An Evaluation of Shared Multicast Trees with Multiple Cores *

Daniel Zappala, Aaron Fabbri, and Virginia Lo
Department of Computer Science, 1202 University of Oregon, Eugene OR 97403-1202
zappala|fabbri|lo@cs.uoregon.edu

Abstract

Native multicast routing protocols have been built and deployed using two basic types of trees: single-source, shortest-path trees and shared, core-based trees. Core-based multicast trees use less routing state compared to shortest-path trees, but generally have higher end-to-end delay and poor fault tolerance.

In this paper we consider a new type of shared multicast structure that uses multiple, independent, simultaneously-active cores. Our design provides for low end-to-end delay, improved fault tolerance, and low source discovery delay, while balancing bandwidth cost and routing state. These results indicate that shared trees with multiple active cores are a viable alternative to shortest-path trees.

The Internet's multicast routing structure is still evolving [1]. Since its inception in 1992, the Multicast Backbone [7] — the multicast-capable subset of the Internet — has primarily consisted of DVMRP [9, 26], PIM [10], and MOSPF [21] routers, tied together with a complex set of interoperability rules and utilizing a flat routing topology. In recent years, network operators have introduced native multicast support, policy, and a hierarchical structure. The current near-term solution consists of domains running PIM-SM [13] internally, connected by MSDP [15] for interdomain reachability. Eventually, MSDP will be replaced by BGMP [20], although some engineers advocate using the single-source architecture proposed by Express [8].

At the heart of all of these multicast routing protocols are two basic types of trees: single-source, shortest-path trees and shared, core-based trees. In each case, a set of senders wants to deliver data to a set of members, known as the multicast group. With shortest-path trees, a separate tree is built for each source, using the least-cost paths between the source and the members. With a shared tree, one tree is built for the entire group and is shared among all the senders. Building an optimal cost shared tree is known as the Minimal Steiner Tree problem and is known to be NP complete [30]. Core-based trees are a simple, low-cost approximation for this problem; a single router is chosen as the core, and a shortest-path tree is built from the core to the members. Senders transmit data toward the core until it reaches the tree.

Shared trees are considered an important part of the multicast routing architecture because only one routing table entry is needed for an entire group, instead of one per source. Hence, BGMP uses shared trees for interdomain multicast to conserve state within the Internet backbone. Shared trees are also useful for rendezvous within a group, since the set of senders may not be known ahead of time. PIM-SM uses a shared tree in this manner, then switches to shortest-path trees once a sender is active.

Despite these advantages, core-based trees have a number of drawbacks relative to shortest-path trees. Foremost among these is that core-based trees on average impose a higher delay between a source and the group members [29]. This is because packets often must travel first to the core and then to the group members, and the core may not be along the shortest path to each member. In addition, the core is a single point of failure; although PIM-SM use a list of backup cores [14], members may experience significant additional delay when a core fails. Finally, using core-based trees may cause traffic concentration, in which some links in the network are much more heavily utilized than others [29, 5].

Surprisingly, very little research has been conducted to study the possibility of using multiple active cores to ameliorate these problems. The designers of both PIM and CBT [2] considered using multiple cores, but chose to use a single active core, with backups available on standby, early in the design stage. OCBT

*This work was supported in part by the National Science Foundation under grants ANI-9977524 and NCR-9714680.

[23] uses a hierarchy of cores, in which cores at lower levels join to their parent in the higher level, forming a star. OCBT's use of multiple cores helps to avoid looping problems that were present in initial designs of CBT. However, because each of the cores cooperate to form a single tree, this structure does not behave any differently than a single-core tree with respect to performance metrics such as delay, routing state, and fault tolerance.

In this paper we demonstrate the promise of building shared multicast trees with multiple, *simultaneously-active*, *independent* cores. Each core is the center of a separate multicast tree, and there is no coordination or dependencies among cores. This design improves the fault tolerance of the shared trees and can significantly improve performance. Our results show that using multiple cores can reduce the average delay experienced by group members, while balancing the bandwidth and routing state used by the tree. We also investigate core placement algorithms and show that placing cores using a dominating set or k-center algorithm can moderately improve delay. Finally, we show that multiple-core trees can generally avoid the problem of traffic concentration, although there are cases where it is a factor. Based on these results, we conclude that shared trees using multiple cores are a significant improvement over single-core trees and thus a viable alternative to shortest-path trees.

The rest of the paper is organized as follows. Section 1 describes related work in the areas of tree comparisons, core placement, and the use of multiple cores. Section 2 describes several alternative designs for using multiple cores and explains how a multicast routing protocol using these cores can be built. Section 3 presents the results of an extensive simulation study examining the performance of these designs, and Section 4 outlines our conclusions.

1 Related work

The tradeoffs of shared trees versus shortest-path trees were initially studied by Wei and Estrin [29]. Their work compares trees constructed using the KMB Steiner Minimal Tree approximation [19] to core-based trees with optimal core placement. Results from their simulation study show that Steiner approximations can have lower cost than shortest-path trees at the expense of higher delay.

Because of the computational cost of computing Steiner approximations, however, the focus of the research community has been on core-based trees. Among the multicast routing protocols that have been deployed, those that use single-core, shared trees include PIM [10], CBT [2], and BGMP [20]. OCBT [23] uses a hierarchy of cores, but no protocol has used multiple independent cores.

With regard to single-core shared trees, the primary focus of the research community has been on the issue of core placement. Wall has shown that trees using optimal core placement have a delay bound twice that of a shortest-path tree [27]. Wei and Estrin [29] studied the average behavior of core-based trees for two forms of optimal placement: optimal among all nodes and optimal among group members. They found that core-based trees are a middle point between the KMB approximation and shortest-path trees, with regard to both cost and delay. A more comprehensive study by Calvert, Zegura, and Donahoo [5] considers random, topology-centered, and group-based placement. They show that group-based placement is most effective, particularly when group members are localized. Both of these studies also consider the problem of traffic concentration, which is an indication of uneven load in the network. Wei and Estrin show that traffic concentration can occur in core-based trees when optimal core placement is used. Calvert, Zegura and Donahoo found that this effect disappears with random core placement.

