A Case for End System Multicast *

Yang-hua Chu, Sanjay G. Rao, and Hui Zhang
Carnegie Mellon University
{yhchu,sanjay,hzhang}@cs.cmu.edu

ABSTRACT

The conventional wisdom has been that IP is the natural protocol layer for implementing multicast related functionality. However, ten years after its initial proposal, IP Multicast is still plagued with concerns pertaining to scalability, network management, deployment and support for higher layer functionality such as error, flow and congestion control. In this paper, we explore an alternative architecture for small and sparse groups, where end systems implement all multicast related functionality including membership management and packet replication. We call such a scheme End System Multicast. This shifting of multicast support from routers to end systems has the potential to address most problems associated with IP Multicast. However, the key concern is the performance penalty associated with such a model. In particular, End System Multicast introduces duplicate packets on physical links and incurs larger end-to-end delay than IP Multicast. In this paper, we study this question in the context of the Narada protocol. In Narada, end systems self-organize into an overlay structure using a fully distributed protocol. In addition, Narada attempts to optimize the efficiency of the overlay based on end-to-end measurements. We present details of Narada and evaluate it using both simulation and Internet experiments. Preliminary results are encouraging. In most simulations and Internet experiments, the delay and bandwidth penalty are low. We believe the potential benefits of repartitioning multicast functionality between end systems and routers significantly outweigh the performance penalty incurred.

1. INTRODUCTION

Traditional network architectures distinguish between two types of entities: end systems (hosts) and the network (switches and routers). One of the most important architectural decisions is then the division of functionality between end systems and networks.

In the Internet architecture, the internetworking layer, or IP, implements a minimal functionality — a best-effort unicast datagram service, and end systems implement all other important functionality such as error, congestion, and flow control. Such a minimalist approach is probably the single most important technical reason for the Internet's growth from a small research network into a global, commercial infrastructure with heterogeneous technologies, applications, and administrative authorities. The growth of the network in turn unleashes the development of new applications, which require richer network functionality.

The key architecture question is: what new features should be added to the IP layer? Multicast and QoS are the two most important features that have been or are being added to the IP layer. While QoS is a functionality that cannot be provided by end systems alone and thus has to be supported at the IP layer, this is not the case for multicast. In particular, it is possible for end systems to implement multicast services on top of the IP unicast service.

In deciding whether to implement multicast services at the IP layer or at end systems, there are two conflicting considerations that we need to reconcile. According to the end-to-end arguments [17], a functionality should be (a) pushed to higher layers if possible, (b) unless implementing it at the lower layer can achieve large performance benefit that outweighs the cost of additional complexity at the lower layer.

In his seminal work in 1989 [4], Deering argues that this second consideration should prevail and multicast should be implemented at the IP layer. This view so far has been widely accepted. IP Multicast is the first significant feature that has been added to the IP layer since its original design and most routers today implement IP Multicast. Despite this, IP Multicast has several drawbacks that have so far prevented the service from being widely deployed. First, IP Multicast requires routers to maintain per group state, which not only violates the "stateless" architectural principle of the original design, but also introduces high complexity and serious scaling constraints at the IP layer. Second, the current IP Multicast model allows for an arbitrary source to send data to an arbitrary group. This makes the network vulnerable to flooding attacks by malicious sources, and complicates network management and provisioning. Third, IP Multicast requires every group to dynamically obtain a globally unique address from the multicast address space and it is

*This research was sponsored by DARPA under contract numbers N66001-96-C-8528, F30602-99-1-0518, and E30602-97-2-0287, and by NSF under grant numbers Career Award NCR-9624979 ANI-9730105, and ANI-9814929. Additional support was provided by Intel, Lucent, and Ericsson. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, Lucent, Ericsson or the U.S. government.

difficult to ensure this in a scalable, distributed and consistent fashion. Fourth, IP Multicast is a best effort service. Providing higher level features such as reliability, congestion control, flow control, and security has been shown to be more difficult than in the unicast case. Finally, IP Multicast calls for changes at the infrastructural level, and this slows down the pace of deployment. While there have been attempts to partially address some of the issues at the IP layer [9, 15, 21], fundamental concerns pertaining to the "stateful" architecture of IP Multicast and support for higher layer functionality have remained unresolved.

In this paper, we revisit the issue of whether multicast related functionality should be implemented at the IP layer or at the end systems. In particular, we consider a model in which multicast related features, such as group membership, multicast routing and packet duplication, are implemented at end systems, assuming only *unicast* IP service. We call the scheme End System Multicast. Here, end systems participating in the multicast group communicate via an overlay structure. The structure is an overlay in the sense that each of its edges corresponds to a unicast path between two end systems in the underlying Internet.

We believe that End System Multicast has the potential to address most problems associated with IP Multicast. Since all packets are transmitted as unicast packets, network provisioning is not affected and deployment may be accelerated. End System Multicast maintains the stateless nature of the network by requiring end systems, which subscribe only to a small number of groups, to perform additional complex processing for any given group. In addition, we believe that solutions for supporting higher layer features such as error, flow, and congestion control can be significantly simplified by leveraging well understood unicast solutions for these problems. We hope to demonstrate this in future works. Finally, an end system based architecture no longer requires global consistency in naming of groups and allows for application specific naming.

While End System Multicast has many advantages, several issues need to be resolved before it become a practical alternative to IP Multicast. In particular, an overlay approach to multicast, however efficient, cannot perform as well as IP Multicast. It is impossible to completely prevent multiple overlay edges from traversing the same physical link and thus some redundant traffic on physical links is unavoidable. Further, communication between end systems involves traversing other end systems, potentially increasing latency. In this paper, we focus on two fundamental questions pertaining to the End System Multicast architecture: (i) what are the performance implications of using an overlay architecture for multicast? and (ii) how do end systems with limited topological information cooperate to construct good overlay structures?

In this paper, we seek to answer these questions in the context of a protocol that we have developed called *Narada*. Narada constructs an overlay structure among participating end systems in a *self-organizing* and *fully distributed* manner. Narada is robust to the failure of end systems and to dynamic changes in group membership. End systems begin with no knowledge of the underlying physical topology, and they determine latencies to other end systems by probing them in a controlled fashion. Narada continually refines the overlay structure as more probe information is available. Narada may be distinguished from many other self-organizing protocols in that it does not require a native multicast medium. We present details in Section 3.

We evaluate the performance penalty of the overlay Narada produces using simulations. In a group of 128 members, the delay between at least 90% of pairs of members increases by a factor of at most 4 compared to the unicast delay between them. Further, no physical link carries more than 9 identical copies of a given packet. We have also implemented Narada and conducted preliminary Internet experiments. For a group of 13 members, the delay between at least 90% of pairs of members increases by a factor of at most 1.5 compared to the unicast delay between them.

While we refer to end systems in this paper, it is with the understanding that this may easily be generalized to nodes at the edge of the network such as campus wide proxies and edge routers. Proxies can exploit native multicast available at the LAN level, have larger processing power than hosts, are better connected to the Internet and allow for organization level agreements. Further, they could potentially exploit temporal and spatial locality in group formations, At the other end of the spectrum, tree building mechanisms might be built into actual applications running on hosts, and could exploit application specific requirements and peculiarities. Irrespective of these differences, common to all these architectures is the design and evaluation of an actual self-organization protocol for construction of overlay structures for data delivery. An end system in our paper refers to the entity that actually takes part in the self-organization protocol, and could be a host, or an application or a proxy.

We believe End System Multicast is more appropriate for small size and sparse groups such as audio/video conferencing, virtual classroom and multiparty network games, but is not appropriate for handling applications such as Internet TV that involve a single source streaming high bandwidth and real time data to several hundred thousand recipients. Regardless, we believe End System Multicast is of interest as we hypothesize that there will be an explosion of small sized and sparse groups in the near future.

