A Protocol for Scalable Application Layer Multicast

Suman Banerjee, Bobby Bhattacharjee, Srinivasan Parthasarathy
Department of Computer Science, University of Maryland, College Park, MD 20742
{suman,bobby,sri}@cs.umd.edu

CS-TR 4278 and UMIACS-TR 2001-58

Abstract

We describe a new application-layer multicast protocol that is specifically designed to scale to large groups. Our scheme is based upon a hierarchical clustering of the application-layer multicast peers and can be used to produce a number of different data delivery trees with specific properties. On average, group members using our protocol maintain only a constant amount of state about other group members, and incur a constant amount of control overhead. We present extensive simulations of both our protocol and the Narada protocol over Internet-like topologies. Our results show that for groups of size 32 or more, we reduce control overhead by orders of magnitude, and link stress by 25%, while retaining similar end-to-end latencies and failure recovery properties.

1 Introduction

IP multicast is not widely deployed over a decade after it was developed [8]. Since the barriers for network layer deployment are not necessarily all technical, it is not clear when global IP multicast deployment will become a reality. It is, however, clear that multicast itself is a useful network service for many applications, including media-streaming and bulk reliable file transfer. A set of recent proposals [7, 9, 4, 11, 12] have proposed alternate architectures in which all multicast functionality is implemented at the end-hosts, thus obviating the need to change the unicast IP infrastructure. In all of these *Application-Layer Multicast* schemes, the set of end-hosts that are part of a logical multicast group participate in a distributed protocol that is used to implement the multicasting functionality.

The basic idea of application-layer multicast is shown in Figure 1. Unlike native multicast where data packets are replicated at routers inside the network, in application-layer multicastdata packets are replicated at end hosts. Logically, the end-hosts form an overlay network, and the goal of application-layer multicast is to construct and maintain an efficient overlay for data transmission. Since application-layer multicast protocols must send the identical packets over the same link, they are less efficient than native multicast. Two intuitive measures of the "goodness" of the overlay were defined in [7]: the *stress* and the *stretch*. The stress metric is defined per-link and counts the number of identical packets sent by a protocol over each underlying link in the network. The stretch metric is defined per-member and is the ratio of path-length from the source to a member versus the length of the unicast shortest path. Consider an application-layer multicast protocol in which the data source unicasts the data to each receiver. Clearly, this "multi-unicast" protocol minimizes stretch, but at a cost of $O(N)$ stress at links near the source (N is the number of group members) and a ($O(N)$control overhead at some single point. However, this protocol is robust in the sense that any number of group member failures do not affect the other members in the group.

All the application-layer multicast protocols discussed in this paper, including ours, are purely end-to-end, i.e. they are oblivious to the underlying network topology. It is, in fact, impossible to generate the least-stress application-

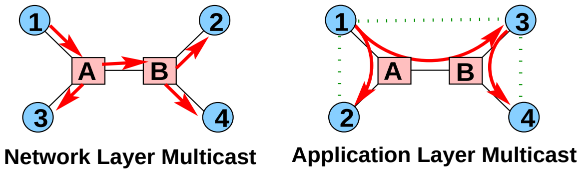

Figure 1: Network-layer and application layer multicast. Square nodes are routers, and circular nodes are end-hosts. The dotted lines represent peers on the overlay.

layer multicast tree without knowing the underlying topology. In general, application-layer multicast protocols can be evaluated along three dimensions:

- Quality of the data delivery path. The quality of the tree is measured using topological metrics such as stress, stretch, and node degrees.
- Robustness of the overlay. Since end-hosts are potentially less stable than routers, it is important for application-layer multicast protocols to mitigate the effect of receiver failures. The robustness of application-layer multicast protocols is measured by quantifying the extent of the disruption in data delivery when different members fail, and the time it takes for the overlay to restore delivery to the other members. None of prior published work evaluates the robustness of the overlay during the members failure, and we present the first comparison of this aspect of the application-layer multicast protocols.
- Control overhead. Since application-layer multicast is inherently a cooperative venture, it is important to design a protocol with low overhead such that the cost of participating in an application-layer multicast group justifies the benefits.

1.1 Existing Approaches

A number of different application-layer multicast schemes have been proposed in recent literature. They can be classified into two broad categories: tree-first approaches and mesh-first approaches. In the tree-first approach, members directly construct an overlay tree topology for data delivery, and additional control links are monitored and maintained to allow quick recovery from member failures. Yoid [9] and ALMI [12] are examples of the tree-first approach. As the name suggests, in the mesh-first approach, members distributedly construct a mesh (on overlay members in which multiple paths exist between pairs of members). Each member then paricipates in a routing protocol on the mesh topology, and generates a source-specific tree to all other members. Narada [7], and Gossamer [4] are examples of the mesh-first approach.

None of these end-to-end distributed approaches can produce a topology with bounded stretch or stress. It is possible to do better if (and only if) the underlying topology is known. In [10], a *centralized* topology-aware tree-building algorithm is described in which the stretch between any pair of end-points is bounded by a constant factor. The tree degree of a member, can however, be unbounded. However, with a slight modification to this algorithm, it is possible simultaneously guarantee a *constant* degree bound for the members, and a $O(\log N)$ bound for the stretch.