Selecting an optimal core is difficult, however, if group membership changes over time. Moreover, as group membership changes, the performance of a once-optimal core degrades to that of a random core [24]. Several studies have explored core migration to cope with membership changes; an initial core is selected and then evaluated periodically to determine whether a better choice can be made. Donahoo and Zegura study core migration among a configured set of candidate cores [11]. Each candidate periodically probes a random subset of the group members to collect performance statistics, then determines whether it is the best core. As the cores are evaluated more frequently, the performance of the tree improves, but the overhead of the scheme also increases. Thaler and Ravishankar have also studied several distributed core

migration protocols [24]. The MIN-MEMB protocol starts the core at the first member (or source) to join the group, then the core may periodically migrate to the other group member (or source) with lowest cost. A second protocol they have developed is HILLCLIMB; again, the core starts at the first member, then it may periodically migrate to a neighbor with lower cost. This study shows that HILLCLIMB performs well compared to MIN-MEMB, topology-centered, and random placement, when considering delay and cost. However, the performance of an optimal core is much better. Note that the only deployed core selection mechanism – that used by PIM – is to administratively select a set of candidate cores, and then have each group randomly choose a core from among this set [14].

Finding an optimal placement for multiple cores is of course much more difficult than for a single core. One way of looking at the multiple-core placement problem is to be given d, the maximum distance from a node to a core, and try to find the smallest number of cores that satisfies this criteria. This problem is known as the minimum d-dominating set problem, and has been shown to be NP-hard [16]. Another way of looking at this problem is to be given k, the number of cores desired, and try to place these cores such that the maximum distance from a node to its nearest core is minimized. This problem is known as the minimum k-center problem and has also been shown to be NP-hard [16]. Vazirani describes an $O(N|E|)$ algorithm which gives a solution with no worse than twice the optimal maximum delay [25]. Jamin et al. have used this algorithm, as well as random placement, in order to place Internet instrumentation. Handler and Marchandi discuss several other heuristics for approximating center placement when average distance is being minimized, notably node partitioning, greedy, node substitution, and branch-and-bound heuristics [18].

2 Multiple-core designs

We are interested in multiple-core, shared trees because they may offer superior performance with respect to single-core trees. We consider two possible designs for multiple-core trees and their implications for designing a multicast routing protocol. Both of these basic designs result in at most one copy of each packet being delivered to the group members.

2.1 Alternative designs

Our multiple-core designs share some basic functionality, namely each core is the root of its own bidirectional, shared multicast tree, spanning some or all of the group members. Senders that are not group members transmit packets toward the core until they reach a router that is part of the bidirectional tree. Using this as a foundation, we explore these two designs:

1. **Senders-To-All.** Each sender transmits data to all the cores; members join to only one of the cores. Fig. 1 illustrates how the Senders-To-All protocol works. In this example, there are three cores, each of which is the center of a separate, bidirectional tree connecting a subset of the group members. A sender, marked by S, is also a member and has joined to core 1. When it transmits a packet, it sends 3 copies, one toward each core, until they reach some router on the tree for that core. To receive packets, a member chooses a core and joins that core's shared tree.

2. **Members-To-All.** Each sender transmits data to just one of the cores; members join all of the cores. Fig. 2 illustrates how the Members-To-All protocol works. As with the previous example, there are three cores, but this time all members have joined to all the cores. In effect, this creates a set of redundant trees, and the sender chooses only one of them to transmit its data. In this example, sender S is also a member so it is joined to all of the trees. When it transmits a packet, it can send it on any one of the three trees, and can in fact choose a different tree for each packet. Likewise, different senders can utilize different cores.

Both of these designs improve fault tolerance, compared to a single-core protocol such as PIM, by reducing the recovery delay when a core fails. With multiple-core trees, a sender or receiver (depending on

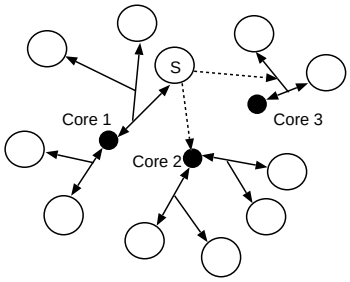

Figure 1: Senders-To-All

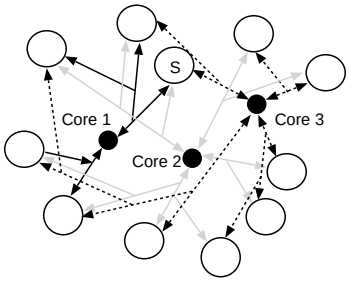

Figure 2: Members-To-All

the variation) can simply choose another core, and the tree for this core is already installed. With PIM, group members switching to a backup core have to wait for the multicast routing protocol to remove state for the old core and re-establish the tree for the new one. Moreover, multiple-core trees localize the recovery delay to only those senders or members who are using the failed core.

If one wants to further increase fault tolerance by delivering duplicate packets, one could add the case where where both the senders and the members use all the cores. However, we do not expect this case to perform significantly different from the other cases above, so our performance evaluation does not cover this case.

We are also investigating a design in which the senders and members both use only one core, and the cores distribute multicast packets among them. Distribution among cores could be done using a spanning tree, a ring, or some other structure. We do not consider this design in this paper; using a core distribution mechanism adds complexity and additional fault tolerance concerns, which are best considered separately.

2.2 Senders-To-All advantages

The Senders-To-All design has several significant advantages when compared to Members-To-All.

First, Senders-To-All will on average use less routing state, which we define as the number of routing entries per group at a given node. For both designs there is one tree per core, but for Members-To-All, each tree connects all of the members. Thus, for Members-To-All each router is likely to have one entry per core for each group, particularly as the group size grows. For Senders-To-All, on the other hand, each router is likely to have only one entry per group, especially if nearest attachment is used.

Second, the Senders-To-All design has the advantage of giving group members control over choosing a core. With Members-To-All, the source is responsible for choosing a core, then monitoring its status so that it can switch to a new core in case of failure. With Senders-To-All members have greater flexibility, since a given core may be good for some members and bad for others, depending on its location and the status of the network. With Senders-To-All, members can react quickly to failures and can even switch cores in order to improve performance characteristics such as loss and delay.

2.3 Core discovery and selection

In order to utilize multiple cores, senders and receivers (or their first-hop routers) need mechanisms to discover the identities of cores and select the one they will use. We refer to the first of these issues as *core discovery* and the second as *core selection*. A separate issue, namely *core placement*, is discussed in our performance evaluation.

For single-core protocols such as PIM, the current core discovery method [14] relies on distributing a set of candidate cores throughout the network. When a sender or receiver needs to send to or join a group, it selects one of these candidate cores to act as the core for the group. All of the sources and group members deterministically select the same core because PIM uses a hash function based on the group identifier.

Our multiple-core designs can use this same mechanism to distribute candidate cores and to choose a different *set* of active cores for each group. The PIM hash function produces a different ordering of cores

for each group and PIM uses this ordering to select a backup core should the primary core fail. Similarly, a multiple-core protocol can use the hash function to select a set of cores, along with backups for each of them.

