An Evaluation of Shared Multicast Trees with Multiple Active Cores

Daniel Zappala and Aaron Fabbri [⋆]

Department of Computer Science, University of Oregon, Eugene, OR 97403-1202
[zappala|fabbri]@cs.uoregon.edu

Abstract. Core-based multicast trees use less router state, but have significant drawbacks when compared to shortest-path trees, namely higher delay and poor fault tolerance. We evaluate the feasibility of using multiple independent cores within a shared multicast tree. We consider several basic designs and discuss how using multiple cores improves fault tolerance without sacrificing router state. We examine the performance of multiple-core trees with respect to single-core trees and find that adding cores significantly lowers delay without increasing cost. Moreover, it takes only a small number of cores, placed with a k-center approximation, for a multiple-core tree to have lower delay than a single-core tree with optimal core placement. We also find that traffic concentration is avoided as long as the load is spread among a set of cores. These results indicate that shared trees with multiple active cores are a viable alternative to shortest-path trees.

1 Introduction

Multicast routing protocols are built using 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. Core-based trees are a simple way to build shared trees; 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 have a significant advantage over single-source trees in that only only one routing table entry is needed for an entire group, instead of one per source. Hence, BGMP [1] uses shared trees for interdomain multicast to conserve state within the Internet backbone.

Despite this advantage, 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 [2]. This is because packets often must travel first to the core and then to the group members,

[⋆] This work was supported in part by the National Science Foundation under Grant ANI-9977524 and NCR-9714680.

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 [3] uses a list of backup cores [4], 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 [2, 5].

Surprisingly, very little research has been conducted to study the possibility of using multiple cores to ameliorate these problems. The designers of both PIM and CBT [6] considered using multiple cores, but chose to use a single core early in the design stage. OCBT [7] uses a hierarchy of cores, in which cores at lower levels join to their parent in the higher level, forming a tree. 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 delay, fault tolerance, or traffic concentration.

In this paper we demonstrate the promise of building shared multicast trees with multiple, *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 decreases the delay experienced by group members, since there is a greater possibility that each member will have a core near its shortest path. Furthermore, we achieve the surprising result that a small set of cores placed with a k-center approximation can produce a tree with lower delay than a single-core tree with optimal core placement. Finally, we show that multiple cores can spread the load of multicast traffic, eliminating the problem of traffic concentration. 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.

We begin by examining two alternatives for multiple-core trees and describe how these protocols can be implemented. Then we present the results of an extensive simulation study examining the performance of these designs with respect to delay, cost, and traffic concentration.

2 Multiple-Core Designs

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:

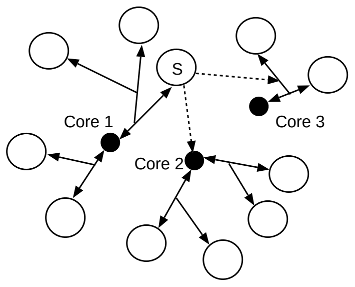


Fig. 1. Senders-To-All

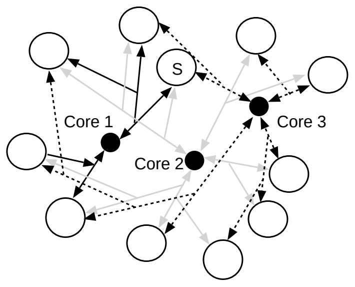


Fig. 2. Members-To-All

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 n 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.

We are also investigating a third 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.

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 router 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.

The Senders-To-All design also 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.

It is also worth noting that, compared to a protocol such as PIM, both of our designs reduce the delay incurred when a core fails. Since the cores are already active (either receiving data or having members joined), the time required to switch cores after a failure is reduced. Moreover, both designs localize the recovery delay to only those senders or members who are using the failed core.

2.3 Using Multiple Cores

In order to use multiple cores, group members (or their first-hop routers) need a mechanism to discover the identities of cores and select the one they will use.