1.2 NICE Trees

Our application-layer multicast protocol was developed in context of the NICE project (NICE is a recursive acronym for "NICE is the Internet Cooperative Environment"). NICE is a programming environment for co-operatively implementing distributed applications over the Internet. The NICE environment uses an underlying application-layer multicast protocol, which is the exclusive focus of this paper. In the rest of this paper, we refer to the NICE application-layer multicast protocol as simply the NICE protocol. Our goals for NICE were to develop a scalable distributed tree-building protocol which did not require any underlying topology information. Specifically, we wanted to reduce the worst-case state and control overhead at any member to $O(\log N)$, maintain a constant degree bound for the group members and approach the $O(\log N)$ stretch bound possible with the topology-aware centralized algorithm.

Unlike previous approaches, NICE is neither an exclusively tree- or mesh-first approach. We create a hierarchically-connected control topology which has higher connectivity than a tree. This control topology is equivalent to a mesh; however, the data delivery path is implicitly defined in the way the mesh is structured and no additional route computations are required. NICE is, therefore, a hybrid of the tree- and mesh-first approaches. Unlike the centralized topology-aware algorithm described in [10], it is not possible to bound pair-wise end-to-end stretch using a distributed algorithm like NICE. However, on our simulated topologies, for topologies of size 128 and higher, *all* of the pair-wise end-to-end paths were, in fact, bounded by the $O(\log N)$ stretch bound.

In our performance comparisons, we compare NICE using detailed simulations to the Narada protocol. Narada is the only protocol that has previously been studied using both simulation [7] and implementation [6]. Narada is a mesh-first protocol which creates source-specific trees and has an aggregate $O(N^2)$ control overhead.

A key contribution of our approach is the definition of a control structure with low aggregate overhead ($O(N)$) over which different data delivery trees can be built. On average, end hosts maintain state about a *constant* number of other members, while a few members maintain state about $O(\log N)$ other members. A scalable application-layer multicast protocol in which the overhead is independent of the tree size is likely to be useful for a number of applications, most notably media-streaming applications with large receiver sets. Compared to unicast solutions[1] now deployed in the Internet, an application-layer multicast solution such as Narada reduces the data overhead at the source from $O(N)$ to $O(1)$. However, we also need to ensure that the control overheads at each end host and the network routers, are low. In comparison, on average NICE reduces both data and control overhead to a constant and can be used to support applications that requre large application-layer multicast groups. For groups of size 64 or more, our simulations show that compared to Narada, NICE imposes orders of magnitude less control overhead, while producing trees with lower average stress and comparable stretch.

The rest of the paper is structured as follows: In Section 2, we present our general approach, explain how different delivery trees are built over NICE, and provide a few theoretical bounds about the NICE protocol. In Section 3, we present the packet-level details of the protocol. We evaulate the performance of NICE and present a detailed comparison with Narada in Section 4. We elaborate on related work in Section 5, and present conclusions in Section 6.

2 Solution Overview

The NICE protocol arranges the set of end hosts into a hierarchy; the basic operation of the protocol is to create and maintain the hierarchy. The hierarchy defines a default data tree since any host can be reached from any other by traversing *up* and *down* the levels in the hierarchy. The member hierarchy is also crucial for scalability, since most members are in the bottom of the hierarchy and only maintain state about a constant number of other members. The members at the very top of the hierarchy maintain (soft) state about $O(\log N)$ other members. Logically, each member keeps detailed state about other members that are *near* in the hierarchy, and only has limited knowledge about other members in the group. The hierarchical structure is also important for localizing the effect of member failures.

In this paper, we use end-to-end path length as the distance metric between hosts. The NICE protocol can also be

[1] For example, see http://www.real.com

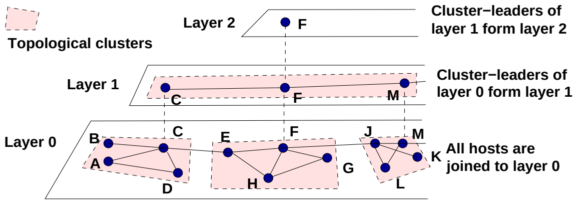

Figure 2: Hierarchical arrangement of hosts in NICE. The layers are logical entities overlaid on the same underlying physical network.

used with other metrics, e.g. latency or available bandwidth, but for the rest of this paper, we assume that distances refer to path lengths. While constructing this hierarchy, members that are "close" with respect to the distance metric are mapped to the same part of the hierarchy: this allows us to produce trees with low stretch. In the rest of this section, we describe how the NICE hierarchy is defined, what invariants it must maintain, and describe how it is used to establish scalable control and data paths.

2.1 A Hierarchical Arrangement of Group members

The NICE hierarchy is created by assigning members to different levels (or layers) as illustrated in Figure 2. Layers are numbered sequentially with the lowest layer of the hierarchy being layer zero (denoted by L_0). Hosts in each layer are partitioned into a set of clusters. Each cluster is of size between k and $2k-1$, where k is a constant, and consists of a set of hosts that are close to each other. Further, each cluster has a cluster leader. In our case, the cluster leader is the (graph-theoretic) center of the cluster, i.e. given a set of hosts in a cluster, the cluster leader has the minimum maximum distance to all other hosts in the cluster.

Hosts are mapped to layers using the following scheme: All hosts are part of the lowest layer, L_0. The clustering protocol at L_0 partitions these hosts into a set of clusters. The cluster leaders of all the clusters in layer L_i join layer L_{i+1}. This is shown with an example in Figure 2, using $k = 3$. The layer L_0 clusters are [ABCD], [EFGH] and [JKLM][2]. In this example, we assume that C, F and M are the centers of their respective clusters of their L_0 clusters, and are chosen to be the leaders. They form layer L_1 and are clustered to create the layer L_1 cluster, $[CFM]$. F is the center, and hence leader, of this cluster and belongs to layer L_2 as well.