Once the set of cores is known, the group members must decide which core to utilize. For the Senders-To-All design, members must decide which core to receive packets from. Likewise, for the Members-To-All design, senders must decide which core to send packets to. The results of our performance evaluation indicate that choosing cores that are near the sender or receiver can significantly improve performance over random selection.

There are a variety of web server selection techniques [12, 17, 6] that are relevant to the core selection problem. Various metrics have been used in this area, including latency, the number of hops, geographic positioning, available bandwidth, and server load. Another technique, called anycasting [22, 3], uses a single IP address for all servers and implements a service that delivers a request from a client to the nearest server.

However, core selection for multicast presents a more challenging problem than web server selection because a receiver is interested in end-to-end performance, from senders to receivers. The core is only an intermediary point, so the performance of the path from the core to the receivers may not be a good enough indicator of end-to-end characteristics. What may be required is for the client to use a mechanism to choose a subset of the cores (perhaps those that are closest) and then dynamically switch between these cores as required.

Certainly, developing an appropriate core selection mechanism for multiple-core trees remains an area for future study.

2.4 Packet forwarding and reception

To implement a multicast routing protocol that uses multiple cores, two additional issues must be addressed: packet forwarding and packet reception. The current multicast routing architecture allows for only one shared tree per group. However, in our designs, there are several or many shared trees per group (one per core). These trees may overlap and thus must be identified and built individually.

To facilitate our discussion we must first explain the current types of routing entries used for multicast. Currently, a multicast routing entry may be designated using either (S, G) or $(*, G)$ where S is the source address of a multicast packet and G is the group address. If a multicast packet matches a (S, G) entry, routers assume a shortest-path tree is being used. Otherwise, if a multicast packet matches a $(*, G)$ entry, routers assume a shared tree is being used and the packet is forwarded accordingly (either on a unidirectional or bidirectional shared tree).

For multiple-core trees to work, we need a separate (S, G) entry for each core, but we need packets matching these entries to be forwarded on a bidirectional tree, not a shortest-path tree.

Thus, to accommodate multiple-core trees, we introduce the notion of a generic multicast routing entry. As illustrated in Fig. 3, a routing entry is identified by a *source* and *destination* address. These fields correspond to the IP source and destination fields (for multicast the destination is the group address). Packets that match the source and destination address are forwarded according to the *type* of the entry. For packets on a unidirectional tree, a router checks that the *incoming interface* matches the interface the packet arrived on, and if so it forwards it on all the *outgoing interfaces*. For a bidirectional tree, a router forwards packets on all outgoing interfaces except the one the packet arrived on.

Fig. 3 shows how entries for a shortest-path tree (PIM, DVMRP), a unidirectional shared tree (PIM) and a bidirectional shared tree (CBT, BGMP) would use this generic format.

Fig. 3 also shows an example of a routing entry for a multiple-core bidirectional tree. The tree is identified by the combination of (S, G), where S is the address of the core. Unlike a shortest-path tree entry, however, packets are forwarded according to the rules for a bidirectional tree. This allows any router on the shared tree to encapsulate a packet using the source address of the core and deliver it along the shared tree.

Fig. 4 illustrates how this forwarding process works. First, when a source sends a packet to the group, its first-hop router encapsulates it and unicasts the packet toward its nearest core. Second, when it reaches a router on the core's bidirectional tree, this router removes the original packet and then re-encapsulates it,

Generic Multicast Routing Entry

Source	= {IPAddress | *}
Destination	= {IPAddress}
Type	= {Unidirectional | Bidirectional}
Incoming Interface	= {InterfaceID}
Outgoing Interfaces	= {[InterfaceID1, InterfaceID2, ...]}

Shortest-Path Tree Entry

Source	= {S}
Destination	= {G}
Type	= {Unidirectional}
Incoming Interface	= {InterfaceID}
Outgoing Interfaces	= {[InterfaceID1, InterfaceID2, ...]}

Unidirectional Shared Tree Entry

Source	= {*}
Destination	= {G}
Type	= {Unidirectional}
Incoming Interface	= {InterfaceID}
Outgoing Interfaces	= {[InterfaceID1, InterfaceID2, ...]}

Bidirectional Shared Tree Entry

Source	= {*}
Destination	= {G}
Type	= {Bidirectional}
Incoming Interface	= {}
Outgoing Interfaces	= {[InterfaceID1, InterfaceID2, ...]}

Multiple-Core Bidiirectional Tree Entry

Source	= {Core}
Destination	= {G}
Type	= {Bidirectional}
Incoming Interface	= {}
Outgoing Interfaces	= {[InterfaceID1, InterfaceID2, ...]}

Figure 3: Generic multicast routing entry and examples

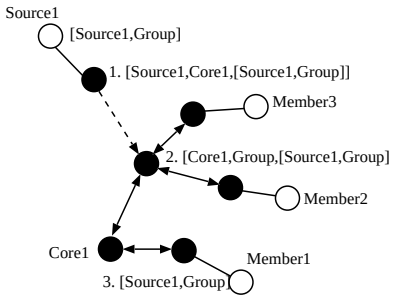

Figure 4: Using a multiple-core bidirectional tree

this time using the core's address as the source address. This packet is multicast along the bidirectional tree until it reaches the leaf routers. Finally, these routers remove the original packet and deliver it to their local members. Note that this process requires packets to be encapsulated twice. If a sender is also a member, then only one encapsulation is needed because the unicasting step is eliminated.

The steps for receiving packets depend on whether the Senders-To-All or Members-To-All design is being used. For Senders-To-All, a member's first hop router chooses one core and joins the bidirectional tree for that core. For Senders-To-All, the first-hop router joins the shared trees for all of the cores. These trees are joined in the usual manner, that is, by sending a separate join message toward the core for each tree.

3 Performance evaluation

We evaluate the performance of multiple-core trees by comparing them to single-core trees. As has been done in several previous studies comparing tree types [29] and core placement algorithms [5], we use shortest-path trees as the baseline for comparison across different graphs and groups. In this section we describe our experiment setup and results.

3.1 Experiment setup

We use a static model, in which we generate a random graph, choose a set of nodes to be group members, then compute shortest-path trees (SPT) and multiple-core trees (MCT) using the Senders-To-All and Members-To-All designs. The metrics we evaluate include end-to-end delay, bandwidth cost, routing state, and traffic concentration. Most experiments are run until the width of the 95% confidence interval for all metrics is within 10% of the mean value.

