Overlay Optimizations for End-host Multicast

Wenjie Wang David A. Helder Sugih Jamin* Lixia Zhang
Department of EECS
University of Michigan
Ann Arbor, MI 48109
{wenjiew,dhelder,jamin}@eecs.umich.edu

Computer Science Department
University of California
Los Angeles, CA 90095
lixia@cs.ucla.edu

ABSTRACT

End-host multicast alleviates the deployment hurdle of IP multicast by building a transport level overlay for data delivery to all members in a multicast group, but at the performance penalty of higher network resource usage and higher latency in data delivery. In this paper we present a systematic approach to reducing latencies between members of an end-host multicast group. Latency reduction is achieved by adding links to an existing overlay network. We have developed several local heuristics to enable individual multicast group members to add new links as needed. We show that, using these local heuristics, data delivery by an end-host multicast overlay can achieve a latency from 30% over to 2.5 times that of native IP multicast, depending on the structure of the underlying physical topology. Our results are applicable to any end-host multicast protocols, either tree-based or mesh-based. A protocol based on these heuristics, called TMesh, has been developed. TMesh can be used in conjunction with any of the existing tree-based protocols to shorten the latency per node pair. It can support large end-host multicast groups with relatively low overhead.

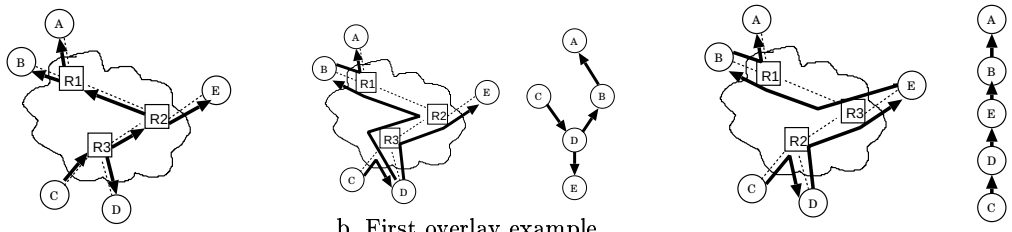
1. INTRODUCTION

Twenty years after its introduction, IP Multicast [1] is still not ubiquitously available on the Internet. Recent efforts to provide multicast delivery have thus shifted on to end-host multicast which builds a transport-layer overlay network between members of a multicast group. There is a rich literature on the design, implementation, and evaluation of various protocols for end-host multicast, for example, BTP [2], HMTP [3], Hypercast [4], Narada [5], and Yoid [6]. End-host multicast avoids the deployment hurdles of IP Multicast at the cost of higher data delivery latencies, as illustrated in the example network in Fig. 1. Nodes A, B, C, D, and E in Fig. 1 are members of a multicast group, nodes $R1$, $R2$, and $R3$ are routers in the network, and the dashed lines connecting the nodes are physical links. The arrows in Fig. 1(a) illustrate how IP multicast would forward data sent by C to the other members of the group. Fig. 1(b) shows a possible end-host multicast overlay where members B, C, and E are connected to member D, and member A is connected to member B.[1] For C to multicast a packet to the group, it forwards the packet to D, which forwards it to B and E; at B the packet is further forwarded to A. The right hand graph of Fig. 1(b) shows the overlay without the underlying physical network. Comparing Figs. 1(a) and (b), it is clear that data delivery using end-host multicast experiences longer delays than native IP Multicast delivery. For example, the latency from C to A on the overlay network is extended at least by the round-trip times between $R3$ and D and between $R1$ and B. Fig. 1(c) shows an even less efficient overlay where multicast packets from C to A traverse D, E, and B before reaching A. This example shows that latencies between members in an end-host multicast overlay depend largely on the quality of the overlay built.

Existing end-host multicast protocols can be categorized into tree-based and mesh-based protocols by the type of overlay they build. A tree is an overlay where there is a single path between any node pair, while a mesh may support more than one path between any node pair. BTP, HMTP, and Yoid are examples of tree-based protocols. Because a tree overlay is an acyclic graph, if any non-leaf member leaves the multicast group or crashes, the tree is partitioned and members in one partition will not be able to communicate with members in the other partition. Tree-based approaches thus require partition detection and recovery mechanisms. A mesh-based overlay, on the other hand, has redundant connectivities among group members, thus is less likely to get partitioned. However, the existence of redundant connectivities *requires* members to run a routing algorithm to construct loop-free forwarding path(s) between members. Narada and Hypercast are examples of mesh-based protocols. Narada uses a path vector algorithm

*At UM this research is supported in part by the NSF CAREER Award ANI-9734145, the Presidential Early Career Award for Scientists and Engineers (PECASE) 1998, the Alfred P. Sloan Foundation Research Fellowship 2001, and by the United Kingdom Engineering and Physical Sciences Research Council (EPSRC) Grant no. GR/S03577/01, and by equipment grants from Sun Microsystems Inc. and HP/Digital Equipment Corp. Part of this work was completed when Sugih Jamin was visiting the Computer Laboratory at the University of Cambridge. At UCLA this research is funded by NSF grant number ANI-9902925.