2. END SYSTEM MULTICAST

In this section, we contrast IP Multicast, End System Multicast and naive unicast with an example, and outline fundamental issues in the design of an End System Multicast protocol.

Consider Figure 1(a) which depicts an example physical topology. $R1$ and $R2$ are routers, while A, B, C and D are end systems. Link delays are as indicated. Thus $R1 - R2$ may be imagined to be a costly transcontinental link, while all other links are cheaper local links. Further, let us assume A wishes to send data to all other nodes.

Figure 1(b) depicts the IP Multicast tree constructed by DVMRP [4]. DVMRP is the classical IP Multicast protocol, where data is delivered from the source to recipients using an IP Multicast tree composed of the shortest paths from each recipient to the source. $R1$ and $R2$ receive a single copy of the packet but forward it along multiple interfaces. At most one copy of a packet is sent over any physical link. Each recipient receives data with the same delay as though A were sending to it directly by unicast.

End System Multicast does not rely on router support for

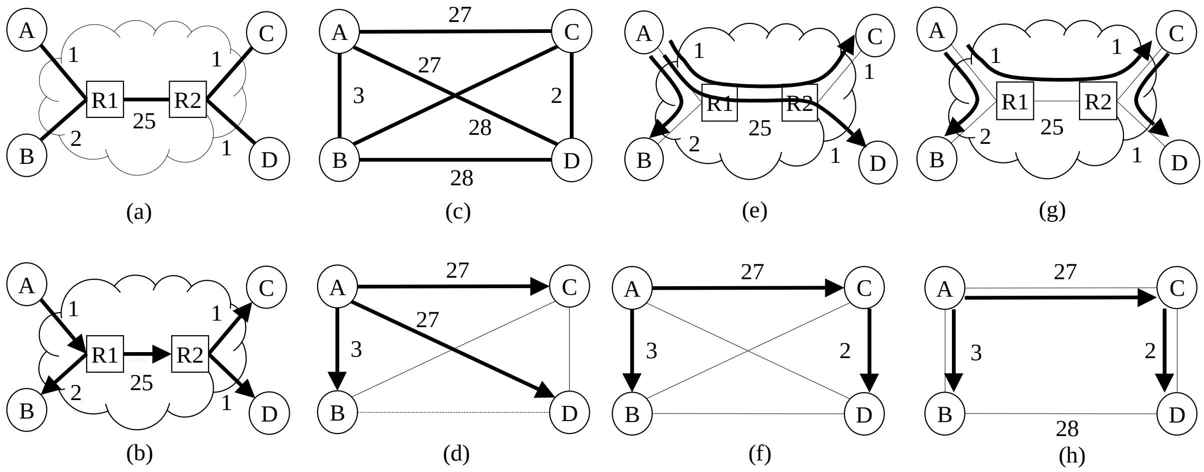

Figure 1: Examples to illustrate IP Multicast, naive unicast and End System Multicast

multicast. It abstracts the physical topology as a *Complete Virtual Graph* (CVG) as shown in Figure 1(c). Further, it tries to construct a spanning-tree of the CVG along which A could send data to other recipients. In particular, this scheme could degenerate to naive unicast transmission as shown in Figure 1(d). Figure 1(e) depicts how naive unicast transmission maps onto the underlying physical network. It is seen that links $R1 - R2$ and $A - R1$ carry 2 and 3 copies of a transmission by A respectively. We refer to the number of identical copies of a packet carried by a physical link as the *stress of a physical link*. Thus, unicast in general leads to a high stress on the link nearest the source.

We could build smarter spanning trees of the CVG such as the one shown in Figure 1(f). Figure 1(g) depicts how this tree maps onto the underlying physical network. It is seen that all physical links have a stress of at most 2. Not only does this tree reduce the worst case physical link stress, but it also reduces the stress of the costlier link $R1 - R2$ to 1. In order to capture this, we introduce the notion of *resource usage*. We define resource usage as $\sum_{i=1}^{L} d_i * s_i$, where, L is the number of links active in data transmission, d_i is the delay of link i and s_i is the stress of link i. The resource usage is a metric of the network resources consumed in the process of data delivery to all receivers. Implicit here is the assumption that links with higher delay tend to be associated with higher cost. The resource usage might be computed to be 30 in the case of transmission by DVMRP, 57 for naive unicast and 32 for the smarter tree , shown in Figure 1(b), Figure 1(d) and Figure 1(f) respectively.

While the spanning tree of Figure 1(f) improves link stress and resource usage as compared to naive unicast transmission, it increases delay from the source to some of the recipients. Thus, the delay from A to D has increased to 29 from 27. We refer to the ratio of the delay between two members along the overlay to the unicast delay between them as the *Relative Delay Penalty* (RDP). Thus, <A,D> has an RDP of $\frac{29}{27}$, while <A,B> and <A,C> have an RDP of 1.

The *out-degree* of a node in the overlay tree is its number of children in the tree. Thus, the out-degree of A in the smarter tree is 2, while it is 3 in naive unicast. The out-degree of a node in the overlay spanning tree directly impacts the stress of the links close to the node.

3. NARADA DESIGN

In this section, we present Narada, a protocol we designed that implements End System Multicast. In designing Narada, we have the following objectives in mind:

• *Self-organizing:* The construction of the end system overlay must take place in a fully distributed fashion and must be robust to dynamic changes in group membership.

• *Overlay efficiency:* The tree constructed must ideally have low stress, low *RDP* and low resource usage. Further, the out-degree of each member in the overlay must reflect the bandwidth of its connection to the Internet.

• *Self-improving in an incremental fashion:* The overlay construction must include mechanisms by which end systems gather network information in a scalable fashion. The protocol must allow for the overlay to incrementally evolve into a better structure as more information becomes available.

There are two basic methods for construction of overlay spanning trees for data delivery. A first approach is to construct the tree directly - that is, members explicitly select their parents from among the members they know [8]. Narada however constructs trees in a two-step process. First it constructs a richer connected graph that we term *mesh*. The mesh could in general be an arbitrary connected subgraph of the CVG (though Narada tries to ensure the mesh has desirable performance properties as discussed later.) In the second step, Narada constructs (reverse) shortest path spanning trees of the mesh, each tree rooted at the corresponding source using well known routing algorithms. Figure 1(h) presents an example mesh that Narada constructs for the physical topology shown in Figure 1(a), along with the shortest path spanning tree rooted at A. We have several reasons for this two step process. First, group management functions are abstracted out and handled at the mesh rather than replicated across multiple (per source) trees. Second, distributed heuristics for repairing mesh partition and mesh optimization are greatly simplified as loop avoidance is no longer a constraint. Third, we may leverage standard routing algorithms for construction of data delivery trees. Finally, a mesh is more resilient to the failure of members than a tree and heavy weight partition repair mechanisms are invoked less frequently.

In our approach, there is no control over the resulting spanning trees for a given mesh. Hence, it becomes important to construct a good mesh so that good quality trees may be produced. In particular, we attempt to ensure the following properties: (i) the shortest path delay between any pair of members along the mesh is at most K times the unicast delay between them, where K is a small constant and (ii) each member has a limited number of neighbors in the mesh which does not exceed a given (per-member) bound chosen to reflect the bandwidth of the member's connection to the Internet.[1] Limiting the number of neighbors regulates the fanout of members in the spanning trees. Second, it controls the overhead of running routing algorithms on the mesh. The extreme case where the mesh is chosen to be the Complete Virtual Graph incurs all the overhead of routing with none of its benefits as the resulting shortest path spanning trees degenerates to naive unicast transmission.