We refer to the first of these issues as *core advertisement*. For single-core protocols such as PIM, the current method for advertising cores [4] is to distribute a set of candidate cores throughout the network. When a source or group member 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 n 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. In most cases, choosing the nearest core should give good performance. If a core becomes congested, however, then a member or sender may want to switch to a different core.

Finally, the placement of the cores can impact the cost and delay of the multicast tree. We examine the effects of several core placement algorithms in the next section.

2.4 Implementation Details

The primary issue that must be addressed to implement our multiple core designs is packet forwarding. The current multicast routing architecture allows for only

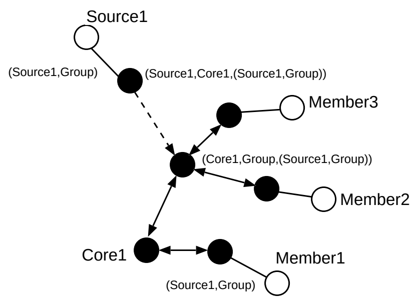


Fig. 3. Using a Bidirectional Tree at a Core

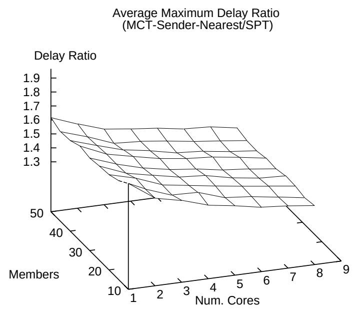
one shared tree per group. However, in our designs, there are several or many shared trees per group (one per core). These trees may overlap and thus must be identified and built individually.

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; the packet must arrive on the incoming interface listed in the entry and is sent on all outgoing interfaces. Likewise, 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).

Multiple core trees, as described above, require a new type of multicast routing entry. We call this a (C, G) entry, where C is the IP address of the core. A router that receives a multicast packet matches it against (C, G) entries the same as it would for an (S, G) entry. However, if a match is found, then the packet is forwarded on a bidirectional tree; that is, it is sent on all interfaces listed except for the interface on which it arrived. In practical terms, this change may only require a few bits in the routing entry to specify how forwarding should be performed.

Fig. 3 illustrates how packets are forwarded on a multiple-core tree using this new type of routing entry. When a source sends a packet to the group, its first-hop router encapsulates it and unicasts the packet toward its nearest core. When it reaches a router on the core's bidirectional tree, this router removes the original packet and then re-encapsulates it, 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. 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



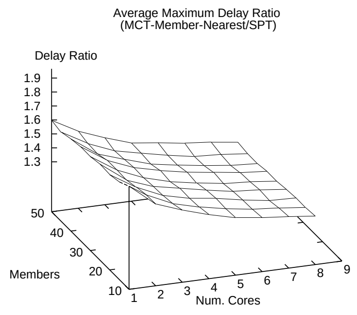


Fig. 4. Delay for Random Core Selection and Nearest Attachment: Senders-To-All (left) and Members-To-All (right)

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.

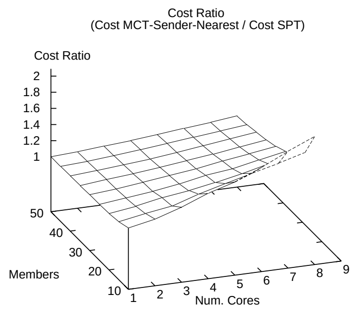
3 Simulation Study

We evaluate the feasibility of multiple-core trees by comparing their performance to single-core trees, as has been done in several previous studies comparing tree types [2] and core selection algorithms [5]. The factors in our experiment include group size (from 5 to 50), core selection (random and dominating set), and core attachment (random and nearest). The metrics we evaluate include cost, delay and traffic concentration. We report ratios to the corresponding SPT metrics, so that we can compare results across different graphs and groups.