[1]We assume "connections" on end-host multicast overlays to be bi-directional unicast connections, either as reliable TCP connections or connectionless UDP sessions.



a. IP Multicast b. First overlay example c. Second overlay example

Figure 1: IP Multicast vs two end-host multicast overlays

for routing on the overlay. Hypercast assigns a logical address to each member. This logical address encodes routing information from which the next forwarding hop towards the destination can be obtained. Unless underlying physical network latencies are taken into account in the construction (and reconstruction) of the overlay, routing by logical addresses can result in long latencies between node pairs.

Although partition is less likely to happen on a mesh overlay, it is not impossible. In particular, members must avoid forming isolated cliques. Once an efficient tree-based overlay is formed, on the other hand, multicast packets can be forwarded along the tree without the further aid of a routing algorithm. Unfortunately, latencies between node pairs on a tree-based overlay are likely to be suboptimal. In the multicast tree shown as solid lines in Fig. 2, for a packet sent by E to reach H, it must first travel up to the root of the tree (R) and then down the other side of the tree to H. If we add an extra link between A and C, shown as the dashed line in the figure, data delivery between E and H is shortened by one hop. Paths between other members on the tree would also benefit from this extra link; in general, the distance between the subtree rooted at C and the subtree rooted at A is shortened by one hop. Adding a few extra links to a multicast tree may result in an overlay with lower latencies between members. We call these extra links "shortcuts," a tree with shortcuts a *TMesh*, the process of adding shortcuts to a tree the *TMesh optimization process*, and the protocol for adding shortcuts the *TMesh protocol*. Fig. 3 shows an example TMesh.

Compared to a tree-based multicast overlay, a TMesh provides shorter latencies between members in the group. It is also more robust against partitioning as it provides redundant connections between members. During tree reconstruction, after a member leave for example, nodes can switch parent without any packet loss. Compared to a mesh-based overlay, partition detection is much easier in a TMesh. As long as the tree "skeleton" is maintained, the graph is guaranteed connected. While mesh-based approaches require "refresh" messages to be periodically sent over the whole mesh to detect partitioning, a TMesh only needs to detect partitioning of the tree, thus reducing partition detection overhead. Furthermore, for multicast groups with a small number of senders, TMesh can be used without running a routing algorithm.

Our experiments show that a TMesh can reduce latencies between node pairs even when shortcuts are selected randomly. Careful selections of the shortcuts allow a TMesh to achieve an average node-pair latencies from 30% over to 2.5 times that of native IP multicast, depending on the structure of the underlying physical topology. Obviously TMesh optimization is useful only if it can operate in a distributed manner. Each node must decide for itself which shortcuts to add to the tree. In Section 3, we evaluate the gain in node-pair latencies achievable with several simple heuristics for shortcut selection. While a number of works have been published on various end-host multicast overlay construction protocol, we are not aware of any work that systematically investigates the effect of adding new links to the overlay to reduce node-pair latencies. In Section 4 we show that TMesh optimization can be used in conjunction with the tree-based Yoid end-host multicast protocol to achieve lower node-pair latencies. We also evaluate the performance of TMesh with various group sizes. We see consistent performance improvement in all cases. Since a TMesh is but a mesh "grown" from a spanning tree, the heuristics we propose for shortcut selection can be equally applied to other mesh-based overlays. In section 6, we summarize our work.

2. PERFORMANCE GAIN OF TMESH

Before delving into the details of shortcut selection heuristics, we want to be more specific in the potential performance gain afforded by TMesh. In particular, since a TMesh is but a mesh, we want to know how it compares with existing mesh-based overlays such as Narada. The *main* performance metric we adopt in this paper, given that the focus of our study is on node-pair latency achieved under various overlays, is the *Average Relative Delay Penalty (ARDP)*. As defined in [5], Relative Delay Penalty (RDP) is the ratio of the latency $D'_{i,j}$ between a node pair i and j on the overlay to the latency $D_{i,j}$ between them on the physical network. ARDP is then the average RDP between all node pairs:

$$ARDP = \frac{1}{N(N-1)} \sum_{i=0}^{N} \sum_{j=0, j \neq i}^{N} \frac{D'_{i,j}}{D_{i,j}}, \qquad (1)$$

where N is the number of members in the multicast group (nodes in the overlay). The smaller the ARDP, the closer node-pair latencies on the overlay are to latencies on the physical network. If the overlay were a full mesh and all members were directly connected to each other, the ARDP would be 1. Fig. 4 shows that not only does TMesh achieve lower latency per node-pair than tree-based overlays such as HMTP and Yoid, it also achieves lower node-pair latencies than Narada, which is a mesh-based protocol. The numbers reported in the figure were obtained from simulations on a 4,000-node topology generated using the Inet-3.0 topology generator [7]. The x-axis shows that the multicast overlays built range in size from 50 to 1,000 members. For each membership size, for each overlay construction protocol, we run the simulation ten times, with different set of nodes

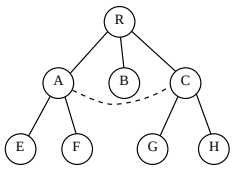


Figure 2: An example of shortcut link.