The factors in our experiment include:

- *Graphs.* For most experiments, we use a set of 10 flat, random graphs of 100 nodes each, using the

Waxman model [28] within the GT-ITM topology generator [4]. All edges have unit weights and the average node degree is near 4. For some of our traffic concentration experiments we use a set of 8 graphs of 50 nodes each, constructed so that the average node degree ranges from 1 to 8.

- *Groups.* We generate group members randomly, with all members also functioning as senders. Group sizes vary from 5 to 50.

- *Number of Cores.* We vary the number of cores from 1 to 8 in order to assess the benefits and costs of adding additional cores.

- *Core Placement.* For most experiments, we generate a separate set of random cores for each group. Several studies have already compared core placement strategies [29, 5, 24], so their performance relative to random placement is well known. In addition, Thaler has shown that optimal core placement degrades to random core placement over time for a group with dynamic group membership [24]. Thus, random placement is useful as a benchmark, particularly since we can use this same placement strategy for both single-core and multiple-core trees.

  We are also interested in assessing how important core placement is for multiple-core protocols. Thus we conduct some some experiments using a dominating set approximation and others using a k-center approximation to select a single set of cores that are used for all groups.

- *Core Selection.* For multiple-core trees, group members must choose which core to join (for Senders-To-All) and senders must choose which core to send packets to (for Members-To-All). We use both random and nearest selection, where the nearest core is the one that is closest in terms of path length.

To conduct the experiments, we wrote a simulator called *treecalc*, which has a C++ core and a Tcl interface.

3.2 Experiment results: delay

Our first set of experiments measures *end-to-end delay* in terms of the number of links traversed between a sender and a receiver. We calculate the maximum delay for each sender, then average these numbers to find the average-maximum delay for the group. Results are nearly identical for the average-average delay. These experiments use random core placement and vary the number of cores and the size of the multicast group.

We find that both Senders-To-All and Members-To-All can reduce the delay experienced by group members when nearest selection is used. As shown in Fig. 5 and Fig. 6, the delay ratio decreases by about 10% for Senders-To-All and 20% for Members-To-All as as the number of cores increases. For both designs, as cores are added, there is a greater chance that the member (or sender) will choose a core that is close to itself and to the shortest paths it will use. Members-To-All fares better because the core is selected by the sender; the more cores there are, the greater the chance that the delay for the sender will be the same as that for a shortest-path tree. Senders-To-All can achieve equivalent performance for delay, but will require more cores.

Our results also show that both designs are somewhat tolerant of members choosing distant cores. For Senders-To-All delay increases slightly (Fig. 7), while for Members-To-All delay actually decreases slightly (Fig. 8). This is a significant result because it means that performance will not suffer if a sender or member must switch to a different core when its nearest one fails.

3.3 Experiment results: cost

We use the same experiments to evaluate *bandwidth cost*, which we define as the number of links in a tree. This measure roughly represents the bandwidth consumed by one packet transmission. We calculate cost separately for each sender, then average these costs together to find the average cost for the whole group. Note that for Senders-To-All, we count each link for each packet sent; thus if three packets are sent to three different cores, it may be possible for one link to be counted three times.

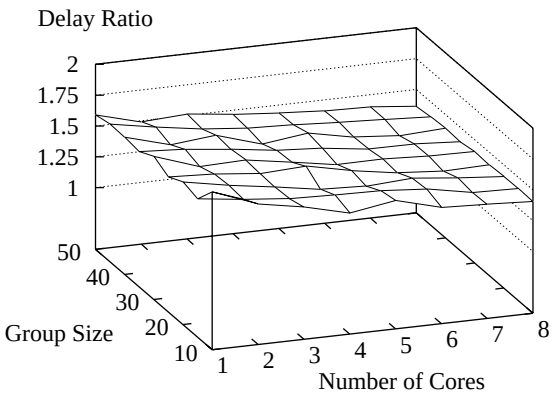

Figure 5: Delay: Senders-To-All, random core placement, nearest core selection

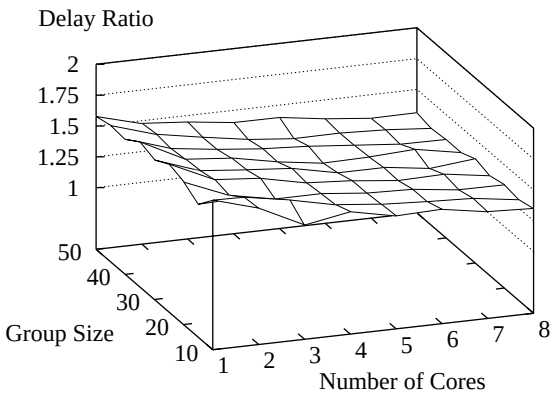

Figure 6: Delay: Members-To-All, random core placement, nearest core selection

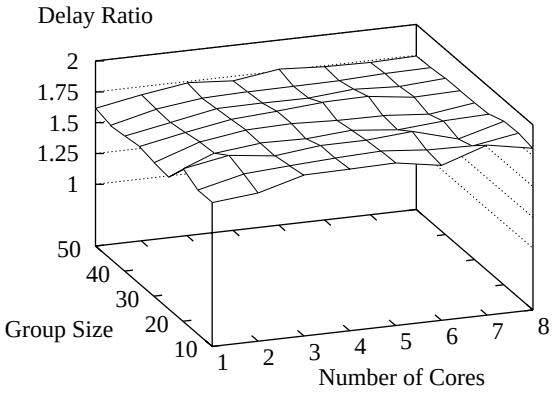

Figure 7: Delay: Senders-To-All, random core placement, random core selection

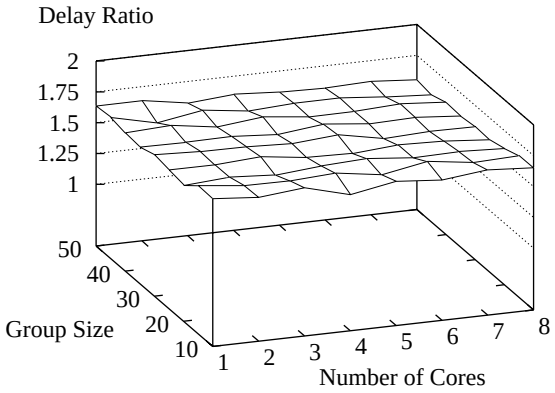

Figure 8: Delay: Members-To-All, random core placement, random core selection

We find that the Members-To-All design performs better than Senders-To-All with respect to cost. When either nearest or random core selection is used, Members-To-All consumes bandwidth nearly equivalent to a shortest-path tree (Fig. 10 and Fig. 12).

