and one-to-many file transfer from BTFTP. We evaluate how well BTP provides each function separately.

The metrics we use to evaluate BTP for many-to-many packet communication are cost, latency, and degree. These metrics were defined and discussed in section II-B. Because the application we had in mind when we designed BTP does not require low latency multicast groups, we choose to optimize for cost over latency.

The metric we use to evaluate BTP for one-to-many file transfer is closeness, $\mathcal{C}$ as defined in Section III-A. If $\mathcal{C}$ is close to one, the host will be transferring the file from a close enough host.

We developed a multicast tree simulator called TREESIM. In our simulations, we assume there is a dominant sender. The simulator randomly selects this sender and a number of group members.

TREESIM uses networks generated by Inet. The Inet topology generator generates AS (Autonomous System) level random topologies following the observed characteristics of the Internet reported in [5]. A description of the Inet topology generator is presented in [4]. When group members are selected, a short edge is added to the topology to simulate the last few hops to the host within the AS.

In each experiment, we simulated groups of between 10 and 200 members (stepping by 10). We assume each group has a dominant sender, which we will call the *source*. For each group size, we ran 100 rounds with a different source and group in each round. For each statistic, we took the average

over all runs for a given group size.

A. BTP as many-to-many packet communication

In our simulations of BTP used for many-to-many packet communication, we simulated other types of trees for comparison.

End-host minimal spanning tree (endhost-MST). An endhost-MST is the minimal spanning tree built over a connected graph of all the nodes in the group. The cost of an edge is the distance between the two nodes through the network. An endhost-MST will have the lowest cost of all EM trees.

Net shortest path first (net-SPF). A net-SPF tree is formed by running the shortest-path-first algorithm on the group using all the edges in the physical (as opposed to virtual) network. This is the type of tree that is formed by Internet routers building source-based multicast trees.

BTP-simultaneous. A BTP-simultaneous tree is a BTP tree where all members join at once. At the start of the tree building process, all nodes start by connecting to the source. Subsequently, nodes switch parents until a fix-point is reached.

BTP-incremental. A BTP-incremental tree is a BTP tree where members join one at a time. For each node, we connect it to the source separately and then let nodes switch. When a fix-point is reached, the next node is connected to the source and the process is repeated.

Switch-any. A switch-any tree is like a BTP tree, except that a node can switch to any node as long as it does not form a loop. Yoid, discussed in Section V, uses switch-any. We allow nodes to switch until a fix-point is reached.

For each tree, we measures its tree cost, average latency, and maximum number of children. Tree cost is the sum of the link costs of each link in the tree. The cost of a link on the tree is the distance between the two end nodes on the physical network (i.e. the unicast latency). Average latency is the average latency between the source and a member. Maximum number of children is the degree of the node with the most children. Note that, in a tree, the degree of a non-root node is one more than the number of children it has.

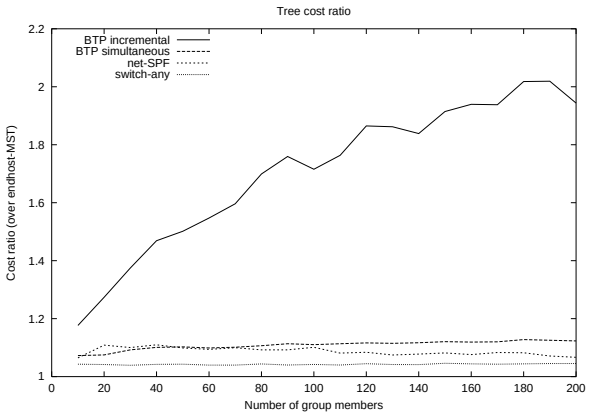

Fig. 4. Ratio of cost of various trees to the cost of endhost-MST

A.1 Cost

The best case for cost occurs when the BTP tree is a minimal spanning tree. We expect BTP to approximate a MST because it tries to use low cost links.

In our simulations, we measure the ratio of the cost of various trees to the cost of a minimal spanning tree. Figure 4 shows the results.

BTP-simultaneous is close to the cost of a MST, but BTP-incremental performs poorly. In BTP-simultaneous, nodes join the tree simultaneously so are initially all siblings. Since all nodes are siblings, nodes are more likely to initially find a very close parent. In contrast, in BTP-incremental a joining node has only a few siblings and so is limited in its initial choice of potential parents. Even if it switches several times, the node will have had fewer sibling in total than a node in the BTP-simultaneous simulation.