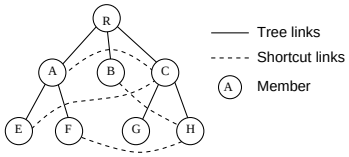


Figure 3: An example overlay built by TMesh.

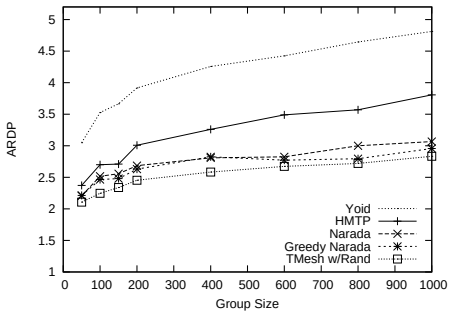


Figure 4: ARDP performance of various overlays.

selected as members, and report the average ARDP on the y-axis. In addition to ARDP, we also look at the 95%-tile RDP in Section 4.

The overlays built by HMTP and Yoid are trees, consisting of $N-1$ links, whereas the number of links in an overlay built by Narada depends on the settings of various parameters used with the protocol (see below). Fig. 4 shows that node-pair latencies on the HMTP trees are 21% to 26% lower than those on Yoid trees. While both HMTP and Yoid try to minimize the latency between each node and the root of the tree through tree reconfigurations,[2] HMTP incorporates several heuristics to recognize and avoid local minima in the tree construction process (see [3]).[3] Nevertheless, HMTP cannot achieve the same node-pair latencies as Narada, which uses more than $N-1$ links on the overlay. The node-pair latencies on HMTP trees are 6% to 24% higher than those on Narada overlays. For the TMesh results reported in Fig. 4, we added a number of links (shortcuts) to trees constructed by HMTP. Links are added between nodes chosen randomly with uniform probability. The total number of links in the resulting TMesh is the same as that on the Narada overlay. The figure shows that our simple TMesh overlay with random shortcuts reduces node-pair latencies by 11% to 26% compared to those on the original HMTP trees. More interestingly, the figure also shows that node-pair latencies on the simple TMesh are 5% to 11% lower than those on Narada overlays!

Nodes running Narada periodically pick a random non-neighbor member and calculate the *utility* of forming a link to that member. The utility of a link (u,v) to node u is defined to be the reduction in distance[4] between node u and all other nodes on the overlay if the link is added to the overlay:

$$util_u((u,v)) = \sum_{j \in G'} \frac{D'(u,j) - D''(u,j)}{D'(u,j)}, \qquad (2)$$

where G' is the set of nodes in the overlay, $D'(u,j)$ the distance between u and j without (u,v) on the overlay, and $D''(u,j)$ the distance with (u,v). When distance or path vector routing algorithm is used to route packets on the overlay, u can simply compute $util_u((u,v))$ by obtaining v's routing table. If the utility of a potential link is above a threshold value, the link is added to the overlay. Following [5], we use a utility threshold of $N/6$ to add a link. Narada also defines a *consensus cost* metric. Links with consensus cost below a threshold is dropped from the overlay.[5] Again, following [5], we use $N/12$ as the consensus cost threshold. To study whether different settings of these threshold values would negate the relative performance gain of TMesh over Narada, we also compare TMesh against what we call "Greedy Narada." With Greedy Narada, instead of picking potential neighbors at random, each node computes the utilities of *all* $N-1$ links between itself and all other nodes on the overlay. Links are added in decreasing utility until the resulting overlay has the same number of links as in the corresponding Narada overlay under study. Similarly, to drop a link under Greedy Narada, the link with the lowest global consensus cost is dropped first. Due to the computation costs incurred, we do not expect Greedy Narada to be practically deployable, we include it here to approximate the minimum node-pair latencies achievable with the Narada protocol. Fig. 4 shows that our simple TMesh can achieve lower node-pair latencies even compared to Greedy Narada.

Comparing the node-pair latencies on Narada, Greedy Narada, and TMesh w/Rand, it is clear that some shortcuts can lower the ARDP more than others. In the next section, we correlate several locally measurable characteristics of a shortcut to the eventual gain in ARDP when the shortcut is added to the overlay.

3. SHORTCUT SELECTION

How would a node decide which shortcut to add to the overlay? For a given overlay, we compute off-line, in a centralized manner, assuming detailed knowledge of the global overlay structure, the effect of *each* potential shortcut on overall node-pair latencies. We do not expect a scalable end-host multicast protocol to perform similar computations online. Hence we next try to correlate locally measurable characteristics with the computed reduction in node-pair laten-

[2]We waited until the trees stabilize before collecting the numbers reported here.

[3]A local minima arises for example when a node fails to switch to a parent closer to the root that has already reached its node degree limit, which in turn prevents the node from switching to other parents even closer to the root (see [2, 3]).

[4]In all simulation results reported here, the distance between two nodes are the Euclidean distance between the nodes on the simulated plane.

[5]Whereas the utility of a link is computed relative to each node incident to it, a single consensus cost is computed for the link based on its usefulness to both nodes incident to it.