Narada has striking differences from self-configuring protocols developed in other contexts. First, Narada distinguishes itself from normal routing protocols in that it changes the very topology over which routing is performed. Second, most existing self-configuring protocols [12, 13, 14, 22] assume native IP Multicast support. Narada attempts self-configuration in the absence of a lower level multicast service, and this is fundamentally more challenging.

We explain the distributed algorithms that Narada uses to construct and maintain the mesh in Section 3.1. We present heuristics that Narada uses to improve mesh quality in Section 3.2. Narada runs a variant of standard distance vector algorithms on top of the mesh and uses well known algorithms to construct per-source (reverse) shortest path spanning trees for data delivery. We discuss this in Section 3.3.

3.1 Group Management

We have seen that Narada tries to construct a mesh among end systems participating in the multicast group. In this section, we present mechanisms Narada uses to keep the mesh connected, to incorporate new members into the mesh and to repair possible partitions that may be caused by members leaving the group or by member failure.

As we do not wish to rely on a single non-failing entity to keep track of group membership, the burden of group maintenance is shared jointly by all members. To achieve a high degree of robustness, our approach is to have every member maintain a list of all other members in the group. Since Narada is targeted towards small sized groups, maintaining the complete group membership list is not a major overhead. Every member's list needs to be updated when a new member joins or an existing member leaves. The challenge is to disseminate changes in group membership efficiently, especially in the absence of a multicast service provided by the lower layer. We tackle this by exploiting the mesh to propagate such information. However, this strategy is complicated by the fact that the mesh might itself become partitioned when a member leaves. To handle this, we require that each member periodically generate a refresh message with monotonically increasing sequence number, which is disseminated along the mesh. Each member i keeps track of the following information for every other member k in the group: (i) member address k; (ii) last sequence number s_{ki} that i knows k has issued; and (iii) *local time* at i when i first received information that k issued s_{ki}. If member i has not received an update from member k for T_m time, then, i assumes that k is either dead or potentially partitioned from i. It then initiates a set of actions to determine the existence of a partition and repair it if present as discussed in Section 3.1.3.

[1] An ideal mesh is a "Degree-Bounded K-spanner" [11] of the Complete Virtual Graph. The problem of constructing Degree-Bounded K-spanners of a graph has been widely studied in centralized settings that assume complete information and is NP-complete even in such scenarios [11].

Let i receive refresh message from neighbor j at i's local time t. Let $< k, s_{kj} >$ be an entry in j's refresh message.
- if i does not have an entry for k, then i inserts the entry $< k, s_{kj}, t >$ into its table
- else if i's entry for k is $< k, s_{ki}, t_{ki} >$, then
  - if $s_{ki} \geq s_{kj}$ i ignores the entry pertaining to k
  - else i updates its entry for k to $< k, s_{kj}, t >$

Figure 2: Actions taken by a member i on receiving a refresh message from member j.

Propagation of refresh messages from every member along the mesh could potentially be quite expensive. Instead, we require that each member periodically exchange its knowledge of group membership with its neighbors in the mesh. A message from member i to a neighbor j contains a list of entries, one entry for each member k that i knows is part of the group. Each entry has the following fields: (i) member address k; and (ii) last sequence number s_{ki} that i knows k has issued. On receiving a message from a neighbor j, member i updates its table according to the pseudo code presented in Figure 2.

Finally, given that a distance vector routing algorithm is run on top of the mesh (Section 3.3), routing update messages exchanged between neighbors can include member sequence number information with minimum extra overhead.

3.1.1 *Member Join*

When a member wishes to join a group, Narada assumes that the member is able to get a list of group members by an out-of-band bootstrap mechanism. The list needs neither be complete nor accurate, but must contain at least one currently active group member. In this paper, we do not address the issue of the bootstrap mechanism. We believe that such a mechanism is application specific and our protocol is able to accommodate different ways of obtaining the bootstrap information.

The joining member randomly selects a few group members from the list available to it and sends them messages requesting to be added as a neighbor. It repeats the process until it gets a response from some member, when it has successfully joined the group. Having joined, the member then starts exchanging refresh messages with its neighbors. The mechanisms described earlier will ensure that the newly joined member and the rest of the group learn about each other quickly.

3.1.2 *Member Leave and Failure*

When a member leaves a group, it notifies its neighbors, and this information is propagated to the rest of the group members along the mesh. In Section 3.3, we will describe our enhancement to distance vector routing that requires a leaving member to continue forwarding packets for some time to minimize transient packet loss.

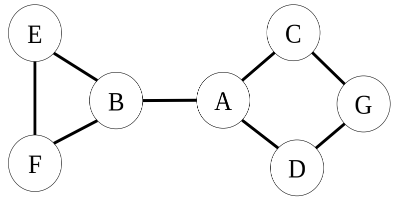

Figure 3: A sample virtual topology

```
Let Q be a queue of members for which i has stopped
receiving sequence number updates for at least Tm
time. Let T be maximum time an entry may remain in Q.
while(1) begin
    Update Q;
    while( !Empty(Q) and
           Head(Q) is present in Q for ≥ T time)
    begin
      j= Dequeue(Q);
      Initiate probe cycle to determine if j is dead
      or to add a link to it.
    end
    if( !Empty(Q)) begin
      prob = Length(Q)/ GroupSize;
      With probability prob begin
        j= Dequeue(Q);
        Initiate probe cycle to determine if j is dead
        or to add a link to it.
      end
     sleep(P). // Sleep for time P seconds
end
```

Figure 4: Scheduling algorithm used by member i to repair mesh partition

We also need to consider the difficult case of abrupt failure. In such a case, failure should be detected locally and propagated to the rest of the group. In this paper, we assume a failstop failure model [19], which means that once a member dies, it remains in that state, and the fact that the member is dead is detectable by other members. We explain the actions taken on member death with respect to Figure 3. This example depicts the mesh between group members at a given point in time. Assume that member C dies. Its neighbors in the mesh, A, G stop receiving refresh messages from C. Each of them independently send redundant probe messages to C, such that the probability every probe message (or its reply) is lost is very small. If C does not respond to any probe message, then, A and G assume C to be dead and propagate this information throughout the mesh.

Every member needs to retain entries in its group membership table for dead members. Otherwise, it is impossible to distinguish between a refresh announcing a new member and a refresh announcing stale information regarding a dead member. However, dead member information can be flushed after sufficient amount of time.

3.1.3 Repairing Mesh Partitions

It is possible that member failure can cause the mesh to become partitioned. For example, in Figure 3, if member A dies, the mesh becomes partitioned. In such a case, members must first detect the existence of a partition, and then repair it by adding at least one virtual link to reconnect the mesh. Members on each side of the partition stop receiving sequence number updates from members on the other side . This condition is detected by a timeout of duration T_m.

Each member maintains a queue of members that it has

```
EvaluateUtility (j) begin
utility = 0
for each member m (m not i) begin
    CL = current latency between i and m along mesh
    NL = new latency between i and m along mesh
                    if edge i-j were added
      if (NL < CL) then begin
          utility += (CL-NL)/CL
      end
 end
 return utility
```

Figure 5: Algorithm i uses in determining utility of adding link to j

stopped receiving sequence number updates from for at least T_m time. It runs a scheduling algorithm that periodically and probabilistically deletes a member from the head of the queue. The deleted member is probed and it is either determined to be dead, or a link is added to it. The scheduling algorithm is adjusted so that no entry remains in the queue for more than a bounded period of time. Further, the probability value is chosen carefully so that in spite of several members simultaneously attempting to repair partition only a small number of new links are added. The algorithm is summarized in Figure 4.