Senders-To-All also performs well with regard to cost as long as nearest selection is used (Fig. 9). The only exception is that the cost increases for small groups with many cores. This is due to senders transmitting packets to cores that do not have any attached members. This suggests that a mechanism is needed where the number of cores can be adjusted dynamically with group size.

However, for random core selection, Senders-To-All can waste a lot of bandwidth (Fig. 11). In this case, the cost ratio nearly doubles as the number of cores increases. This happens because the sender generates a separate packet for each core and transmits each copy on a separate, but possibly overlapping tree. The copies will often travel some of the same links, multiplying the amount of bandwidth consumed by the sender. This suggests that members should choose nearby cores if their nearest one fails.

3.4 Experiment results: routing state

Our second set of experiments measures *routing state*, which we define as the number of multicast routing entries stored at a router per group. For these experiments, we use the same 100-node networks, and load

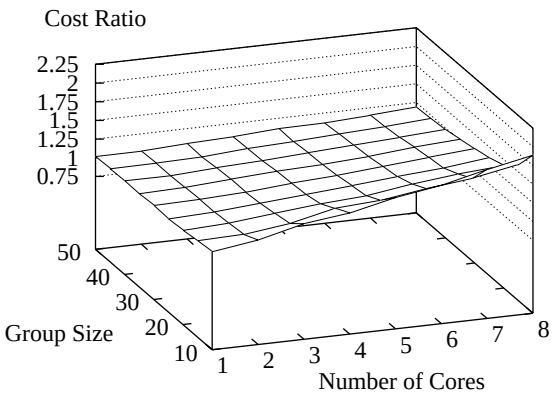

Figure 9: Cost: Senders-To-All, random core placement, nearest core selection

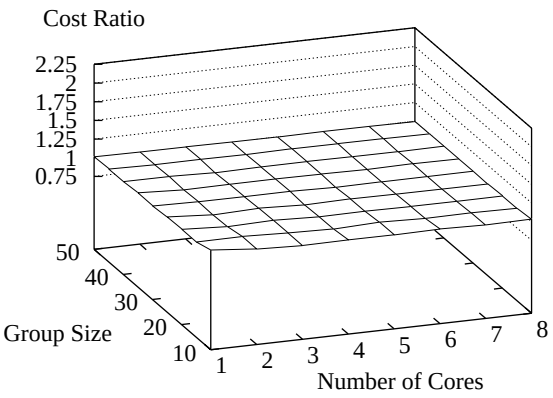

Figure 10: Cost: Members-To-All, random core placement, nearest core selection

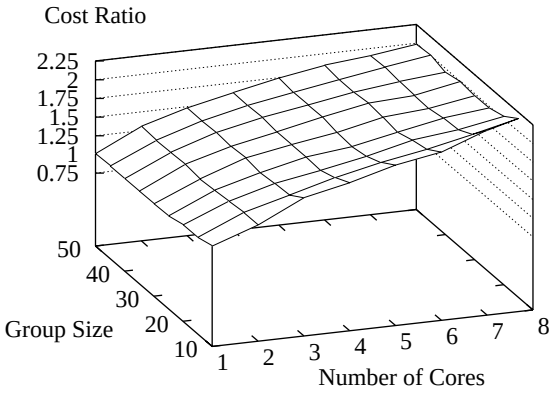

Figure 11: Cost: Senders-To-All, random core placement, random core selection

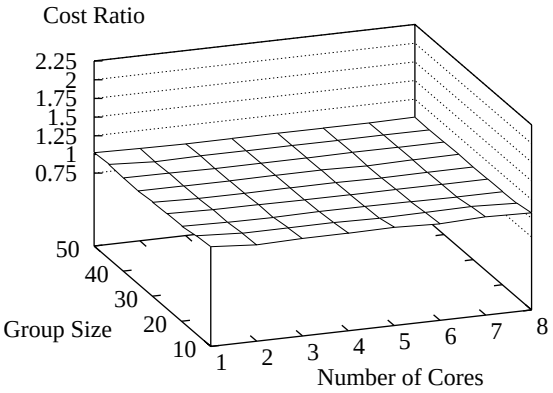

Figure 12: Cost: Members-To-All, random core placement, random core selection

each network with 30 groups. Each group uses random core placement, and we vary the number of cores and the size of the groups. For each network, we calculate the average amount of routing state over all nodes and run the experiments until the width of the 95% confidence interval for all metrics is within 10% of the mean value.

Our results show that using the Senders-To-All variant with nearest selection uses the same amount of routing state as a tree with a single core, regardless of the number of cores used (Fig. 13). This occurs because members choose a core near them, and the trees for the different cores do not overlap. For Members-To-All, the amount of state increases as cores are added and as the group size grows (Fig. 14). Here, the trees for each of the cores overlaps so the amount of state is proportional to the number of cores. Using random core selection does not affect the amount of routing state used by Members-To-All; it only slightly increases the amount of state used by Senders-To-All

In all cases, the amount of state is far less than that of shortest-path trees, where the state is proportional to the number of senders. For a group size of 50, for example, the average number of routing entries with shortest-path trees is near 35.

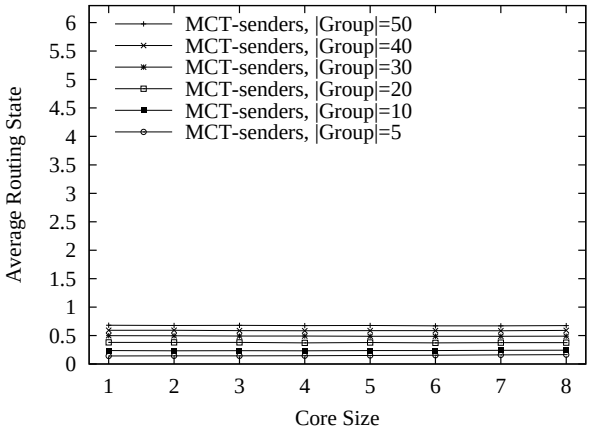

Figure 13: Routing state: Senders-To-All, random core placement, nearest core selection