The NICE clusters and layers are created using a distributed algorithm described in the next section; it is important to note that a separate instance of this clustering algorithm runs at each layer of the hierarchy. The following properties hold for the distribution of hosts in the different layers:

- A host belongs to only a single cluster at any layer.
- If a host is present in some cluster in layer L_i, it must occur in one cluster in each of the layers, $L_0, \ldots, L_{i-1}$. In fact, it is the cluster-leader in each of these lower layers.
- If a host is not present in layer, L_i, it cannot be present in any layer L_j, where $j > i$.
- There are at most $\log_k N$ layers, and the highest layer has only a single member.

We also define the term "super-cluster": for any host, H, it is the cluster to which its cluster leader belongs to. It follows that there is only one super-cluster defined for every host (except the host that belongs to the top-most

[2] We denote a cluster comprising of hosts $X, Y, Z, \ldots$ by $[XYZ\ldots]$.

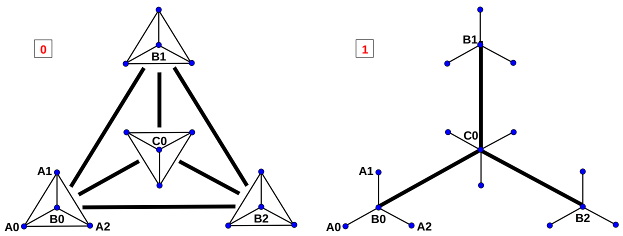

Figure 3: Different choices for control and data delivery paths for a two-layer hierarchy. All A_i hosts are members of only L_0 clusters. All B_i hosts are members of both layers L_0 and L_1. The only C host is the leader of the L_1 cluster comprising of itself and all the B hosts.

Procedure : $\mathit{MulticastDataForward}(h, p)$
  $\{\ h\ \in\ \ \mathit{layers}\ L_0, \ldots, L_i\ \mathit{in\ clusters}\ C_0, \ldots, C_i\ \}$
  for j in $[0, \ldots, i]$
  if $(p \notin C_j)$
    $\mathit{ForwardDataToSet}(C_j - \{h\})$
  end if
end for

Figure 4: Data forwarding operation at a host, h, that itself received the data from host p.

layer, it does not have a super-cluster), and the super-cluster is in the layer immediately above the highest layer that H belongs to. For example, in Figure 2, cluster $[CFM]$ in Layer 1 is the super-cluster for nodes B, A, and D. In NICE, each host maintains state about all the clusters it belongs to and about its super-cluster.

2.2 Control and Data Paths

The host hierarchy can be used to define different overlay structures for the control and data delivery paths. It is useful to have a structure with higher connectivity for the control messages, since this will cause the protocol to converge quicker.

In Figure 3, we illustrate two choices of control and data paths using clusters of size 4. Each set of four hosts arranged in a 4-clique in Panel 0 are the clusters in layer L_0. Hosts B_0, B_1, B_2 and C_0 are the cluster leaders of these four L_0 clusters. and form the single cluster in layer L_1. Host C_0 is the leader of this cluster in layer L_1. The clique corresponds to the highest possible connected structure for a cluster. In comparison, the least connected structure, a tree, is shown in Panel 1. Each cluster is organized as a star around the cluster leader. Other configurations, such as a ring or a balanced binary tree are also possible.

For our experiments, we chose the clique cluster configuration for the control path (Panel 0, Figure 3). This allows us to have a richer connectivity on the control path and also provides an upper bound on our control cost. Using this control path, in the worst case, each member peers with $O(k \log N)$ other members. The control path is only used for periodic soft state refreshes and updates and does not generate high volumes of traffic. For the data path, we chose the star connectivity within each cluster. This is the simplest data path in NICE, and is the worst case topology to demonstate the effect of host failures. The algorithm for multicast data forwarding on the topology

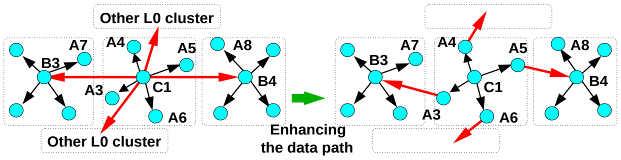

Figure 5: Data path enhancements using simple delegation.

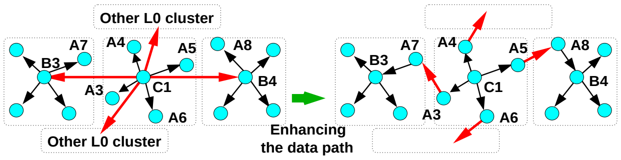

Figure 6: Data path enhancements using advanced delegation.

is as shown in Figure 4.

2.3 Analysis

Assume that each cluster in the hierarchy has exactly k hosts. Then for the control topology, a host that belongs only to layer L_0 peers with $k-1$ other hosts for exchange of control messages. In general, a host in layer L_i peers with $k-1$ other hosts in each of the layers $L_0, \ldots, L_i$. Therefore, the cluster-leader of the highest layer cluster (Host C_0 in Figure 3), will peer with a total of $(k-1)\log N$ neighbors. Using amortized analysis, it follows that on average, each host peers with only $O(k)$ neighbors.