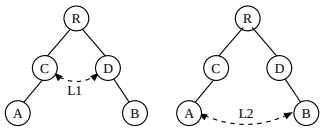


Figure 5: Example of link selection.

cies attributable to each shortcut. From these correlations we propose a set of heuristics for shortcut selection.

The effect of a shortcut on overall node-pair latencies depends to a large extent on the structure of the overlay topology. Fig. 5 shows a simple sample topology. Consider the two potential shortcuts $L1$ and $L2$ depicted in the figure. Assuming shortest path routing on the overlay, adding $L1$ reduces the shortest path between four node pairs (CD, AD, BC, and AB) by one hop each. Adding $L2$ reduces the shortest path between two node pairs (AD and BC) by one hop each, and between one node pair (AB) by three hops. $L2$ reduces the shortest path lengths between fewer node pairs than $L1$, but the AB node pair sees larger reduction. The ARDP metric accounts for both number of node pairs with reduced path lengths and the size of the reductions. When two shortcuts are added to an overlay, the resulting reduction in ARDP is unfortunately not the sum of the reductions when each of them is added singly. After the first shortcut is added, adding the second shortcut may not result in as big a reduction in ARDP as when it was added without the first shortcut.

For a tree overlay with N nodes, there are $N(N-1)/2 - (N-1) = (N-1)(N-2)/2$, or $O(N^2)$, potential shortcuts and $2^{(N-1)(N-2)/2}$ possible combinations of shortcuts. Even taking node degree limit into account and assuming that each node can add only one or two shortcuts on average, we are still looking at $O(2^N)$ possible combinations of shortcuts, which is simply beyond our computing power to simulate, for any meaningfully large N. Instead, we study the effect of adding each link in isolation.[6] Starting with an initial overlay, we add *one* extra link to it, and compute the reduction in ARDP due to this single shortcut. Then we remove the link, add another one, and compute the ARDP reduction brought on by this other shortcut. We continue in this vein for all $O(N^2)$ potential shortcuts. In computing ARDP reduction, we use incremental Dijkstra shortest path first (SPF) algorithm [8]. We experiment with group sizes ranging from 50 to 1,000 members, with the members randomly distributed with uniform distribution on a 4,000-node random topology generated using the Inet-3.0 topology generator.[7] For each group size, we run ten simulations with varying member sets and different initial overlays. All the numbers reported in the remainder of this paper are from this set of simulations.

The experiment described above provided us with an ARDP reduction value for each potential shortcut. We next try to correlate locally measurable characteristics with reduction in ARDP. We look at two locally measurable characteristics already introduced in the previous section: *RDP gain* and *link utility*.

[6]In Section 4 we look at several sample paths of adding multiple links.

[7]We have also conducted experiments on 6,000- and 8,000-node topologies, the results are comparable and are not included due to space constraints.

3.1 hRDP

Recall that RDP (Relative Delay Penalty) between two nodes is the ratio of the latency between the nodes on the overlay over the latency on the underlying physical network. When a shortcut is added between the two nodes, the RDP between the two nodes on the overlay becomes one. The original RDP between the two nodes before the shortcut addition is thus the *RDP gain* of the shortcut. When a shortcut is added, it can be incorporated into the shortest paths of multiple node-pairs, depending on the outcome of the Dijkstra SPF computation. Hence the RDP gain of a shortcut is not necessarily linearly correlated with the overall ARDP gain.

Fig. 6 plots the RDP gain of a shortcut against the overall ARDP improvement when the shortcut is added. The numbers are averaged from ten simulations with overlay size of 600 nodes. The ARDP improvements are small because we are only looking at the effect of adding a *single* link to the overlay. The figure shows the importance of the initial overlay. The HMTP overlay avoids local minima in the tree construction process. As evident in Fig. 4, HMTP's overlays already have lower ARDP than Yoid's overlays even without TMesh. When the initial overlay does not avoid local minima in the construction process, each individual shortcut can not improve ARDP by a large amount. For well constructed initial overlay, there is a clear correlation between RDP gain and ARDP gain.

From the above results, one can form the heuristics to add only shortcuts with RDP gain above a certain threshold (we call this heuristics *hRDP*). Fig. 7 shows how many potential shortcuts (y-axis) have a given RDP gain (x-axis). Even for the HMTP overlay, there are more than 10,000 potential shortcuts with RDP gain larger than 10. To calculate the RDP between nodes u and v, node u needs to measure both its latencies to v on the physical network and on the overlay. Latency on the physical network can be obtained using `ping` or by querying a distance estimation service such as IDMaps [9]. Latencies between a source and its receivers on the overlay can be measured by the use of timestamps, after taking clock synchronization into account. For tree-based protocols in which each node retains a root path (the path of the node to the root on the tree), such as HMTP and Yoid, latencies on the overlay between two members can be computed simply by merging the two root paths. We will briefly discuss some practical implementation issues in setting the appropriate threshold value in Section 5.