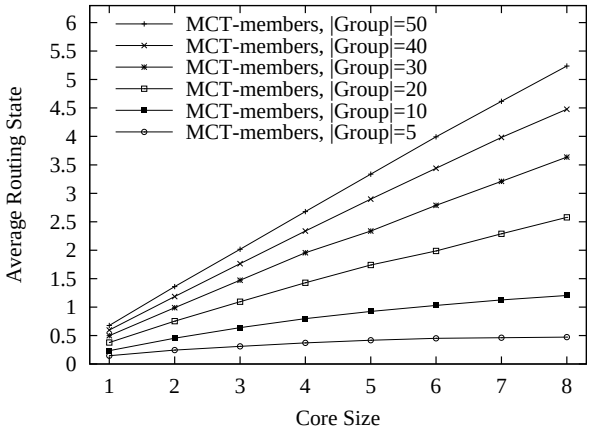

Figure 14: Routing state: Members-To-All, random core placement, nearest core selection

3.5 Experiment results: core placement

To examine the effects of core placement, we repeated the delay and cost experiments using two approximation algorithms to place the cores. Our *dominating set* approximation algorithm finds a subset of the nodes in the graph such that all nodes are within d hops of this set. Our *k-center* approximation algorithm algorithm fixes the number of cores to be equal to k, then tries to place them so as to minimize the distance from all nodes to the cores. Both finding a minimal dominating set and choosing an optimal placement of centers is NP-hard. We use approximation algorithms based on node degree, and our experiments indicate these are good approximations in random graphs.

3.5.1 Dominating set core placement

Our dominating set algorithm takes as input a parameter d, then chooses a subset of the nodes in the graph such that all nodes are within d hops of this dominating set. During the algorithm, we store two sets of nodes: the *dominating set* and the *covered set*. A node that is added to the dominating set covers itself and all neighbors within d hops. At each step in the algorithm, a node keeps track of its *covering degree*, which is its number of neighbors that are not yet covered. Initially both the dominating set and the covering set are empty.

Our algorithm works as follows. First, the nodes are sorted into a list in decreasing order of their covering degree. The node with the largest covering degree removes itself from the list and adds itself to the dominating set. At the same time, this node adds itself and all nodes within d hops to the covered set. The covering degrees of all nodes are recomputed according to the new covered set, and the list of nodes is sorted again in order of largest covering degree. This ends the first iteration of the algorithm, and it is repeated until all nodes are in the covered set.

To determine how well this placement algorithm performs, we compare it to random placement; doing this is a little difficult. The approximation algorithm calculates one dominating set for each graph, so all groups on the same graph use the same set of cores. In addition, for a given d, each graph may have a different size d-dominating set. What we have done is to calculate the 2- and 3-dominating set for each of the graphs, then find the average size dominating set for each case. We then compare random placement to dominating set placement using the same average number of cores. We use only nearest core selection for these experiments.

Using this methodology, we find that dominating set placement may slightly improve end-to-end delay for multiple-core trees. Fig. 15 shows the delay ratio for Senders-To-All and Members-To-All using both random placement and placement with a 3-dominating set. For this case, the average 3-dominating set size is 6, so we include random placement of 6 cores in this graph. Senders-To-All improves by at most 10%,

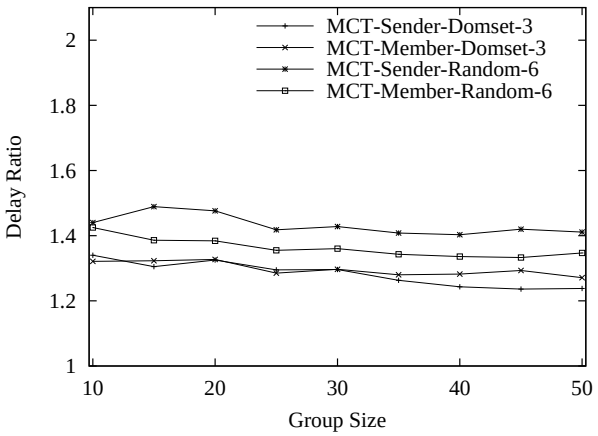

Figure 15: Delay: 3-dominating set placement and nearest selection

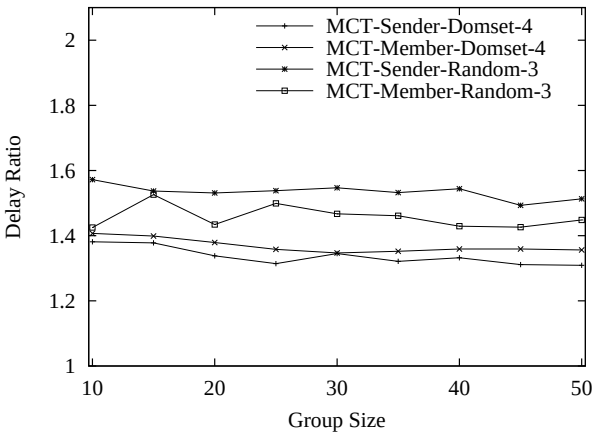

Figure 16: Delay: 4-dominating set placement and nearest selection

while Members-To-All shows only a marginal improvement. Similar results are obtained for a 4-dominating set; see Fig. 16.

3.5.2 K-center core placement

While the dominating set algorithm computes a small set of cores such that all nodes are within a given number of hops, a network administrator may prefer to specify the number of cores desired and then place them in the most efficient manner. For this situation, we evaluate an approximation to the k-center problem. The algorithm we use is identical to that used for approximating a d-dominating set, with the modification that we continue choosing nodes until k are selected. The nodes are still ordered according to covering degree as they are selected.

To determine how well k-center placement performs, we compare it to random core placement with the same number of cores and to optimal core placement for a single core. For optimal core placement we use the core that has the lowest average-maximum delay to all group members. For these experiments we keep the group size constant at either 10 or 50 members, and vary the number of cores. We use nearest selection for the multiple-core trees.

Our results show that multiple-core trees using k-center placement can moderately improve the end-to-end delay performance for multiple-core trees. Fig. 17 and Fig. 18 shows results for k-center placement using d equal to 2 for groups of size 10 and 50 respectively. As the number of cores increases, multiple core trees using k-center placement can have delay equivalent to a single-core tree with optimal core placement. For both group sizes this occurs when the number of cores is 4.

3.6 Experiment results: traffic concentration

Our fourth set of experiments measures *traffic concentration*, which is an indication of overload in a network. To determine whether traffic concentration is occurring, we calculate the number of flows traversing each link, where a flow is a packet stream sent from a source to a group. With multiple-core trees, it is possible that a given link is traversed more than once by a single flow, due to encapsulation, and these occurrences are counted. Traffic concentration occurs when some links have a very large number of flows and other links are under-utilized. For these experiments, we use flat, random, 50-node graphs of varying degrees, and load each network with 100 groups of 30 members.