3.2 Improving mesh quality

The constructed mesh can be quite sub-optimal, because (i) initial neighbor selection by a member joining the group is random given limited availability of topology information at bootstrap; (ii) partition repair might aggressively add edges that are essential for the moment but not useful in the long run; (iii) group membership may change due to dynamic join and leave; and (iv) underlying network conditions, routing and load may vary. Narada allows for incremental improvement of mesh quality. Members probe each other at random and new links may be added depending on the perceived gain in *utility* in doing so. Further, members continuously monitor the utility of existing links, and drop links perceived as not useful. This dynamic adding and dropping of links in the mesh distinguishes Narada from other topology maintenance protocols.

The issue then is the design of a utility function that reflects mesh quality. A good quality mesh must ensure that the shortest path delay between any pair of members along the mesh is comparable to the unicast delay between them. A member i computes the utility gain if a link is added to member j based on (i) the number of members to which j improves the routing delay of i; and (ii) how significant this improvement in delay is. Figure 5 presents pseudo code that i uses to compute the gain in utility if a link to member j is added. The utility can take a maximum value of n, where n is the number of group members i is aware of. Each member m can contribute a maximum of 1 to the utility, the actual contribution being $i's$ relative decrease in delay to m if the edge to j were added.

We now present details of how Narada adds and removes links from the mesh.

Addition of links: Narada requires every member to constantly probe other members. Currently, the algorithm that we use is to conduct a probe periodically, and probe some random member each time. This algorithm could be made smarter by varying the interval between probes depending on how satisfied a member is with the performance of the mesh, as well as choosing whom to probe based on results

```
EvaluateConsensusCost(j) begin
 Cost_ij = number of members for which i uses j as
           next hop for forwarding packets.
 Cost_ji = number of members for which j uses i as
           next hop for forwarding packets.
 return max(Cost_ij, Cost_ji)
end
```

Figure 6: Algorithm i uses to determine consensus cost to a neighbor j

of previous probes.

When a member i probes a member j, j returns to i a copy of its routing table. i uses this information to compute the expected gain in utility if a link to j is added as described in Figure 5. i decides to add a link to j if the expected utility gain exceeds a given threshold. The threshold value is a function of $i's$ estimation of group size, and the current and maximum fanout values of i and j respectively. Finally, i may also add a link to j if the physical delay between them is very low and the current overlay delay between them very high.

Dropping of links: Ideally, the loss in utility if a link were to be dropped must exactly equal the gain in utility if the same link were immediately re-added. However, this requires estimating the relative increase in delay to a member if a link were dropped and it is difficult to obtain such information. Instead, we overestimate the actual utility of a link by its *cost*. The cost of a link between i and j in $i's$ perception is the number of group members for which i uses j as next hop. Periodically, a member computes the *consensus cost* of its link to every neighbor using the algorithm shown in Figure 6. It then picks the neighbor with lowest consensus cost and drops it if the consensus cost falls below a certain threshold. The threshold is again computed as a function of the member's estimation of group size and its current and maximum fanout. The consensus cost of a link represents the maximum of the cost of the link in each neighbor's perception. Yet, it might be computed locally as the mesh runs a distance vector algorithm with path information.

Our heuristics for link-dropping have the following desirable properties:

• *Stability:* A link that Narada drops is unlikely to be added again immediately. This is ensured by several factors: (i) the threshold for dropping a link is less than or equal to the threshold for adding a link; (ii) the utility of an existing link is overestimated by the cost metric; (iii) dropping of links is done considering the perception that both members have regarding link cost; (iv) a link with small delay is not dropped.

•*Partition avoidance:* We present an informal argument as to why our link dropping algorithm does not cause a partition assuming steady state conditions and assuming multiple links are not dropped concurrently. Assume that member i drops neighbor j. This could result in at most two partitions. Assume the size of $i's$ partition is S_i and the size of $j's$ partition is S_j. Further, assume both i and j know all members currently in the group. Then, the sum of S_i and S_j is the size of the group. Thus $Cost_{ij}$ must be at least S_j and $Cost_{ji}$ must be at least S_i, and at least one of these must exceed half the group size. As long as the drop threshold is lower than half the group size, the edge will not be dropped.

3.3 Data Delivery

We have described how Narada constructs a mesh among participating group members, how it keeps it connected, and how it keeps refining the mesh. In this section we explain how Narada builds data delivery tree.

Narada runs a distance vector protocol on top of the mesh. In order to avoid the well-known count-to-infinity problems, it employs a strategy similar to BGP [16]. Each member not only maintains the routing cost to every other member, but also maintains the path that leads to such a cost. Further, routing updates between neighbors contains both the cost to the destination and the path that leads to such a cost. The per-source trees used for data delivery are constructed from the reverse shortest path between each recipient and the source, in identical fashion to DVMRP [4]. A member M that receives a packet from source S through a neighbor N forwards the packet only if N is the next hop on the shortest path from M to S. Further, M forwards the packet to all its neighbors who use M as the next hop to reach S.

The routing metric used in the distance vector protocol is the latency between neighbors. Each endpoint of a link independently estimates the latency of the link and could have different estimates. Using the latency as a metric enables routing to adapt to dynamics in the underlying network. However, it also increases the probability of routing instability and oscillations. In our work, we assume that members use an exponential smoothing algorithm to measure latency. Further, the latency estimate is updated only at periodic intervals. The period length can be varied to tradeoff routing stability with reactivity to changing conditions.

A consequence of running a routing algorithm for data delivery is that there could be packet loss during transient conditions when member routing tables have not yet converged. In particular, there could be packet loss when a member leaves the group or when a link is dropped for performance reasons. To avoid this, data continues to be forwarded along old routes for enough time until routing tables converge. To achieve this, we introduce a new routing cost called *Transient Forward (TF)*. TF is guaranteed to be larger than the cost of a path with a valid route, but smaller than infinite cost. A member M that leaves advertises a cost of TF for all members for which it had a valid route. Normal distance vector operations leads to members choosing alternate valid routes not involving M (as TF is guaranteed to be larger than the cost of any valid route). The leaving member continues to forward packets until it is no longer used by any neighbor as a next hop to reach any member, or until a certain time period expires.

4. SIMULATION EXPERIMENTS

In this section, we evaluate the properties of the overlay structure that Narada produces, and the overheads associated with the Narada protocol.

4.1 Performance Indices

An overlay structure fundamentally cannot perform as efficiently as IP Multicast. We are interested in evaluating the quality of the structure produced by Narada and in comparing it with two alternate methods of data dissemination, IP Multicast using DVMRP[4] and naive unicast. To facilitate our comparison, we consider the following metrics:

• *Relative Delay Penalty (RDP)* , defined in Section 2, which is a measure of the increase in delay that applications perceive while using Narada.

• *Worst Case Stress*, defined as $\max_{i=1}^{L} s_i$, where L is the number of physical links used in transmission and s_i is the stress (Section 2) of link i. This metric measures the effectiveness of Narada in distributing network load across physical links.

• *Normalized Resource Usage (NRU)*, defined as the ratio of the resource usage (Section 2) of Narada relative to resource usage of DVMRP. NRU is a measure of the additional network resources consumed by Narada compared to IP Multicast.

DVMRP has an RDP of 1 (assuming symmetric routing), a worst case stress of 1 and by definition an NRU of 1. Naive unicast has an RDP of 1 (by definition) and a worst case stress of r, when r is the number of receivers.

We also evaluate the time it takes for the overlay to stabilize and the protocol overhead that Narada introduces. In this paper, we do not consider performance metrics related to behavior under transient conditions, such as packet loss.

4.2 Factors that affect Narada's Performance

We have investigated the effects of the following factors on Narada's performance: (i) topology model; (ii) topology size; (iii) group size and (iv) fanout range.

We used three different models to generate backbone topologies for our simulation. For each model of the backbone, we modeled members as being attached directly to the backbone topology. Each member was attached to a random router, and was assigned a random delay of $1-4ms$.