For the data path, each host in layer L_0 peers with only one other host — its cluster-leader in the respective L_0 cluster. However, analogous to the control topology, a host that occurs in layer L_i, and no other higher layer, peers with $k-1$ hosts in each layer $L_0, \ldots, L_{i-1}$, and with one host in layer L_i (its cluster-leader). Thus, while the average host on the data path peers with ≤ 2 other hosts on the data path (again via amortized analysis), in the worst case a host would peer with $O(k \log N)$ other members. This is the case for the cluster-leader of the highest cluster. While an $O(k \log N)$ upper-bound is acceptable for the control topology, for high data rates, hosts may be unable or unwilling to forward data to $O(k \log N)$ other hosts. As we explain next, it is possible to reduce the per-host data overhead to a constant.

2.4 Enhancing Data Paths

The basic data path in NICE routes data packets up and down the hierarchy like any hierarchical routing protocol. We define an enhancement to this basic data path by allowing the cluster leaders to *delegate* data forwarding responsibility to some of its cluster members in a deterministic manner. The basic data path transformation is illustrated with an example in Figure 6. Host C_1 is the leader of an L_0 cluster, $[C_1, A_3, A_4, A_5, A_6]$ and a leader of an L_1 cluster, $[C_1, B_3, B_4, \ldots]$. In the basic data path, it is responsible for forwarding data to all the other members in its two clusters. In the enhanced path, it uses *delegation* of data forwarding responsibility.

We define two different delegation schemes. They are:

Simple Delegation: Consider a host, H, that is part of all layers, $L_0, \ldots, L_i$, and no higher layer. Clearly, it is the cluster-leader in the layers $L_0, \ldots, L_{i-1}$ and not a leader in its L_i cluster. Let S_j denote the set of the other cluster members in the layer L_j cluster of H, where $0 \leq j < i$. In the previously described data path, H peers with all the hosts in $\cup_j S_j$ and also the cluster-leader of H in L_i. In the enhanced data path, H delegates the hosts in S_{j-1} to peer with the hosts in S_j. Since the cluster sizes are bounded between k and $2k-1$, each host in S_{j-1} is therefore required by H to peer with at most two hosts in S_j. H still continues to peer with the hosts in S_0. We illustrate this by an example in Figure 5.

Using the above notation, for host, C_1, the set $S_0 = \{A_3, A_4, A_5, A_6\}$ and set $S_1 = \{B_3, B_4, \ldots\}$. There it delegates its overlay data paths to the set S_1 members to the set S_0 members, as the two overlay links: $\langle A_3, B_3 \rangle$ and

$\langle A_5, B_4 \rangle$.

In the enhanced data path, each host that occurs only in L_0 peers with *at most* 3 hosts, and all other hosts peers with *at most* $2k+2$ other hosts.

Advanced Delegation: In the advanced delegation, the multicast data in the group logically flows through horizontally across the layer L_0 clusters. Each host, H,that belongs to all layers $L_0, \ldots, L_i$ and no higher layer, we define the sets S_j as before (in Simple Delegation). H forwards data to all members in S_0 as before. It also delegates peering relationship between members in sets S_j and S_{j-1}. However, in this case, these hosts in sets S_j and S_{j-1} further delegate these peering relationships down the hierarchy. Consequently, all the data paths on the hierarchy are through peers in the lowest layer L_0.

We state the above more formally as follows: Consider a host, x that is the cluster-leader of a cluster C_k, in layer L_k.(Note that x belongs to some cluster C_j, in each layer $L_j, j \leq k$.) We define $\mathcal{L}_k(x) = C_k$ and $\mathcal{L}_j(x) = \cup_{y \in \mathcal{L}_{j+1}(x)} \mathcal{L}_j(y)$. Thus for every peering relation of x, and y who are in layers $L_i, i > 0$ and $L_j, j > 0$ respectively on the control path, we pick two hosts, one each in $\mathcal{L}_i(x)$ and $\mathcal{L}_j(y)$ to be peers on the data path. If these hosts are chosen so that they are the closest pair among the two $\mathcal{L}$ sets, it helps in reducing the stretch on the data paths.

2.5 Invariants

All the properties described in the analysis hold as long as the hierarchy maintains its integrity. Thus, the objective of NICE protocol is to scalably maintain the host hierarchy as new members join and existing members depart. Specifically the protocol described in the next section maintains the following set of invariants:

- At every layer, hosts are paritioned into clusters of size between k and $2k-1$.
- All hosts belong to an L_0 cluster, and each host belongs to only a single cluster at any layer
- The cluster leaders are the centers of their respective clusters and belong to the immediate higher layer.

3 The NICE Protocol

For the NICE protocol, we assume the existence of a special host that all members know of a-priori. Using nomenclature developed in [7], we call this host the Rendezvous Point (RP). Each host that intends to join the application-layer multicast group contacts the RP to initiate the join process. For ease of exposition, we assume that the RP is always the leader of the single cluster in the highest layer of the hierarchy. It interacts with other cluster members in this layer on the control path, and is bypassed on the data path. (Clearly, it is possible for the RP to not be part of the hierarchy, and for the leader of the highest layer cluster to maintain a connection to the RP, but we do not belabor that compexity further). For a real application like streaming media transfer, the RP could be a distinguished host in the domain of the data source.

The NICE protocol itself has three main components: cluster assignment to new hosts as they join, periodic cluster maintainenance and refinement, and cluster updates for host departures and leader failures. We discuss these in turn.

3.1 New Host Joins

This component is responsible for mapping new hosts to clusters such that the protocol invariants are met.