Unfortunately in practice BTP will be used more like BTP-incremental than BTP-simultaneous because most applications allow members to join over a period of time (this was even one of our goals). The lesson learned is that to effectively optimize for cost, a node needs to be able to examine a wide variety of nodes. A node should not be limited to switching to nodes at the same level of the tree only.

A.2 Latency

A net-SPF tree provides the theoretical best average latency. Although we optimize for cost, we

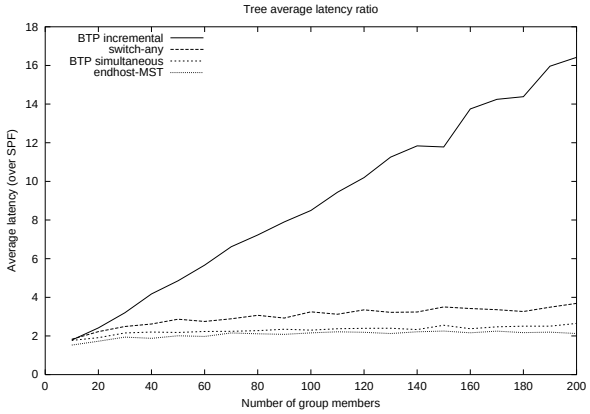

Fig. 5. Ratio of average latency of various trees to average latency of net-SPF

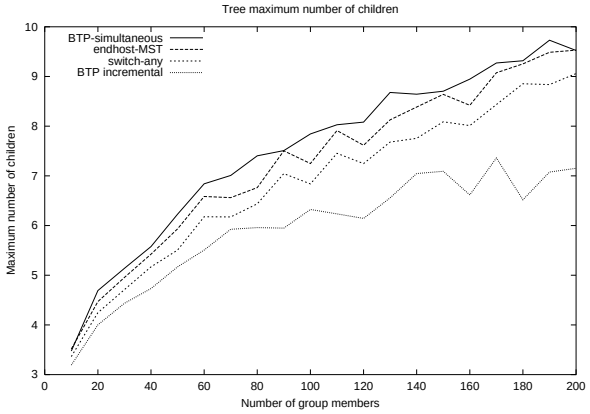

Fig. 6. Maximum number of children for various trees

expect BTP to have a reasonable average latency.

In our simulations, we measure the ratio of the average latency of various types of trees to the average latency of an net-SPF tree. Figure 5 shows the results.

Switch-any trees have greater latencies than BTP-simultaneous trees because nodes in the tree are more likely to find a closer parent, switch, and thus create a taller tree. Because a BTP tree considers fewer potential parents, it is less likely to find a better parent and switch, so a BTP tree is shorter.

However, BTP-incremental trees have considerably more latency than both switch-any and BTP-simultaneous trees. We suspect this is not because it switches more, but because the links used have a higher cost. This explanation is also supported by the cost experiment above.

A.3 Number of children

The best case for maximum degree occurs when the tree is a chain of nodes. The worst case for maximum degree occurs when the root has all nodes as its children. We want to avoid both these extreme cases. A chain of nodes has poor latency and a star topology puts considerable strain on the root. Simulations show that these cases does not occur in practice.

Figure 6 shows the maximum number of children of various types of trees observed in our simulations. In all cases, the maximum degree is acceptably low. Note that theoretically, the average degree of any tree will be just less than two.

B. BTP as one-to-many file transfer

In this section, we simulate BTP used for one-to-many file transfer. When used in this way, a node joins a tree where all members have the file. It quickly switches parents until it cannot switch to a closer parent. It then downloads the file from its parent.

We look a three different types of BTP trees. In the first two types, parents have a target minimum number of children of 4 and 8. If a node's parent does not have the target minimum number of children, it switches. We set a target minimum number of children so that switching nodes will examine more nodes before switching. This will increase the chance that it finds a close node. In the third type, there is no minimum. For comparison, we also consider the case when a node does not switch at all and downloads the file from the root.

In evaluating the different types of BTP trees, we use the closeness $\mathcal{C}$ metric defined in Section III-A. Figure 7 shows the results of our simulations. Hosts that use BTP to find a close host will find a considerably closer host than the root. Setting a minimum number of children improves the closeness. This is because nodes have more children. When a node examines siblings when trying to switch, it examines more nodes, so is more likely to find a very close node.

V. Related Work

In this section we describe several end-host multicast protocols we are aware of.