Our results show that multiple-core trees do not suffer from traffic concentration as long as random core placement is used. With random core placement, each group uses a different set of cores, so the traffic is evenly spread throughout the network. When k-center placement is used, a small set of cores is used for all groups. In this case, increasing the number of cores decreases traffic concentration for Members-To-All but increases it for Senders-To-All.

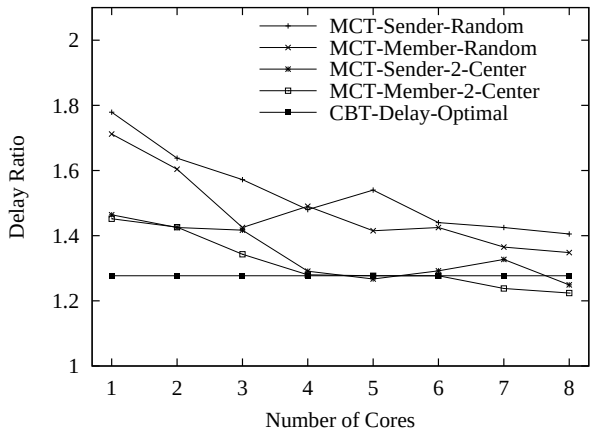

Figure 17: Delay: k-center placement, nearest selection, group size 10

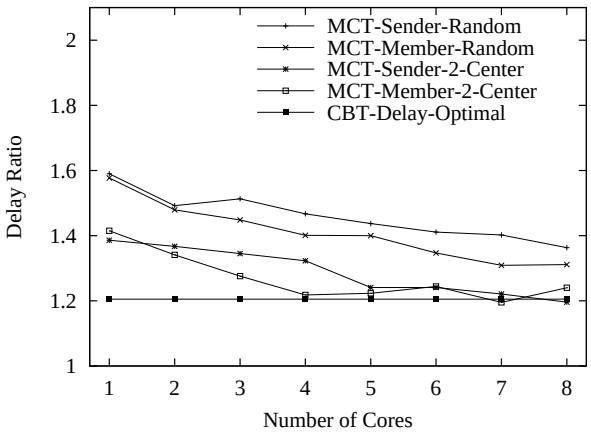

Figure 18: Cost: k-center placement, nearest selection, group size 50

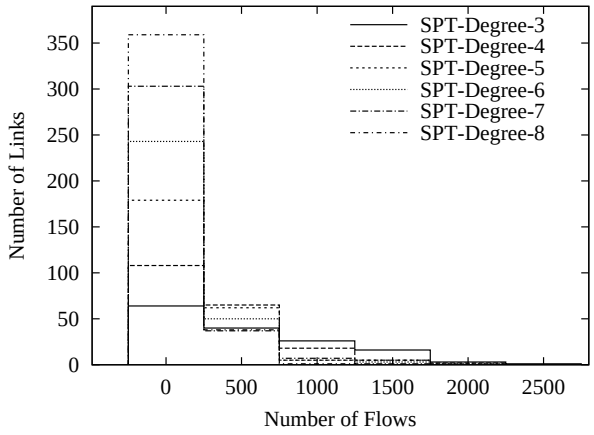

Figure 19: Traffic distribution: SPT

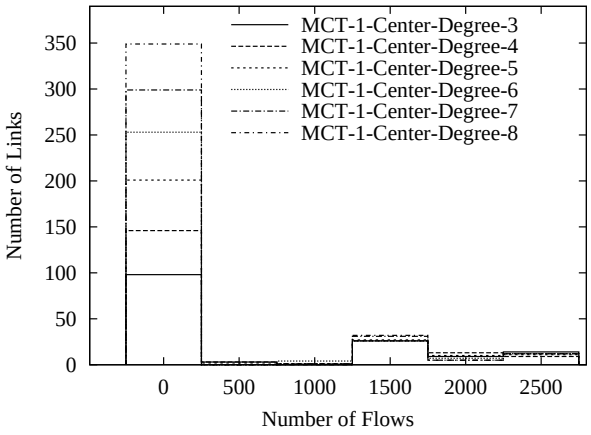

Figure 20: Traffic distribution: Senders-To-All, k-center placement, 1 core

To illustrate these results, we first use Fig. 19 to show the lack of traffic concentration when shortest-path trees are used. This figure shows the distribution of the number of flows per link, with separate histograms for networks of various average node degree. For a network of degree 3, the distribution has a long tail, indicating some links are very heavily loaded. As the degree of the network increases, shortest-path trees are able to use the extra links to balance traffic in the network.

Comparing this distribution to Fig. 20 we can see that traffic concentration does occur when a small number of cores is used for all groups. This figure shows the distribution of the number of flows per link for Senders-To-All using k-center placement of just 1 core, with d equal to 2. Regardless of the degree of the network, the distribution has a long tail, indicating there are always some links that are very heavily loaded.

In order to determine the range of situations over which traffic concentration occurs, we measure the maximum number of flows on a link for multiple-core trees over networks with varying degrees. Fig. 21 shows the maximum number of flows as a function of node degree for Members-To-All. As the number of cores grows from 1 to 8, Members-To-All is able to take advantage of the number of paths available in denser networks. Contrasting this with Fig. 22, we see that adding cores actually increases traffic concentration when the network degree is low.

Most of these observations confirm the results of Calvert, Zegura, and Donahoo [5] in their study of traffic concentration with single-core trees. In that study, the authors find that traffic concentration occurs only when a small number of cores is used for all groups. Our results show that this rule does not not hold universally for multiple-core trees. Rather, the Members-To-All variant can actually reduce the effects of

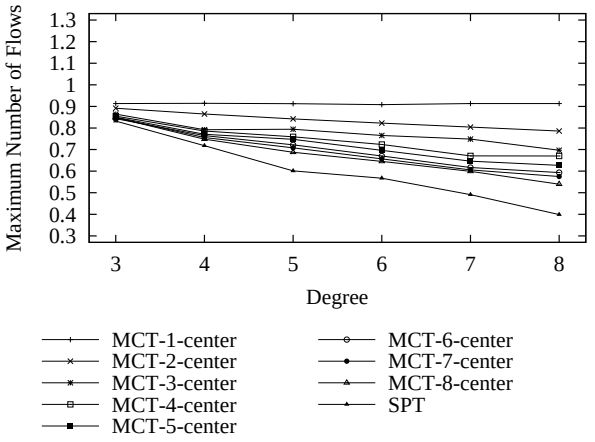

Figure 21: Maximum traffic: Members-To-All, k-center placement

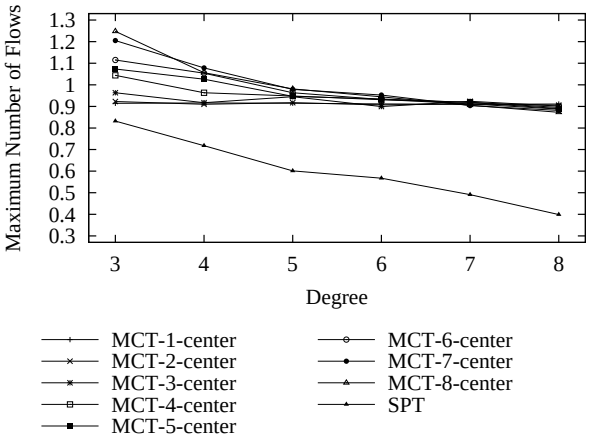

Figure 22: Maximum traffic: Senders-To-All, k-center placement

traffic concentration as cores are added.