When a new host joins the multicast group, it must be mapped to some cluster in layer L_0. We illustrate the join procedure in Figure 7. Assume that host A_3 wants to join the multicast group. First, it contacts the RP with

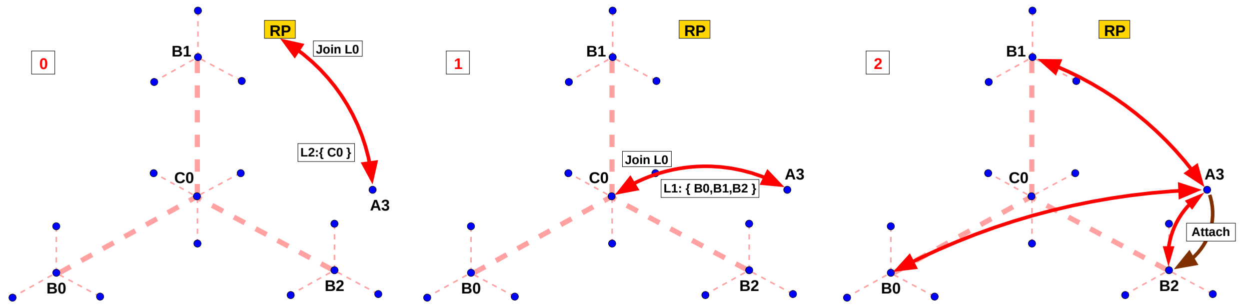

Figure 7: Host A_3 joins the multicast group.

its join query (Panel 0). The RP responds with the hosts that are present in the highest layer of the hierarchy. The joining host then contacts all members in the highest layer (Panel 1) to identify the member closest to itself. In the example, the highest layer L_2 has just one member, C_0, which by default is the closest member to A_3 amongst layer L_2 members. Host C_0 informs A_3 of the three other members (B_0, B_1 and B_2) in its L_1 cluster. A_3 then contacts each of these members with the join query to identify the closest member among them (Panel 2), and iteratively uses this procedure to find its L_0 cluster.

It is important to note that any host, H, which belongs to any layer L_i is the center of its L_{i-1} cluster, and recursively, is an approximation of the center among all members in all L_0 clusters that are below this part of the layered hierarchy. Hence, querying each layer in succession from the top of the hierarchy to layer L_0 results in a progressive refinement by the joining host to find the most appropriate layer L_0 cluster to join that is close to the joining member. This procedure, however, is not infallible. If $d_{highest}$ is the least distance from the joining host to a host in the highest layer, then it is possible for a host to join a cluster that is approximately $d_{highest}$ away from its nearest cluster leader.

3.1.1 Join Latency

The joining process involves a message overhead of $O(k \log N)$ query-response pairs. The join-latency depends on the delays incurred in this exchanges, which is typically about $O(\log N)$ round-trip times. In our protocol, we aggresively locate possible "good" peers for a joining member, and the overhead for locating the appropriate attachments for any joining member is relatively large In contrast, in Narada a joining member initially peers with a set of random other members, and gradually improves the quality of the overlay.

To reduce the delay between a member joining the multicast group, and its receipt of the first data packet on the overlay, we allow joining members to temporarily peer, on the data path, with the leader of the cluster of the current layer it is querying. For example, in Figure 7, when A_3 is querying the hosts B_0, B_1 and B_2 for the closest point of attachment, it temporarily peers with C_0 (leader of the layer L_1 cluster) on the data path. This allows the joining host to start receiving multicast data on the group within a single round-trip latency of its join.

3.1.2 Joining Higher Layers

An important invariant in the hierarchical arrangement of hosts is that the leader of a cluster be the center of the cluster. Therefore, as members join and leave clusters, the cluster-leader may occasionally change. When the leadership of a cluster, C, in layer L_i changes, the existing leader of C removes itself from all layers L_{i+1} and higher

Procedure : $JoinLayer(h, i)$
$\{$ *if* $i = 0$ *then* h *is a new host joining layer* L_0 $\}$
$(C, j) \leftarrow QueryRP()$
$\{$ *RP returns the cluster,* C *in highest layer* L_j $\}$
while $(j > i)$
for $(p \in C)$
$C_p \leftarrow QueryHost(p, j)$
end for
$C \leftarrow C_{p'}$ s.t. $\text{dist}(p', h) = \min\{\text{dist}(h, p), p \in C\}$
$j \leftarrow j - 1$
end while
$\{$ *Appropriate cluster,* C*, found in layer* L_i*, send attach* $\}$
$Attach(\text{Ldr}(C), i)$

Figure 8: Join procedure to a cluster in layer L_i for host, h. If at any layer, during the *QueryHost* for loop, no response is received from any of the cluster members, in that layer L_j, host h, restarts the query from L_{j+1}, or from the RP, as is appropriate.

to which it is attached. The new leader of C joins the appropriate cluster of L_{i+1}. The procedure for joining a higher layer is same the above, except that the process terminates at the layer L_{i+1}, instead of L_0. However, in this case, the new leader does not have to start by querying the RP since members keep information about their super-cluster. It is possible for all of the super-cluster information to be stale; in this case, the member does have to start by contacting the RP.

The join procedure to any layer is presented in pseudo-code as shown in Figure 8.

3.2 Cluster Maintenance and Refinement

Each member H of a cluster C, sends a *HeartBeat* message every h seconds to each of its cluster peers. The message contains the distance estimate of H to each other member of C. It is possible for H to have inaccurate or no estimate of the distance to some other members, e.g. immediately after it joins the cluster.