• *Waxman:* The model considers a set of n vertices on a square in the plane and places an edge between two points with a probability of $\alpha e^{\frac{-d}{\beta * L}}$, where, d is the distance between vertices , L is the length of the longest possible edge and α and β are parameters. We use the Georgia Tech. [23] random graph generators to generate topologies of this model.

• *Mapnet:* Backbone connectivity and delay are modeled after actual ISP backbones that could span multiple continents. Connectivity information is obtained from the CAIDA Mapnet project database [7]. Link delays are assigned based on geographical distance between nodes.

• *Automous System map (ASMap):* Backbone connectivity information is modeled after inter-domain Internet connectivity. This information is collected by a route server from BGP routing tables of multiple geographically distributed routers with BGP connections to the server [6]. This data has been analyzed in [5] and has been shown to satisfy certain power laws. Random link delays of $8-12$ ms was assigned to each physical link.

In our simulations, we used backbone topology sizes consisting of up to 1070 members and multicast groups of up to 256 members. The fanout range of a member is the minimum and maximum number of neighbors each member strives to maintain in the mesh. An increase of the fanout range could decrease mesh diameter and result in lower delay penalties. However, it could potentially result in higher stress on links near members.

In addition, we identify network routing policy and group distribution as factors that could impact Narada's performance but do not investigate these in this paper. Routing policy could be significant because in the event that routing is not based on shortest path, some pairs of members could have an RDP of less than 1 with Narada. Group distribution is important as presence of clusters in groups could improve Narada's performance compared to unicast. This is because Narada could minimize the number of copies of a packet that enter a cluster via costlier inter-cluster links and distribute them along cheaper intra-cluster links.

4.3 Simulation Setup

We use a locally written, packet-level, event-based simulator to evaluate our protocol. The simulator assumes shortest delay routing between any two members. The simulator models the propagation delay of physical links but does not model queueing delay and packet losses. This was done to make our simulations more scalable. To consider dynamic network conditions we are currently conducting a detailed evaluation of Narada on the Internet and we present preliminary results in Section 5.

All experiments we report here are conducted in the following manner. A fixed number of members join the group in the first 100 seconds of the simulation in random sequence. A member that joins is assumed to contain a list of all members that joined the group previously. After 100 seconds, there is no further change in group membership. One sender is chosen at random to multicast data at a constant rate. We allow the simulation to run for 40 minutes. In all experiments, neighbors exchanges routing messages every 30 seconds. Each member probes one random group member every 10 seconds to evaluate performance.

4.4 Simulation Methodology

We do not adopt a full factorial design that investigates every possible combination of all factors. Instead we study the influence of each individual factor on Narada's performance one at a time, keeping other factors fixed.

We begin by presenting results from a typical experiment that characterizes key aspects of Narada's performance in Section 4.5. In Section 4.6, we present results that investigate the influence of the factors on Narada's performance. We present protocol overhead incurred with Narada in Section 4.7. Finally, we summarize and interpret our results in Section 4.8.

4.5 Simulation Results with a Typical Run

This section presents results from a single typical experiment. The results are typical in the sense they capture some of the key invariants in the performance of Narada across all runs. In the experiment, we used a topology generated by the Waxman model consisting of 1024 nodes and 3145 links. We used a group size of 128 members, and each member had a fanout range of <3-6>.

Delay Penalty and Stabilization Time

Figure 7 plots the cumulative distribution of RDP at different time instances during the simulation. The horizontal axis represents a given value of RDP and the vertical axis represents the percentage of pairs of group members for which the RDP was less than this value. Each curve corresponds to the cumulative distribution at a particular time instance. It might happen that at a given time, some members have not yet learned of the existence of some other

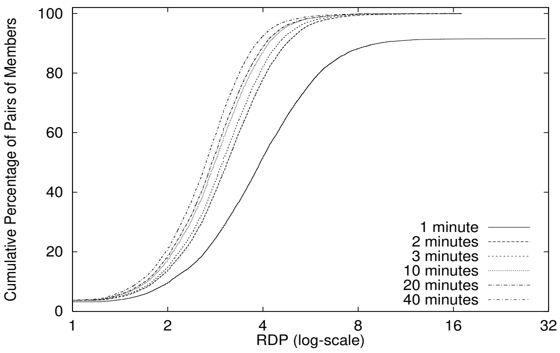

Figure 7: Cumulative distribution of RDP shown at various snapshots of the simulation. The minutes denote the time after the last join.

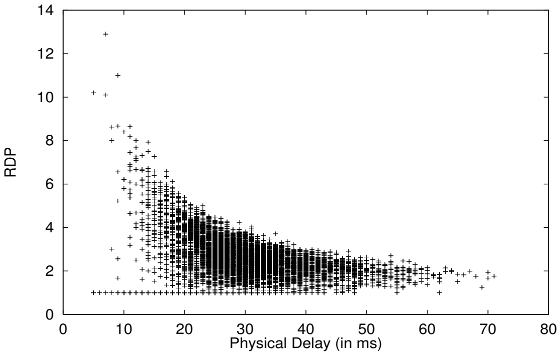

Figure 8: RDP vs. physical delay. Each point denotes the existence of a pair of members with a given physical delay and RDP

members or do not have routes to others. Thus, 1 minute after the last join, approximately 10% of pairs do not have routes to each other, indicated by the lower curve. All pairs have routes to each other 2 minutes after the last join. As time increases, the curve moves to the left, indicating the RDP is reduced as the quality of the overlay improves.

When the system stabilizes, about 90% of pairs of members have RDP less than 4. However, there exist a few pairs of members with high RDP. This tail can be explained from Figure 8. Each dot in this figure indicates the existence of a pair of members with a given RDP and physical delay. We observe that all pairs of members with high RDP have very small physical delays. Such members are so close to each other in the physical network that even a slightly sub-optimal configuration leads to a high delay penalty. However, the delay between them along the overlay is not too high. This can be seen from Figure 9, where each point represents the existence of a pair of members with a given overlay delay and a given physical delay. It may be verified that the delay between all pairs of members along the overlay is at most $160ms$, while the physical delay can be as high as $71ms$.

In future experiments, we summarize RDP results of an experiment by the *90 percentile RDP* value. We believe this is an appropriate method of summarizing results because: (i) it is an upper bound on the RDP observed by 90% of pairs of members; (ii) for pairs of members with a RDP value higher than the *90 percentile*, the overlay delay is small as discussed

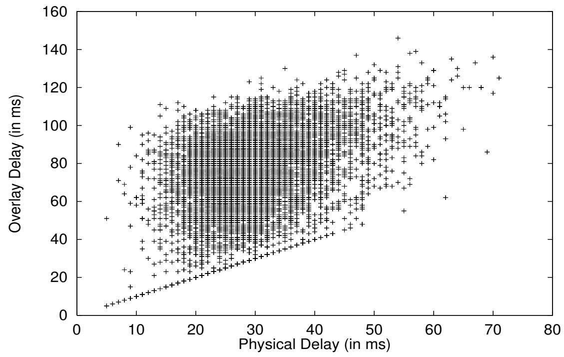

Figure 9: Overlay delay vs. physical delay. Each point denotes the existence of a pair of members with a given physical delay and overlay delay

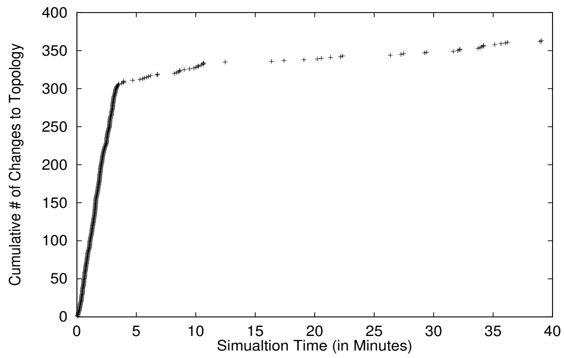

Figure 10: Cumulative number of virtual links added and removed vs. time

in the previous paragraph; and (iii) it is fairly insensitive to particular experiment parameters, unlike the omitted tail

Figure 10 plots the cumulative number of virtual links added and removed from the mesh as a function of simulation time. We observe that most of the changes happen within the first 4 minutes of the simulation. This is consistent with the behavior seen in Figure 7 and indicates that the mesh quickly stabilizes into a good structure.