4 Conclusions and future work

Having examined the performance of multiple-core trees, we can conclude that they are a feasible alternative to shortest-path trees. Due to their design, multiple-core trees have low source discovery delay and good fault tolerance. By adding additional cores, we can reduce end-to-end delay relative to having a single core.

The two multiple-core designs we consider tradeoff bandwidth cost for routing state. Senders-To-All uses very little routing state, but can have a high bandwidth cost due to its reliance on unicast to deliver packets from senders to cores. Members-To-All reduces bandwidth cost, but increases the amount of routing state since all group members join all of the cores' trees.

We also have studied core placement strategies and found that k-center placement can moderately reduce delay. However, this placement strategy can also lead to traffic concentration for the Senders-To-All variant. As long as random core placement is used, both multiple-core designs balance traffic as well as shortest-path trees.

Due to the difficulty of deploying native multicast protocols, our current efforts are focusing on application-layer multicast. Using the foundation of our multiple-core designs, we have designed a proxy-based protocol that runs on a Single-Source Multicast infrastructure [31]. SSM proxies can provide many of the same benefits of multicast cores, with the added flexibility that is afforded by operating at the application layer. This flexibility leads to more efficient multicast delivery, and allows us to design additional mechanisms for dynamic proxy configuration and access control.

Acknowledgments

The authors would like to thank Krishnan Sivaramakrishna Iyer for help developing the dominating set approximation algorithm and Ragini Singh for help verifying the simulator.

References

[1] K. Almaroth. The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment. *IEEE Network*, January 2000.

[2] A. J. Ballardie, P. F. Francis, and J. Crowcroft. Core Based Trees. In *ACM SIGCOMM*, August 1993.

[3] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei. Application-Layer Anycasting. In *IEEE Infocom*, 1997.

[4] Ken Calvert and Ellen Zegura. Georgia Tech Internetwork Topology Models. Software at http://www.cc.gatech.edu.

[5] K.L. Calvert, E.W. Zegura, and M.J. Donahoo. Core Selection Methods for Multicast Routing. In *International Conference on Computer Communications Networks*, September 1995.

[6] Robert L. Carter and Mark E. Crovella. Dynamic Server Selection using Bandwidth Probing in Wide Area Networks. In *IEEE INFOCOM*, April 1997.

[7] S. Casner and S. Deering. First IETF Internet Audiocast. *ACM SIGCOMM*, 22(3), July 1992.

[8] D.R. Cheriton and H.W. Holbrook. EXPRESS Multicast: Making Multicast Economically Viable. In *ACM SIGCOMM*, August 1999.

[9] Stephen Deering. Multicast Routing in Internetworks and Extended LANs. In *ACM SIGCOMM*, August 1988.

[10] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. An Architecture for Wide-Area Multicast Routing. In *ACM SIGCOMM*, August 1994.

[11] M. Jeff Donahoo and Ellen W. Zegura. Core Migration for Dynamic Multicast Routing. In *International Conference on Computer Communication Networks*, 1996.

[12] S. G. Dykes, C. L. Jeffery, and K. A. Robbins. An Empirical Evaluation of Client-Side Server Selection Algorithms. In *IEEE INFOCOM*, 2000.

[13] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification. RFC 2117, June 1997.

[14] Deborah Estrin, Mark Handley, Ahmed Helmy, and Polly Huang. A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing. In *IEEE INFOCOM*, 1999.

[15] Dino Farinacci, Yakov Rekhter, David Meyer, Peter Lothberg, Hank Kilmer, and Jeremy Hall. Multicast Source Discovery Protocol (MSDP). Internet Draft: work in progress, July 2000.

[16] Michael R. Garey and David S. Johnson. *Computers and Instractability*. W. H. Freeman and Co., 1999.

[17] A.J.D. Guyton and M.F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *ACM SIGCOMM*, 1995.

[18] Gabriel Y. Handler and Pitu B. Mirchandani. *Location on Networks: Theory and Algorithms*. The MIT Press, 1979.

[19] L. Kou, G. Markowsky, and L. Berman. A Fast Algorithm for Steiner Trees. *Acta Informatica*, 15:141–145, 1981.

[20] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D.Estrin, and M. Handley. The MASC/BGMP Architecture for Inter-domain Multicast Routing. In *ACM SIGCOMM*, August 1998.

[21] J. Moy. Multicast Extensions to OSPF. RFC 1584, March 1994.

[22] C. Patridge, T. Mendez, and W. Milliken. Host Anycasting Service. RFC 1546, November 1993.

[23] C. Shields and J.J. Garcia-Luna-Aceves. The Ordered Core Based Tree Protocol. In *IEEE INFOCOM*, 1997.

[24] David G. Thaler and Chinya V. Ravishankar. Distributed Center-Location Algorithms: Proposals and Comparisons. In *IEEE INFOCOM*, 1996. Also published in *IEEE Journal on Selected Areas in Communications*, April 1997.

[25] Vijay Vazirani. *Approximation Methods*. Springer-Verlag, 2000.

[26] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. RFC 1075, November 1988.

[27] David W. Wall. *Mechanisms for Broadcast and Selective Broadcast*. PhD thesis, Department of Electrical Engineering, Stanford University, 1980.

[28] Bernard M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9), December 1988.

[29] Liming Wei and Deborah Estrin. The Trade-offs of Multicast Trees and Algorithms. In *1994 International Conference on Computer Communications Networks*, September 1994.

[30] Pawel Winter. Steiner Problem in Networks: A Survey. *IEEE Networks*, 17(2):129–167, 1987.

[31] Daniel Zappala and Aaron Fabbri. Using SSM Proxies to Provide Efficient Multiple-Source Multicast Delivery. In *Sixth Global Internet Symposium, Globecom 2001*, November 2001.