The cluster-leader includes the complete updated cluster membership in its *HeartBeat* messages to all other members. This allows existing members to set up appropriate peer relationships with new cluster members on the control path. For each cluster in level L_i, the cluster-leader also periodicaly sends the its immediate higher layer cluster membership (which is the super-cluster for all the other members of the cluster) to that L_i cluster.

It is important to note that all of the cluster member state can be sent via unreliable messages and is kept by each cluster member as soft-state, refreshed by the periodic *HeartBeat* messages. A member H is declared no longer part of a cluster independently by all other members in the cluster if they do not receive a message from H for a configurable number of *HeartBeat* message intervals.

3.2.1 Cluster Split and Merge

A cluster-leader periodically checks the size of its cluster, and appropriately splits or merges the cluster when it detects a size bound violation. However, if a cluster that just exceeds the cluster size upper bound $2k - 1$ is split, it creates two clusters of size k each. Any single departure from these clusters will subsequently require a cluster merge operation to meet the size lower-bound. For this reason, we relax the size upper bound to be $3k - 1$ and leave the

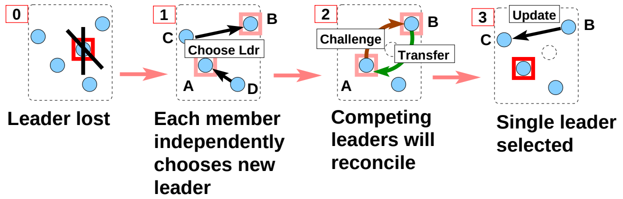

Figure 9: Restructuring when a cluster-leader departs.

lower bound unchanged. With this new upper bound, when a cluster is split into two equal parts, each of the parts is guaranteed to be at least $3k/2$, thus avoiding an immediate subsequent merge.

If the size of a cluster exceeds $3k - 1$, the leader initiates a cluster split operation. Given a set of hosts and the pairwise distances between them, the cluster split operation partitions them into subsets that meet the size bounds, such that the maximum radius (in a graph-theoretic sense) of the new set of clusters is minimized. This is similar to the K-center problem (known to be NP-Hard) with an additional size constraint. We use an approximation strategy — the leader splits the current cluster into two equal-sized clusters, such that the maximum of the radii among the two clusters is minimized. It also chooses the centers of the two partitions to be the leaders of the new clusters and transfers leadership to the new leaders through *LeaderTransfer* messages. If these new clusters still violate the size upper bound, they are split by the new leaders using identical operations.

If the size of a cluster C_i (say in layer L_i) falls below k, its leader J, initiates a cluster merge operation. Note, J itself belongs to a layer L_{i+1} cluster, C_{i+1}. J chooses its closest cluster-peer, K, in C_{i+1}. K is also the leader of a layer L_i cluster, C_i'. J initiates the merge operation of C_i with C_i' by sending a *ClusterMergeRequest* message to K. J updates the members of C_i with this merge information. K similarly updates the members of C_i'. Following the merge, J removes itself from layer L_{i+1}.

3.2.2 Refining Cluster Attachments

During a phase a period of rapid membership changes to the group, a joining member may not be able to locate its closest L_0 cluster, and therefore, attach to some other cluster. This may be true in some cases for higher layer members as well. Therefore, each member in any layer (say L_i) periodically probes all members in its super-cluster (they are the leaders of layer L_i clusters), to identify a closer cluster for itself in layer L_i. If a closer cluster is found, then the probing member leaves its current cluster and joins the closer cluster.

3.3 Host Departure and Leader Selection

When a host, H, leaves the multicast group, it sends a *Remove* message to all clusters to which it is joined. This is a graceful-leave. However, if H fails without being able to send out this message all cluster peers of H detects this departure through non-receipt of the periodic *HeartBeat* message from H. If H was a leader of a cluster, this triggers a new leader selection in the cluster. Each remaining member, J, of the cluster independently select a new leader of the cluster, depending on who J estimates to be the center among these members. Multiple leaders are re-conciled into a single leader of the cluster through exchange of additional control messages (*LeaderChallenge* and *LeaderTransfer*) each time two candidate leaders detect this multiplicity. This is shown in Figure 9.

It is possible for members to have an inconsistent view of the cluster membership, and for transient cycles to develop on the data path. These cycles are eliminated once the protocol reconciles the cluster view for all members, and restores the hierarchy invariants.

4 Simulation Experiments

In this section, we analyze the performance of NICE and compare it to three other protocols using detailed simulations. The three other schemes we consider are: multi-unicast, native IP-multicast using the Core Based Tree protocol [2], and the Narada application-layer multicast protocol.

Clearly, native IP multicast trees will have the least (unit) stress, and the multi-unicast trees will have the best (unit) stretch. Thus, these two schemes provide us a reference against which to compare both Narada and NICE.

4.1 Simulation Environment

We have implemented a packet-level simulator for the four different protocols[3]. Our network topologies were generated using the Transit-Stub graph model, using the GT-ITM topology generator [3]. All topologies in these simulations had $10,000$ routers with an average node degree between 3 and 4. End-hosts were attached to a set of route, chosen at uniformly at random, from among the stub-domain nodes. The number of such hosts in the multicast group were varied between 8 and 2048 for different experiments. In our simulations, we only modeled loss-less links; thus, there is no data loss due to congestion, and no notion of background traffic or jitter. However, data is lost whenever the application-layer multicast protocol fails to provide a path from the source to a receiver, and duplicates are received whenever there is more than one path. Thus, our simulations study the dynamics of the multicast protocol and its effects on data distribution; when these protocols are implemented, the performance would also be affected by other factors such as additional link latencies due to congestion.