3.2 hUtil

In addition to hRDP, one can also form a heuristics similar to Narada's: add only shortcuts with average utility above a certain threshold (we call this heuristics *hUtil*). The average utility is simply the utility computed in Eq. 2 divided by the number of destinations. Recall that the utility metric captures how much closer the shortcut can bring nodes incident to it to *all* destinations. By design, each node running Narada computes the shortest paths to all other nodes using a path vector algorithm. In TMesh, the initial skeletal tree-based overlay ensures that the overlay is not partitioned. If the application using the overlay does not require low latencies between all node pairs, shortcuts can be added to reduce latencies only to data senders. Each node (acting as receiver) maintains its distance information to all potential senders; utility of a shortcut (u, v) is computed only between

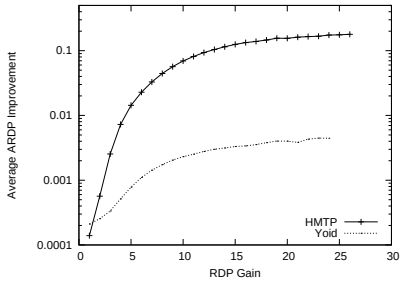


Figure 6: Average ARDP improvement of shortcuts with different RDP gains.

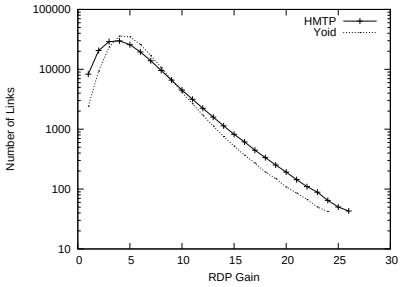


Figure 7: The number of shortcuts with different RDP gains.

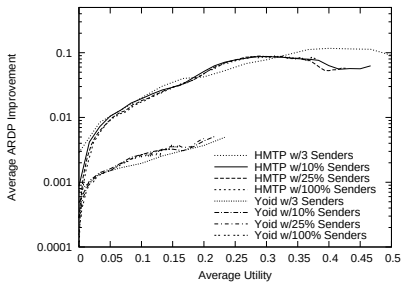


Figure 8: Average ARDP improvement of shortcuts with different average utility.

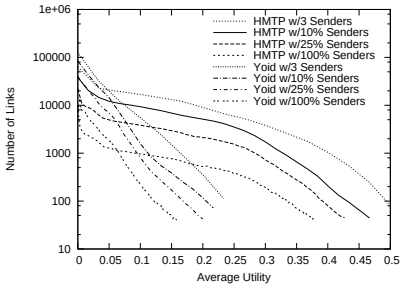


Figure 9: The number of shortcuts with different average utility.

a node (u) and potential senders:

$$util'_u((u,v)) = \sum_{j \in S} \frac{D'(u,j) - D''(u,j)}{D'(u,j)}, \qquad (3)$$

where S is the set of sender nodes.

Data delivery on the per-sender tree can be done by flooding on-tree neighbors. Such "receiver-based" shortcut construction process is possible in TMesh because links on overlays are bi-directional application-level connections. Similar to PIM sparse mode [10], each sender on TMesh starts out sending data on the underlying tree skeleton of TMesh. When the sending rate of a source goes above a threshold, receivers can start adding shortcuts to the overlay for that sender. When the sending rate of a source drops below the threshold, shortcuts to it can be automatically removed. Sources that send below this threshold can continue to use the tree skeleton.

3.3 ARDP Correlation

Fig. 8 shows the correlation between the locally computed average utility against ARDP improvements from ten simulations of 600-node overlays. The figure shows average utility computed when only 3 of the 600 nodes are senders, 10% of the nodes are senders, 25% of the nodes are senders, or all of the nodes are senders. In all cases, the ARDP improvements are computed between *all* node pairs, not just between senders and other members. As with hRDP, we see that each shortcut can only achieve a small ARDP gain when the initial tree does not avoid local minima during the construction process. Fig. 9 further shows that there are more potential shortcuts with higher average utilities in the HMTP case than in Yoid's case. The more surprising result apparent from Fig. 8 is how shortcuts that improve average utility to a small number of senders achieve the same overall ARDP gain as shortcuts that improve average utility to all nodes! Since senders are selected randomly, a shortcut that improves average utility to far away senders already provide maximal improvement in overall ARDP. From data not presented here, we note that when there are only a small number of senders, the maximum and minimum ARDP improvements are further from the mean than when there are a larger number of senders.

4. EVALUATION OF HEURISTICS

We showed in the previous section that locally measured RDP gain and average utility of a shortcut can be good indicators for its overall ARDP gain. In this section we evaluate the performance gain of these heuristics in constructing a TMesh. In particular, while we consider the ARDP gain of *individual* shortcut in the previous section, in this section we look at the performance gain of a TMesh as a whole, after multiple link additions. We use the same simulation scenarios used throughout this paper. In addition, we limit the degree of each node to ten. Of these, only six are allowed to connect to on-tree neighbors, the other four can only be used to form shortcuts. Each node add as many "qualified" shortcuts as its free degrees allow. A shortcut is considered "qualified" if either its RDP gain or average utility is above certain threshold, depending on the heuristics employed.