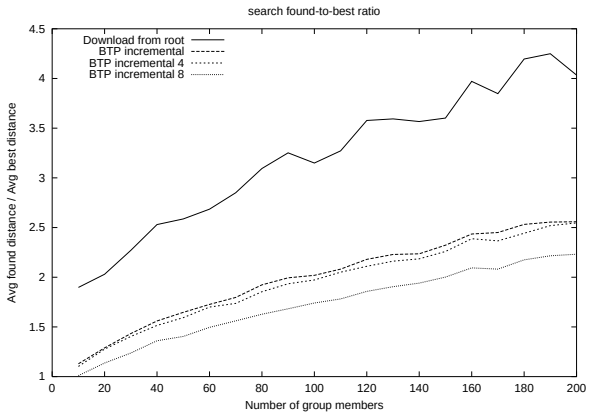

Fig. 7. Ratio of distance to BTP found host to distance to closest host

A. Yoid

Yoid[6] is a feature-rich end-host multicast protocol and intends to support a wide range of applications that require multicast. Yoid builds both a tree and a mesh. The tree is used for normal content transfer and the mesh is used for control data, fault tolerance, and partition detection. The root is not fixed—members of the tree can elect a new root if the root fails or if the network is partitioned. BTP is much simpler than Yoid, but BTP may not support as wide a range of applications as Yoid is intended for.

In Yoid, a node can switch to any other node, but much more work is required to do this than a simple switch in BTP. In the common case, a node will send an intent-to-join (ITJ) message to the prospective parent, who will then forward the message through the tree back to the root. The purpose of this message is to detect loops. If the message reaches the node before it reaches the root, then there must be a loop. Also, intermediate nodes must maintain a graph of pending ITJ's in order to detect a potential loop formation caused by multiple, concurrent switches. A node can also make an emergency switch if its parent goes down. In this case, it immediately switches to another node and then sends a trace message to the root. If the node receives this message, it knows that it is its own ancestor, so there must be a loop.

B. Narada

Narada [7] creates a mesh and then builds a delivery tree from the mesh using an SPF algorithm. Members add and remove edges to optimize the mesh for node-to-node latency, while BTP does this to optimize for cost. Partition detection in Narada relies on timeouts. In BTP, partitions are detected and repaired immediately. Narada uses heuristics to maintain high connectivity, which lowers the chance of a partition when a node fails. Simulations show that Narada creates a tree with reasonable bandwidth, delay, and network stress.

C. AMRoute

AMRoute [8] is intended for ad-hoc wireless networks. Like Narada, it creates a mesh and then uses an SPF algorithm to build the tree. However, it requires the ability to broadcast packets with a bounded TTL (time-to-live) to create the initial mesh, so it would not directly translate to an Internet end-host multicast protocol. AMRoute trees have "logical cores," where new members can rendezvous, and these cores can migrate. Nodes cannot switch parents in AMRoute, instead, the tree will occasionally be reformed from the mesh.

VI. Conclusion

We have designed BTP, an end-host multicast protocol. Our simulations show that while it performs well in ideal situation, it does not perform that well in more realistic scenarios. Further investigation shows that a self-optimizing end-host multicast protocol needs to be able to perform a wide range of graph transformations in order to effectively optimize the graph. We hope to redesign BTP to allow nodes larger degree of freedom in optimization.

References

[1] "Napster homepage," 2000, http://www.napster.com.
[2] "Jungle Monkey homepage," 2000, http://www.junglemonkey.net.
[3] L. Wei and D. Estrin, "The trade-offs of multicast trees and algorithms," 1994.
[4] Sugih Jamin, Cheng Jin, Yixin Jin, Dan Raz, Yuval Shavitt, and Lixia Zhang, ""on the placement of internet instrumentation"," *Proc. of IEEE INFOCOM*, Mar. 2000.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. of ACM SIGCOMM '99*, pp. 251–262, Aug. 1999.

[6] Paul Francis, "Yoid: extending the internet multicast architecture," Tech. Rep., NTT, Apr. 2000.

[7] Yang-hua Chu, Sanjay Rao, and Hui Zhang, "A case for endsystem-only multicast," *Proc. of ACM SIGMETRICS'00*, June 2000.

[8] Mingyan Liu, Rajesh R. Talpade, and Anothony McAuley, "Amroute: adhoc multicast routing protocol," Tech. Rep., ISR, 1999.