As mentioned in Section 1, the Narada protocol involves an aggregate control overhead of $O(N^2)$. In our simulation setup, we were unable to simulate Narada with groups of size 1024 or larger since the completion time for these simulations were on the order of a day for a single run of one experiment on a 550 MHz Pentium III machine with 4 GB of RAM[4].

4.1.1 Our implementation of Narada

We implemented the entire Narada protocol from the description given in [7]. As described before, Narada is a mesh-first application-layer multicast approach, designed primarily for small multicast groups. In Narada, the initial set of peer assignments to create the overlay topology is done randomly. While this initial data delivery path may be of "poor" quality, over time Narada adds "good" links and discards "bad" links from the overlay. Narada has $O(N^2)$ aggregate control overhead because of its mesh-first nature: it requires each host to periodically exchange updates and refreshes with all other hosts.

The protocol, as defined in [7], has a number of user-defined parameters that we needed to set. These include the link add/drop thresholds, link add/drop probe frequency, the periodic refresh rates, the mesh degree, etc. We experimented with a wide-range of values for these parameters to understand the behavior of Narada and observed some interesting trade-offs in choosing these parameters. Specifically, we found that:

- The mesh degree bound for hosts should not be strictly enforced to ensure connectivity. Instead additional mechanisms that limit the degree of the data path on the mesh should be used.
- There is a clear tradeoff between choosing a high versus low frequency for periodic probes to add or drop links on the mesh. A high frequency allows members to aggressively add and drop good and bad overlay links respectively. However, this leads to frequent changes to the data paths on the mesh, which can lead to a temporary loss of data path to other members. (This effect is different than when a route changes and state

[3] Our simulator is available upon email request to the authors

[4] The NICE implementation with 1024 hosts on the same environment finishes in less than 10 minutes.

for the old route can be temporarily maintained to mitigate the effect of the route change). We observed this effect in our experiments where we use a high periodic probe frequency, especially if this parameter is set higher than the route packet exchange frequency. In contrast, using a low probe frequency leads to more stable paths; however, this implies that the mesh topology takes a long time to stabilize.

4.1.2 Data model

In these experiments, we wanted to model the scenario of a distinguished source streaming multimedia data to the multicast group. Therefore, we chose a single end-host, uniformly at random, to be the data source, and generate constant bit rate data. Since our simulations only model losses due to the application-layer multicast protocol behavior, the bandwidth of the data stream is irrelevant. Thus, each packet in the data sequence samples the data path on the overlay topology at that time instant, and the entire data packet sequence captures the evolution of the data path over time.

4.2 Performance Metrics

We compare the performance of the different schemes along the following dimensions:

- *Quality of data path:* This is measured by three different metrics — tree degree distribution, stress on links and routers and stretch of data paths to the group members.
- *Recovery from host failure:* As hosts join and leave the multicast group, the underlying data delivery path adapts accordingly to reflect these changes. However, in transience, and particularly after host failures, path to some hosts may be unavailable. It is also possible for multiple paths to exist to a single host and for cycles to develop temporarily. To observe these effects, we measured the fraction of hosts that correctly receive the data packets sent from the source. We also recorded the number of duplicates at each host. In all of our simulations, for both the application-layer multicast protocols, the number of duplicates was insignificant and zero in most cases.
- *Control traffic overhead:* We report the aggregate control bandwidth overheads at both routers and end hosts.

In the next section, we present a number of results from our simulations. Broadly, our findings can be summarized as follows: *NICE trees have data paths that have stretch comparable to Narada. The stress on links and routers are lower in NICE, especially as the multicast group size increases. The failure recovery of both the schemes are comparable, however, NICE provides similar performance with orders of magnitude lower control overhead for groups of size* > 32.

4.3 Simulation Results

In our experiments, we have simulated a wide-range of topologies, group sizes, member join-leave patterns, and protocol parameters [5]. We begin with results from a representative experiment that captures all the of different aspects comparing the various protocols.

[5] For NICE, we set the cluster size parameter, k, as 3 in all the experiments presented here.

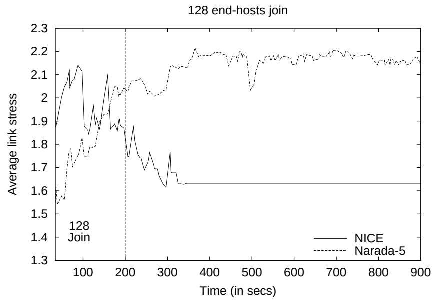

Figure 10: Average link stress

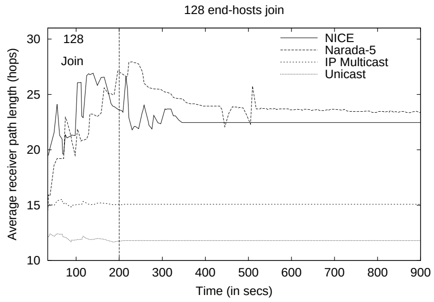

Figure 11: Average path length

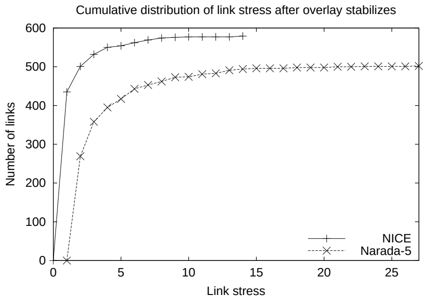

Figure 12: Stress distribution

4.3.1 Representative Scenario

This experiment has two different phases: a join phase and a leave phase. In the join phase a set of 128 members[6] join the multicast group uniformly at random between the simulated time 0 and 200 seconds. These hosts are allowed to stabilize into an appropriate overlay topology till simulation time 1000 seconds. The leave phase starts at time 1000 seconds: 16 hosts leave the multicast group over a short duration of 10 seconds. This is repeated four more times, at 100 second intervals. The remaining 48 members continue to be part of the multicast group till the end of simulation. All member departures are modeled as host failures since they have the most damaging effect on data paths. We experimented with different numbers of member departures, from a single member to 32 members leaving over the ten second window. Sixteen departures from a group of size 128 within a short time window is a drastic scenario, but it helps illustrate the failure recovery modes of the different protocols better. Member departures in smaller sizes cause correspondingly lower disruption on the data paths.