We briefly discuss some practical issues related to the setting of these thresholds in Section 5. For the simulations reported here, based on data presented in the previous section, we set the RDP gain threshold in the hRDP heuristics to $\sqrt{N}$, where N is the group size. For the hUtil heuristics, we assume 10% of the members are senders and set the average utility threshold for adding shortcuts to 1/8. In

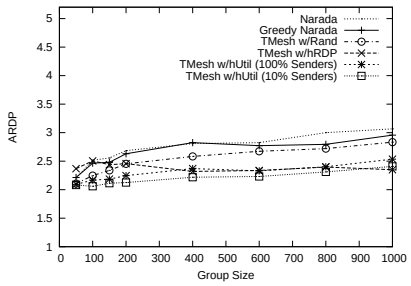


Figure 10: ARDP for different overlay schemes.

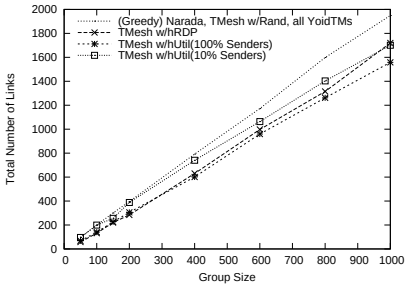


Figure 11: The number of links used in different overlay schemes.

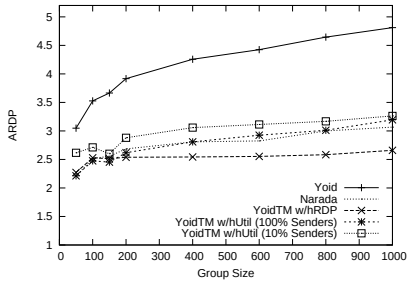


Figure 12: ARDP of *YoidTM* with different group size.

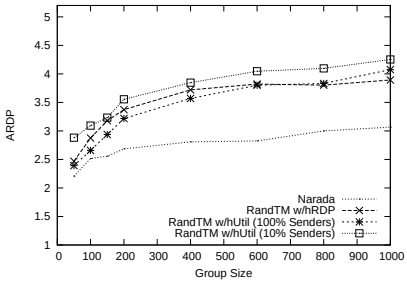


Figure 13: ARDP of TMesh overlay built on random tree.

addition to ARDP, we evaluate the performance of TMesh along several other metrics: 95%-tile RDP, node degree distribution, link load, and protocol overhead. We also run several simulations on a router-level Internet topology.

4.1 Multiple Shortcuts

Fig. 10 shows that TMeshes grown with the various non-random heuristics roughly achieve similar ARDP gain with each other for group sizes larger than 300. The figure also shows that TMeshes achieve lower ARDP than both Narada and "Greedy Narada". Fig. 11 shows the number of links used in the various meshes. We added as many random shortcuts in "TMesh w/Rand" overlays as necessary such that they have the same total number of links as the corresponding Narada and "Greedy Narada" overlays. "TMesh w/hRDP" uses fewer number of links than "TMesh w/Rand," which explains why it cannot achieve as low ARDPs as "TMesh w/Rand" for overlays with less than 150 nodes, as shown in Fig. 10. "TMesh w/hUtil (10% Senders)" uses more shortcuts in its meshes than "TMesh w/hUtil (100% Senders)," which explains why it can achieve lower ARDP than the latter.

From Figs. 6 and 8, we know that each individual shortcut does not provide as much gain in ARDP when the initial tree was built using Yoid. These figures also show that in Yoid's case, adding shortcuts based on their average utilities does not provide as much improvement in average ARDP as adding shortcuts based on their RDP gains. Fig. 12 bears out these observations. In all cases, we add as many shortcuts as necessary such that TMeshes built from Yoid's trees ("YoidTMs") have the same number of total links as the corresponding Narada's overlays. An interesting observation is that even though each individual shortcut contributes only a minuscule improvement to average ARDP, in aggregate, they provide quite a significant improvement in ARDP to Yoid-based TMeshes, as can be seen from Fig. 12.

In the previous section, we attributed the difference in performance between Yoid-based and HMTP-based TMeshes to the quality of the initial trees. Since "TMesh w/hRDP" reduces the RDP between nodes incident to the shortcuts, it practically "corrects" cases where the tree fell into local minima in the tree construction process. Hence we see that "YoidTM w/hRDP" achieves lower ARDP than "YoidTM w/hUtil." To further emphasize the crucial role of the initial tree from which TMeshes are grown, we show in Fig. 13 the performance of TMeshes when the initial trees are randomly generated ("RandTM"). In all cases, they do not perform as well as Narada's overlays, not even as well as YoidTMs. The efficient HMTP trees maintained inside the TMeshes play a crucial role in TMeshes' performance, which also explains the performance gain of TMesh over Narada even though the heuristic hUtil is practically what Narada uses in forming its overlays.

Figs. 14 and 15 show the CDF (Cumulative Distribution Function) and 95%-tile of RDP in the various overlays. The CDFs were computed from simulations with 600-node overlays. These figures provide a more complete picture of the performance gain afforded by the different overlays.