3.1 Experiment Results: Delay

For these experiments, we use a set of 10 flat, random graphs of 50 nodes each, using the Waxman model [8] within the GT-ITM topology generator [9]. All edges have unit weights and the average node degree is near 4. For each graph, we generate 50 random groups and measure the delay experienced by group members. We define *delay* as the number of links traversed between one sender and one member in the group. We calculate the maximum delay for each sender, then average these numbers to find the average-maximum for the group.

We find that both Senders-To-All and Members-To-All can significantly reduce the delay experienced by group members when nearest attachment is used. As shown in Fig. 4, the delay decreases dramatically as the number of cores increases. This is because there is a greater chance that the member (or sender) will choose a core that is close to the shortest path.



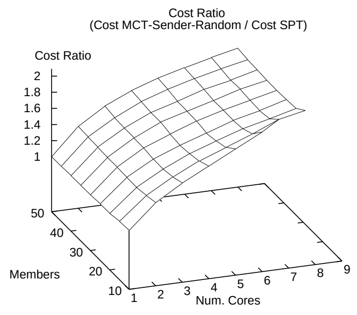


Fig. 5. Cost for Senders-To-All with Random Core Selection: Nearest Attachment (left) and Random Attachment (right)

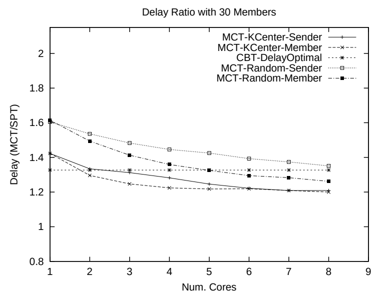
Particularly interesting in these results is that, for nearest core attachment, most of the benefits are seen with only 5 cores, after which the graphs are mostly flat. Thus only a small number of nodes – about 10% – need to be used as cores for a group.

Our results also show that the Members-To-All design is tolerant of members choosing distant cores: with random attachment delay increases only slightly as more cores are added. The Senders-To-All design, on the other hand, can suffer from large maximum delays when using random attachment. In the extreme case, with both a large group and a large number of cores, there is good chance that a member will choose to use some core that is distant from both itself and the sender. With the Senders-To-All design, the sender must transmit to all cores, so it will use the distant core and incur a large maximum delay.

3.2 Experiment Results: Cost

To evaluate cost, we use the same experiment setup as for delay. We define *cost* as the number of links in a tree, representing 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.

We find that the Members-To-All design performs better than Senders-To-All with respect to cost. Sending packets using Members-To-All does not consume much more bandwidth than a shortest-path tree. The cost ratio is nearly flat and close to 1 for both random and nearest attachment. In fact, the Members-To-All design actually performs slightly better than a single-core tree in this regard. This is because a sender is able to choose a core whose performance is very close to that of a shortest-path tree.



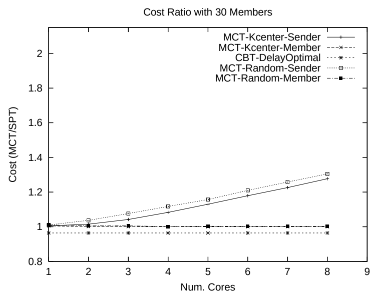


Fig. 6. *K*-Center Placement and Nearest Attachment: Delay Ratio (left) and Cost Ratio (right)

Senders-To-All also performs well with regard to cost as long as nearest attachment is used (see Fig. 5). The only exception is that the cost increases for small groups with many cores, due to the number of copies generated on links close to the sender.

Senders-To-All does not perform well with random attachment; as shown in Fig. 5 the cost ratio 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.

3.3 Experiment Results: Core Selection

To examine the effects of core selection, we used both a *dominating set* algorithm and a *k-center* algorithm. A dominating set is a subset of the nodes in the graph such that all nodes are within n hops of this set. A k-center 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 cores is NP-hard. We use approximation algorithms based on node degree, and our experiments indicate these are good approximations in random graphs.