Physical Link Stress

We study the variation of physical link stress under Narada and compare the results we obtain for a typical run with physical stress under DVMRP and naive unicast in Figure 11. One of the members is picked as source at random, and we evaluate the stress of each physical link. Here, the horizontal axis represents stress and the vertical axis represents the number of physical links with a given stress. The stress of any physical link is at most 1 for DVMRP, indicated by a solitary dot. Under both naive unicast and Narada, most links have a small stress - this is only to be expected. However, the significance lies in the tail of the plots. Under naive unicast, one link has a stress of 127 and quite a few links have a stress above 16. This is unsurprising considering that links near the source are likely to experience high stress. Narada however distributes the stress more evenly across physical links, and no physical link has a stress larger than 9. While this is high compared to DVMRP, it is a 14-fold improvement over naive unicast.

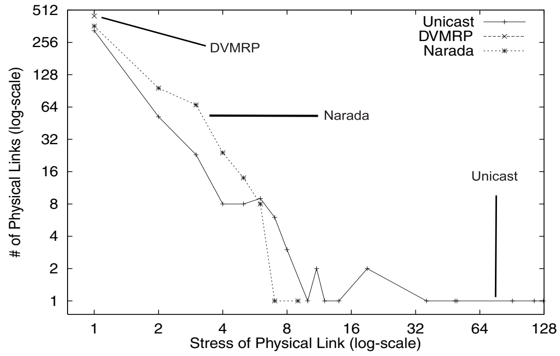

Figure 11: No. of physical links with a given stress vs. Stress for naive unicast, Narada and DVMRP

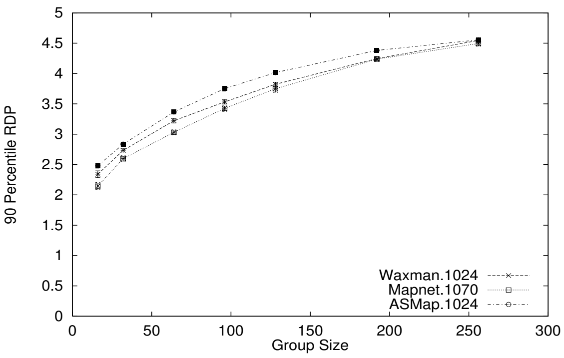

Figure 12: 90 percentile RDP vs. group size for topologies from three models

4.6 Impact of factors on performance

We are interested in studying variation in Narada's performance due to each of the following factors: (i) topology model; (ii) topology size; (iii) group size; and (iv) fanout range. Keeping other factors fixed at the default, we study the influence of each individual factor on Narada's performance. By default, we used a Waxman topology with 1024 nodes and 3145 links, a group size of 128 and a fanout range of <3-6> for all group members. For all results in this section, we compute each data point by conducting 25 independent simulation experiments and we plot the mean with 95% confidence intervals. Due to space constraints, we present plots of selected experiments and summarize results of other experiments.

Topology Model and Group Size

We used a Waxman topology consisting of 1024 routers and 3145 links, an ASMap topology consisting of 1024 routers and 3037 links and a Mapnet topology consisting of 1070 routers and 3170 links.

Figure 12 plots the variation of the 90 percentile RDP with group size for three topologies. Each curve corresponds to one topology. All the curves are close to each other indicating that the RDP is not sensitive to the choice of the topology model. For all topologies and for a group size of 128 members, the 90 percentile RDP is less than 4. For each topology, the 90 percentile RDP increases with group size. This is because an increase of group size results in an increase of mesh diameter and hence an increase of RDP.

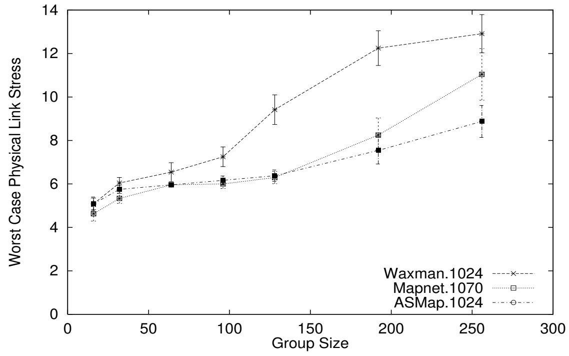

Figure 13: Worst case physical link stress vs. group size for topologies from three models

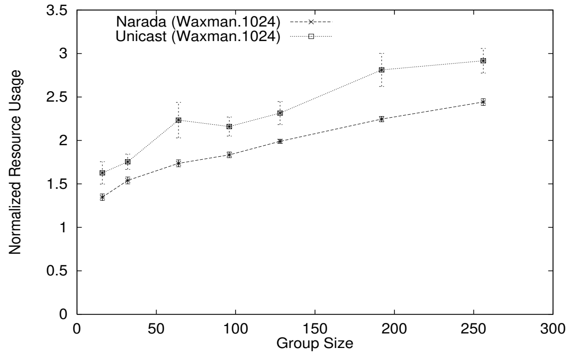

Figure 14: Effect of group size on NRU : Narada vs. naive unicast

Figure 13 plots the variation of worst case physical link stress against group size for three topologies. Each curve corresponds to one topology. We observe that the curves are close to each other for small group sizes but seem to diverge for larger group sizes. Further, for all topologies, worst case stress increases with group size. Thus, for a group size of 64, mean worst case stress is about $5-7$ across the three topologies, while for a group size of 256, it is about $8-14$. We believe this increase of stress with group size is an artifact of the small topologies in a simulation environment relative to the actual Internet backbone. We analyze this in detail in Section 4.8.

Figure 14 plots the normalized resource usage (NRU) against group size for the Waxman model alone. The lower and upper curves correspond to Narada and unicast respectively. First, Narada consumes less network resources than naive unicast, and this is consistent for all group sizes. For a group size of 128, the NRU for Narada is about 1.8 and 2.2 for naive unicast. Second, NRU increases with group size. While these results imply a nearly 20% savings of network resources, we believe that the savings could be even more significant if members are clustered. We have repeated this study with the Mapnet and ASMap topologies and observe similar trends. For all topologies, the NRU is at most 1.8 for a group size of 128.

Topology Size

For each topology model, we generate topologies of sizes varying from about 64 nodes to about 1070 nodes and evaluate the impact on Narada's performance. Figure 15 plots the worst case physical link stress against topology size for

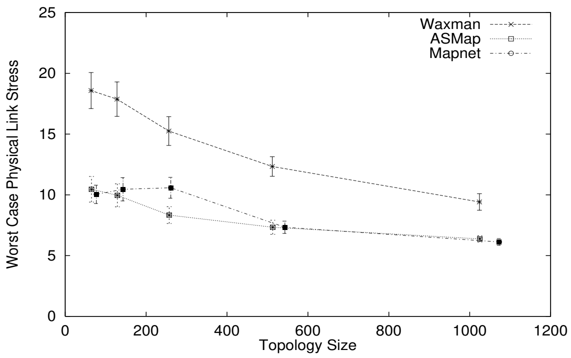

Figure 15: Worst case physical link stress vs. topology size for topologies from three models

each topology model. Across all topology models, we observe that the worst case stress increases with decrease in topology size. While the same general trend is observed for all topology models, it seems more pronounced for Waxman. We analyze the significance of this result in Section 4.8.