4.2 Node Degree Distribution and Link Load

Fig. 16 shows the node degree distribution in 600-node overlays constructed using the various construction processes. In all cases, the node degree limit is 10. Using TMesh, 65% to 70% of nodes have degree less than 3. Nodes with high degrees consume more resources. The larger number of low degree nodes in TMesh means that content providers can engineer their overlays and place well provisioned nodes on the network to serve as high degree nodes.

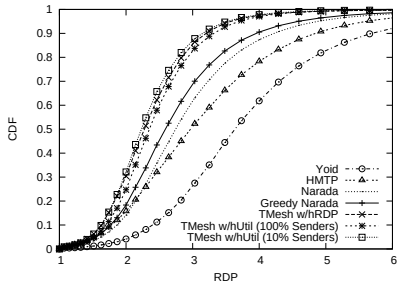


Figure 14: CDF of RDP for various overlays with group size 600.

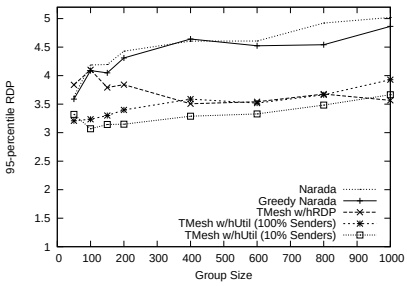


Figure 15: 95%-tile RDP of various overlays.

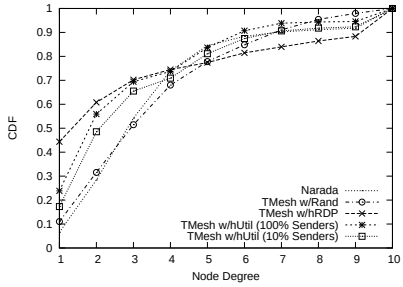


Figure 16: Node degree distribution.

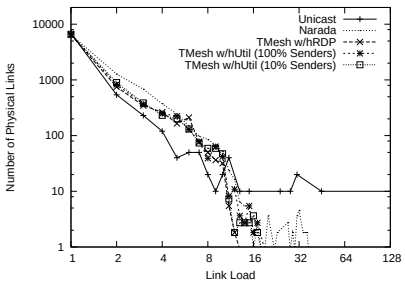


Figure 17: Link load.

When a node has a higher degree on the overlay than its number of physical connections, at least one of its physical connection will see multiple copies of the same data sent on the overlay. We call this the link's load. Link load of one means the physical connection sees only one copy of data sent on the overlay. Fig. 17 shows the distribution of link loads from ten simulations of 600-node overlays built with Narada. In each simulation, we pick one random node to serve as a data source and count how many copies of data sent traverse each physical link used to form the overlay. A routing algorithm is implemented on the overlay and data is forwarded through source-rooted shortest paths on the overlay. The figure also shows the link load distribution when unicast delivery is assumed. With unicast delivery, the source sends a copy of the data to each member directly using a separate unicast connection. The maximum link load in the unicast case is 597 (not shown), due to nodes having multiple physical connections. The maximum link load on TMesh is about half that of Narada.

4.3 Protocol Overhead

We categorize protocol overhead into two types: routing overhead and tree maintenance overhead. We do not consider packet header overhead as it does not have to be very different between protocols. Assuming that TMesh uses the same path vector algorithm used on Narada, Fig. 18 shows that the cost to maintain path vectors to all nodes ("100% Senders") on a 600-node TMesh overlay can be as high as 2.3 Mbits per update, comparable to the overhead on Narada. (The figure shows the routing overhead incurred by each node in descending order.) When receiver-based shortcut construction is employed (see Section 3), with only 10% of the nodes acting as senders, maximum routing overhead can be cut to about 233 Kbits per update.

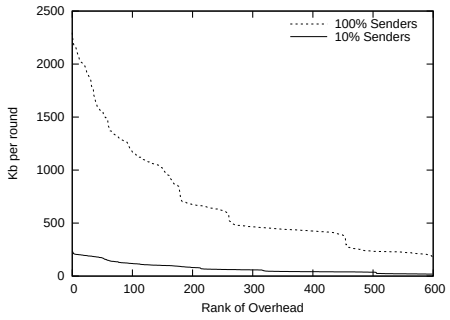


Figure 18: Routing overhead of each node running TMesh.

In Table 1 we show the overhead per second for *both* routing and tree construction/maintenance for various overlay construction mechanisms. The original Narada assumes all nodes are potential data sources. We also show the overhead when only 10% of nodes are potential sources, under a modified Narada and under TMesh implementing the hRDP and hUtil heuristics. The modified Narada keeps only distance information to the 10% of nodes that are potential sources. This modified Narada is included here only for rough comparison purposes. A deployable version would need a partition detection mechanism added, which will increase the overhead. In our simulations we use a routing update period of 15 seconds. Shortcuts are added in TMesh only after a member has remained connected to a parent for 45 seconds without switching. A member checks for better shortcuts every 30 seconds, which is doubled after every two shortcut additions.