We find that both dominating set placement and k-center placement improve the delay and cost ratios for multiple-core trees. Delay is reduced by 10 to 20% for the different designs; the cost ratio is close to 1 already so the improvement is slight. Moreover, as the number of cores increases, multiple core trees using both random core selection and k-center placement can have lower delay than a single-core tree with optimal core placement! Fig. 6 shows these results. In this same figure we also show the cost ratio for the corresponding experiment. Again, K-center placement is an improvement over random core placement. The single-core tree with optimal core placement retains the lone advantage that it can have lower cost than a shortest-path tree. This is in keeping with the result from Calvert et al. [5] in which optimal core placement is the only placement mechanism they study where single-core trees have lower cost than shortest-path trees.

3.4 Experiment Results: Traffic Concentration

For traffic concentration, we followed the methodology used by Wei and Estrin.[2] For a given graph, we generate 300 groups and construct their multicast trees. Then, for each group, we transmit a single packet from each source. As each packet is sent, we count the number of times each link in the network is traversed. With multiple-core trees, it is possible that a given link is traversed more than once by a single packet, due to encapsulation.

Our results show that as long as random core selection is used, both Members-To-All and Senders-To-All do not suffer from traffic concentration, regardless of the number of cores. This confirms the results of Calvert et al. [5] indicating that traffic concentration is only observed when a small number of cores is used across all groups. With random core selection, each group uses a different set of cores, so the traffic is spread throughout the network.

We do observe traffic concentration when using k-center placement, because in this case all groups use the same set of cores. In this situation, the effects of traffic concentration appear when there are fewer than 4 cores. As the number of cores increases beyond this amount, traffic concentration disappears.

4 Conclusions and Future Work

Our results indicate that multiple-core trees are a feasible alternative to shortest-path trees. They can have lower delay than trees using a single core and cost comparable to shortest-path trees. In addition, multiple-core trees do not suffer from traffic concentration, as long as a reasonably large set of candidate cores is used. An ISP may also use a k-center algorithm to choose a static set of cores; this can reduce delay and will avoid traffic concentration as long as a large enough set is used.

We are particularly interested in the Senders-To-All variant since it uses less router state and provides group members with more flexible control when reacting to failed cores and congestion. In most situations, the delay and cost of the Senders-To-All design are close to that experienced with shortest-path trees, as long as nearest attachment is used. In this case, its primary drawback is its higher cost with small groups and large numbers of cores. We are exploring ways to reduce cost in this situation. Costs can increase when members choose distant cores, but a natural disincentive (higher delay) or policy rules will likely prevent this from happening.

The Members-To-All variant has better performance in terms of cost and delay, and is also tolerant of group members choosing distant cores. It is less flexible with regard to fault tolerance and uses more router state, but we still consider it a viable option for multicast.

We are continuing to explore the design of multiple-core multicast protocols, particularly with respect to single-source multicast. We are also intrigued by the possibility of multiple core trees that use core distribution and are actively investigating the cost and delay attributes of these structures as well.

Acknowledgments

The authors would like to thank Virginia Lo for substantial contributions during the development of this work, as well as Krishnan Sivaramakrishna Iyer for help developing the dominating set approximation algorithm.

References

1. S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D.Estring, and M. Handley, "The MASC/BGMP Architecture for Inter-domain Multicast Routing," in *ACM SIGCOMM*, August 1998.
2. Liming Wei and Deborah Estrin, "The Trade-offs of Multicast Trees and Algorithms," in *1994 International Conference on Computer Communications Networks*, September 1994.
3. 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.
4. Deborah Estrin, Mark Handley, Ahmed Helmy, Polly Huang, and David Thaler, "A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing," submitted to ACM/IEEE Transactions on Networking.
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. A. J. Ballardie, P.F. Francis, and J. Crowcroft, "Core Based Trees," in *ACM SIGCOMM*, August 1993.
7. C. Shields and J.J. Garcia-Luna-Aceves, "The Ordered Core Based Tree Protocol," in *IEEE INFOCOM*, 1997.
8. Bernard M. Waxman, "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, December 1988.
9. Ken Calvert and Ellen Zegura, "Georgia Tech Internetwork Topology Models," http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html.