We have also studied the effect of topology size on RDP and NRU. Across all topology models, RDP appears largely unaffected by topology size, while NRU decreases with increase in topology size. We omit the plots due to space constraint.

Fanout Range

So far, we have assumed that each member strives to maintain <3-6> neighbors in the mesh. We have investigated the effect of variation of fanout range on Narada's performance. In summary, when the fanout range increases, mesh diameter decreases and stress on links close to members increases. Consequently, RDP decreases while worst case stress increases. For a group of 128 members, as fanout range increases from <2-4> to <8-16>, the 90 percentile RDP decreases from about 5.5 to 2 while the worst case physical stress increases from about 9 to 15.

4.7 Protocol Overhead

Narada incurs a protocol overhead for two reasons. First, members periodically exchange routing tables and control information between each other. Second, members estimate their delays to other members by probing them periodically. We define *Protocol Overhead Ratio (POR)* as the ratio of bytes of non-data traffic that enter the network to bytes of data traffic. While we do not present results, we find that POR increases linearly with group size. Further, we note that the protocol traffic that Narada introduces is independent of source data rate and thus the POR decreases with increase in data traffic. For a group size of 128 members, the POR is about 0.25 for a source data rate of 16 kilobits per second (kbps), and less than 0.04 for a source data rate of 128 kbps. For a 64 member group and a source data rate of 128 kbps, the POR is hardly 0.02.

4.8 Results Summary and Interpretation

In this section, we summarize key results that we have presented and attempt to explain the results.

• Across a range of topology models, Narada results in a low RDP for small group sizes. For example, for a group size of 16, the 90 percentile RDP is less than 2.5. Even for group sizes of 128 members, the 90 percentile RDP is less than 4. We hypothesize that RDP values might be lower on the Internet, as Internet routing is policy based and sub-optimal while the simulator assumes shortest path routing. Preliminary Internet evaluation indicates that the 90 percentile RDP for a 13 member group can be as low as 1.5 (Section 5).

• Across a range of topology models, Narada results in a low worst case stress for small group sizes. For example, for a group size of 16, the worst case stress is about 5. While for larger group sizes, worst case stress may be higher, it is still much lower than unicast. For example, for a group of 128 members, Narada reduces worst case stress by a factor of 14 compared to unicast.

We hypothesize that worst case stress on the Internet is lower than seen in simulations. The largest topologies that we use in our simulations (around 1000 nodes) are still orders of magnitude smaller than the Internet. Consequently, the ratio of the group size to topology size, which we term *density*, is much higher in simulations than in actual practice. Our simulations indicate that higher group density results in higher worst case link stress. This can be deduced from Figures 13 and 15, where we observe that the worst case stress increases with group size and decreases with topology size. We hypothesize that an increase in group density increases the probability that an internal physical link could be shared by multiple uncorrelated virtual links. The links are uncorrelated in the sense that they connect distinct pairs of end systems.[2] This could increase worst case stress with Narada because Narada is only able to regulate fanout of members and consequently can only control stress of links near member and not stress of internal physical links. For the range of group sizes we consider, we expect that the density ratio is much lower on the Internet and thus we expect lower link stress.

• Narada lowers resource usage by at least 20% compared to unicast for a range of group sizes. We believe that if members are clustered, Narada can result in even larger improvement in resource usage.

5. INTERNET EXPERIMENTS

We have implemented Narada and we report preliminary results obtained by conducting experiments on the Internet.

Our experiments consisted of 13 hosts distributed throughout the United States. In each experiment, every host was initially provided the list of names of all other hosts, and all hosts join the group at approximately the same time. Each host attempted to maintain a fanout range of <2-4>. A host at CMU was designated the source and sent data at periodic intervals. At present, we assume that Narada attempts to minimize the propagation delay between hosts on the mesh. Thus, each host assumes its physical delay to another host is the minimum delay observed across multiple estimates. Each experiment was run for about 20 minutes.

Figure 16 shows the overlay spanning tree that Narada produced, which was used to route data from the source (CMU1) to other recipients, for a typical experiment. The links of the tree are labeled by their delays (in milliseconds), as mea-

[2] For example, consider the physical topology shown in Figure 1(a) and assume that Narada constructs a mesh as shown in Figure 1(h). Uncorrelated virtual links $A-C$ and $B-D$ share the physical link $R1-R2$.

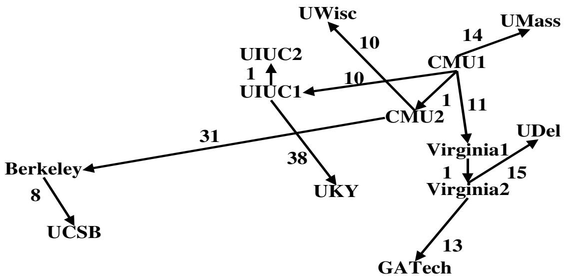

Figure 16: Overlay spanning tree Narada produces in a typical Internet experiment with CMU1 as source. Note that it produces a mesh, and edges of the mesh not in the tree are omitted.

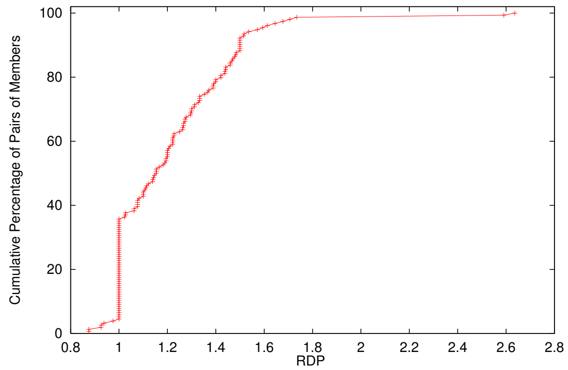

Figure 17: Cumulative distribution of RDP

sured by one end point of the link. Here, UIUC1 and UIUC2 belong to the same campus as do Virginia1 and Virginia2. Berkeley and UCSB are closer together (on the West coast) as compared to all other hosts. Narada is able to ensure that only a single copy of data from CMU1 is delivered to UIUC, Virginia and the West Coast and shorter links here are used for distribution of data within the sites.

Narada in fact constructs a mesh, and there are additional edges of the mesh not in the tree which we have omitted in Figure 16. Further, Narada may dynamically add and drops links to improve mesh quality. In this experiment, the mesh did not have links between Berkeley and UCSB, and between UIUC1 and UIUC2 in the first few minutes of the experiment. The self-improving nature of Narada resulted in addition of these links and improved the efficiency of data delivery. Narada was also able to drop a link between GATech and Delaware which it identified as not useful in data delivery.

Figure 17 plots the cumulative distribution of the RDP of the mesh that Narada produced. The worst case RDP is only 2.6 and 90% of pairs of members have an RDP of at most 1.5. Further, we find that the delay along the mesh between any pair of members is at most *84 ms* , while the worst case unicast delay between any pair of members can be as high as *64 ms* (observed between UKY and UCSB). The physical delay used in RDP calculations were the delays estimated by an end point of the link. End points independently estimate link delay and thus might have different estimates. In our experiments however, we find that link delay estimates by end points are generally consistent with each other and the estimates are very close to the ping times between the machines. Pairs with an RDP of 1 correspond to pairs of hosts with mesh links between them. Interestingly, we find a small number of pairs for which the RDP is slightly less than 1. There are two reasons for this effect. In some cases, this was due to minor inaccuracies in delay measurements. However, in some cases, the reason was more fundamental and due to the policy based nature of Internet routing. This effect has been reported in [18].

In the future we plan to conduct larger scale Internet experiments with emphasis on studying the dynamics of Narada, transient behavior and effects of congestion and packet loss.