Group size	50	100	150	200	400	600	800	1000
Narada (100% Senders)	2.590	4.243	5.261	6.031	15.084	21.443	26.626	29.748
Narada (10% Senders)	0.012	0.033	0.065	0.106	0.373	0.732	1.168	1.566
TMesh w/hUtil (10% Senders)	0.009	0.022	0.037	0.058	0.178	0.349	0.573	0.818
TMesh w/hRDP (10% Senders)	0.008	0.018	0.032	0.048	0.163	0.359	0.593	0.876

Table 1: Protocol overhead (Kbps).

4.4 Internet Topology

We construct a router-level topology of a large ISP from traceroute results. The traceroutes were initiated from 50 sites on the Internet to 200,000 IP addresses (see [11] for a detailed description of the topology construction process). The resulting topology contains 1,426 nodes. We run ten simulations for each overlay construction process on this topology. For each simulation, we randomly pick 200 nodes to serve as members of an end-host multicast group, the maximum node degree limit is set to 8. The resulting ARDPs are: 1.76 (Narada), 1.38 (TMesh w/hRDP), and 1.26 (TMesh w/hUtil, 10% senders). The 95%-tile RDPs are: 3.93 (Narada), 2.5 (TMesh w/hRDP), and 1.99 (TMesh w/hUtil). The protocol overhead are 0.37 Kbps (Narada, 10% senders), 0.248 Kbps (TMesh w/hRDP) and 0.178 Kbps (TMesh w/hUtil).

5. IMPLEMENTATION ISSUES

As with other network protocols, efficiency and scalability are some of the key goals in TMesh protocol design. We stipulate that each node running TMesh does not need to keep a full member list and that shortcut addition must be made by each node independently, based on locally measurable characteristics.

As described in Section 3, only shortcuts whose RDP gain or average utility is above certain thresholds are added to TMesh. Higher thresholds mean only links with high potential to reduce ARDP significantly are added to the TMesh. However, higher thresholds also mean that fewer links will qualify for addition, which unfortunately could lead to the TMesh having too few shortcuts. In general, two factors should be considered in setting shortcut selection threshold values:

Node degree limit: A shortcut can be added only if both nodes incident to it still have a free degree. The implication being that not all shortcuts with potential gains/utilities above threshold could actually be added. For example, to build a TMesh overlay with $3N$ links, we would need about $2N$ shortcuts. If we rank all potential shortcuts in descending order and set the threshold at the $2N$-th cut off point, we could very well end up with less than $2N$ shortcuts on the TMesh because not all of them could be added due to node degree limit.

Group size: It is more likely that we will see large latencies between members on large overlays than on small overlays. When TMesh is used with the hRDP heuristics, if the RDP gain threshold is set too low, relative to the group size, one could end up with a large number of shortcuts with values above the threshold. (For the hUtil heuristics, the computation of a shortcut's average utility already takes group size into account.)

6. CONCLUSION

In this paper we presented a systematic study on correlating locally measurable characteristics of a link with the potential reduction of node-pair latencies in host-based multicast data delivery. We proposed a protocol called TMesh that can be used in conjunction with any existing tree-based host multicast protocol to reduce node-pair latencies. We showed that, even with randomly selected shortcut links, TMesh achieves lower ARDP than Narada. Our simulation results also indicated that the initial topology used in growing TMesh has a critical effect on achievable ARDP. In particular, it is important that the protocol used to built the underlying tree avoids local minima in the construction process. Furthermore, we showed that adding shortcuts to a small number of far apart nodes can achieve most of the performance gain of having shortcuts to a large number of close-by nodes.

7. REFERENCES

[1] S. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proc. of ACM SIGCOMM '88*, pages 55–64, 1988.

[2] D. Helder and S. Jamin. End-host Multicast Communication Using Switch-Trees Protocols. *Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, 2002.

[3] B. Zhang, S. Jamin, and L. Zhang. Host Multicast: A Framework Delivering Multicast To End Users. *Proc. of IEEE INFOCOM '02*, 2002.

[4] J. Liebeherr and T. Beam. HyperCast: A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology. In *Networked Group Communication*, pages 72–89, 1999.

[5] Y. Chu, S. Rao, and H. Zhang. A Case For End System Multicast. *ACM Sigmetrics*, pages 1–12, 2000.

[6] P. Francis. Yoid: Extending the Internet Multicast Architecture. *Unrefereed report*, Apr. 2000. http://www.aciri.org/yoid.

[7] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report CSE-TR-456-02, EECS Department, University of Michigan, 2002.

[8] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Semidynamic Algorithms for Maintaining Single-Source Shortest Path Trees. *Algorithmica*, 22(3):250–274, 1998.

[9] P. Francis et al. IDMaps: A Global Internet Host Distance Estimation Service. *ACM/IEEE Transactions on Networking*, 9(5):525–540, 2001.

[10] S. Deering et al. The PIM Architecture for Wide-Area Multicast Routing. *ACM/IEEE Transactions on Networking*, 4(2):153–162, 1996.

[11] C. Jin. *Building a Scalable Network Measurement Infrastructure: Theory and Practice*. PhD thesis, University of Michigan, 2002.