6. RELATED WORK

The works that come closest to the End System Multicast architecture we propose here and which share much of our motivation are Yallcast [8] and Scattercast [3]. Both projects challenge the appropriateness of using IP Multicast for all forms of group communication. Yallcast argues the need for hosts to auto-configure into tunneled topologies for efficient data dissemination. Yallcast emphasizes architectural aspects and has not yet considered performance implications of using an overlay. In contrast, our work places a central emphasis on performance tradeoffs associated with an overlay network, and this greatly influences the design of our self-organization protocol. Yallcast presents the design of a rendezvous service for the bootstrapping of a new member. The bootstrapping mechanism is however orthogonal to the issues we consider in this paper - while the mechanisms suggested in Yallcast could be used, we are open to other application specific and out-of-band bootstrap mechanisms. At the protocol level too, Yallcast and Narada have differences. Narada constructs an efficient mesh among participating members, and then constructs spanning trees of the mesh. In contrast, Yallcast constructs a spanning tree among participating members directly. We believe that a mesh-first approach helps avoid replicating group management functions across multiple (per-source) trees, simplifies overlay maintenance (as loop avoidance is no longer an issue), allows for leveraging on standard routing algorithms, and provides a structure more resilient to the failure of members.

Scattercast argues for infrastructure support, where proxies deployed in the network run self-organization protocols on behalf of the clients, while clients subscribe to nearby proxies. Although we have not emphasized infrastructure support in this paper, we wish to make explicit that our notion of an end system is not restricted to clients of a multicast group, and includes network proxies. In particular, we believe that Narada can be used to build an overlay structure among proxies on a scattercast architecture, while the self-organization protocol proposed in Scattercast called Gossamer can be adapted to an End System Multicast architecture. At the protocol level, while Gossamer too adopts a mesh-first approach, it relies on centralized rendezvous points for repairing mesh partition. Although this assumption significantly simplifies the partition recovery mechanisms in Gossamer, the members of the mesh could become partitioned from each other in event of failure of all rendezvous points.

The MBone [2] and 6Bone [10] are popular existing examples of overlay networks. However, these are statically configured in a manual and adhoc fashion. Narada, on the other hand, strives for a self-configuring and efficient overlay network. Internet routing protocols are self-configuring. The most striking difference between Narada and standard routing protocols is that while the latter work on a fixed physical topology, Narada alters the very topology over which it routes data. Routing protocols merely route around a link that has failed and have no notion of dynamic adding or dropping of links. Narada might dynamically add links to ensure connectivity of the virtual topology, and drop links it perceives as not useful.

Self-configuration has been proposed in other contexts. AMRoute [1] allows for robust IP Multicast in mobile adhoc networks by exploiting user-multicast trees. Several reliable IP Multicast protocols [12, 13, 22] involve group members self-organizing into structures that help in data recovery. Adaptive Web Caching [14] is a self-organizing cache hierarchy. The key feature that distinguishes Narada from these protocols is that Narada does not assume a native multicast medium - AMRoute assumes a native wireless broadcast channel, while all other protocols assume the existence of IP Multicast. Self-configuration in the absence of such a native multicast medium is a much harder problem.

7. CONCLUSIONS

We have made two contributions in this paper. First, we have shown that, for small and sparse multicast groups, it is feasible to use an end system overlay approach to *efficiently* and *robustly* support all multicast related functionality including membership management and packet replication. The shifting of multicast support from routers to end systems, while introducing some performance penalties, has the potential to address most problems associated with IP Multicast. We have shown, with both simulation and Internet experiments, that the performance penalties are low in the case of small and sparse groups. We believe that the potential benefits of repartitioning the multicast functionality between end systems and routers significantly outweigh the performance penalty incurred.

Second, we have proposed one of the first self-organizing and self-improving protocols that constructs an overlay network on top of a dynamic, unpredictable and heterogeneous Internet environment without relying on a native multicast medium. We also believe this is among the first works that attempt to systematically evaluate the performance of a self-organizing overlay network protocol and the tradeoffs in using overlay networks. In [20], it was argued that the overlay approach is a fundamental technique to incrementally deploy services and evolve networks. We believe that the techniques and insights developed in this paper are general and can be applied to overlay networks in contexts other than multicast.

In this paper, we emphasize the performance aspect of using an end system overlay approach to support multicast. We are currently extending this work in several dimensions. First, as mentioned in Section 1, the "end system" in the protocol can be either an application module, a host, or a shared proxy. We are exploring architectural issues involved in adapting Narada to each individual context. Second, in the current form, Narada considers latency between members as the sole criterion that needs to be optimized. We are extending this to consider measured packet loss rates and bandwidth availability. Finally, we are studying how support for error, flow, and congestion control functionality can be added in End System Multicast.

8. REFERENCES

[1] E. Bommaiah, A. McAuley, R. Talpade, and M. Liu. Amroute: Adhoc multicast routing protocol. Internet draft, Internet Engineering Task Force, August 1998.

[2] S. Casner and S. Deering. First IETF internet audiocast. *ACM Computer Communication Review*, pages 92–97, 1992.

[3] Y. Chawathe, S. McCanne, and E. A. Brewer. An architecture for internet content distribution as an infrastructure service, February 2000. Unpublished work.

[4] S. Deering. Multicast routing in internetworks and extended lans. In *Proceedings of the ACM SIGCOMM 88*, pages 55–64, Stanford,CA, August 1988.

[5] C. Faloutsos, M. Faloutsos, and P. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of ACM Sigcomm*, August 1999.

[6] National Laboratory for Applied Network Research. Routing data. http://moat.nlanr.net/Routing/rawdata/.

[7] Cooperative Association for Internet Data Analysis. Mapnet project. http://www.caida.org/Tools/Mapnet/Data/.

[8] P. Francis. Yallcast: Extending the internet multicast architecture, http://www.yallcast.com, September 1999.

[9] H.W.Holbrook and D.R. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proceedings of the ACM SIGCOMM 99*, August 1999.

[10] IPV6 backbone. http://www.6bone.org/.

[11] G. Kortsarz and D. Peleg. Generating low-degree 2-spanners. *SIAM Journal on Computing, Volume 27, Number 5*.

[12] B. N. Levine, D. B. Lavo, and J. J. Garcia-Luna-Aceves. The case for concurrent reliable multicasting using shared ACK trees. In *Proceedings of ACM Multimedia'96*, November 1996.

[13] J. Liebeherr and B. S. Sethi. A scalable control topology for multicast communications. In *Proceedings of IEEE Infocom*, April 1998.

[14] S. Michel, K. Nguyen, A. Rozenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive web caching: towards a new global caching architecture. *Computer Networks and ISDN Systems*, November 1998.

[15] R. Perlman, C. Lee, T. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green. Simple multicast: A design for simple, low-overhead multicast. Internet Draft, Internet Engineering Task Force, March 1999. Work in progress.

[16] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4), RFC 1771, March 1995.

[17] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):195–206, 1984.

[18] S. Savage, A. Collins, E. Hoffman, J.Snell, and T. Anderson. The end-to-end effects of internet path selection. In *Proceedings of ACM Sigcomm*, August 1999.

[19] F.B. Schneider. Byzantine generals in action: Implementing fail-stop processors. *ACM transactions on Computer Systems, 2(2)*, pages 145–154, 1984.

[20] K. Sripanidkulchai, A. Myers, and H. Zhang. A third-party value-added network service approach to reliable multicast. In *Proceedings of ACM Sigmetrics*, August 1999.

[21] I. Stoica, T.S.E. Ng, and H. Zhang. REUNITE: A recursive unicast approach to multicast. In *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.

[22] R. X. Xu, A. C. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *Proceedings of NOSSDAV'97*, May 1997.

[23] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom*, March 1996.