We experimented with different periodic refresh rate for Narada. For a higher refresh rate the recovery from host failures is quicker, but at a cost of higher control traffic overhead. For Narada, we used different values for route update frequencies and periods for probing other mesh members to add or drop links on the overlay. In our results, we report results from using route update frequencies of once every 5 seconds (labeled Narada-5), and once every 30 seconds (labeled Narada-30). The 30 second update period corresponds to the what was used in [7]; we ran with the 5 second update period since the heartbeat period in NICE was set to 5 seconds. Note that we could run with a much smaller heartbeat period in NICE without significantly increasing control overhead since the control messages are limited within clusters and do not traverse the entire group. We also varied the mesh probe period in Narada and observed data path instability effect discussed above. In these results, we set the Narada mesh probe period to 20 seconds.

Data Path Quality

In Figures 10 and 11, we show the average link stress and the average path lengths for the different protocols as the data tree evolves during the member join phase. Note that the figure shows the actual path lengths to the end-hosts; the stretch is the ratio of average path length of the members of a protocol to the average path length of the members in the multi-unicast protocol.

As explained earlier, the join procedure in NICE aggressively finds good points of attachment for the members in

[6] We show results for the 128 member case because that is the group size used in the experiments reported in [7]; NICE performs increasingly better with larger group sizes.

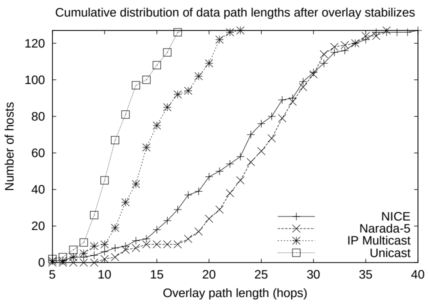

Figure 13: Path length distribution

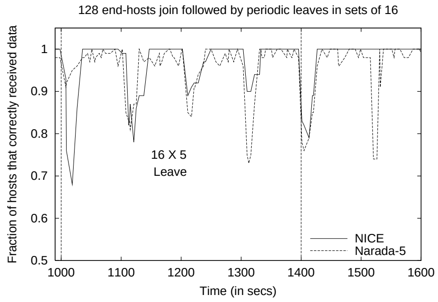

Figure 14: Fraction of members that received data packets over the duration of member failures.

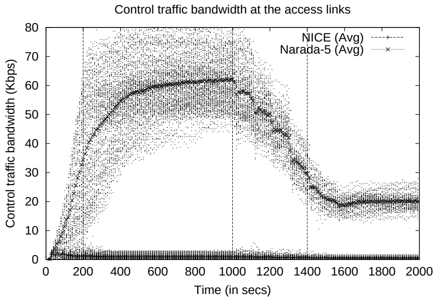

Figure 15: Control bandwidth required at end-host access links

the overlay topology, and the NICE tree converges quicker to a stable value (within 350 seconds of simulated time). In contrast, the Narada protocols gradually improve the mesh quality, and consequently so does the data path over a longer duration. Its average data path length converges to a stable value of about 23 hops between 500 and 600 seconds of the simulated time. The corresponding stretch is about 2.18. In Narada path lengths improve over time due to addition of "good" links on the mesh. At the same time, the stress on the tree gradually increases since the Narada decides to add or drop overlay links based purely on the stretch metric.

The cluster-based data dissemination in NICE reduces average link stress, and in general, for large groups NICE converges to trees with about 25% lower average stress. In this experiment, the NICE tree had lower stretch than the Narada tree; however, in other experiments the Narada tree had a slightly lower stretch value. In general, comparing the results from multiple experiments over different group sizes, (See Section 4.3.2), we concluded that the data path lengths to receivers were similar for both protocols.

In Figures 12 and 13, we plot a cumulative distribution of the stress and path length metrics for the entire member set (128 members) at a time after the data paths have converged to a stable operating point.

Narada uses fewer number of links on the topology than NICE, since it is comparably more aggressive in adding overlay links with shorter lengths to the mesh topology. However, due to this emphasis on shorter path lengths, the stress distribution of the links is heavy-tailed. More than 25% of the links have a stress of four and higher in Narada, compared to $< 5\%$ in NICE. The distribution of the path lengths for the two protocols are comparable. The multi-unicast scheme (presented for comparison) shows the shortest path length distribution if stress on links is ignored.

Failure Recovery and Control Overheads

To investigate the effect of host failures, we present results from the second part of our scenario: starting at simulated time 1000 seconds, a set of 16 members leave the group over a 10 second period. We repeat this procedure four more times and no members leave after simulated time 1400 seconds when the group is reduced to 48 members. When members leave, both protocols "heal" the data distribution tree and continue to send data on the partially connected topology. In Figure 14, we show the fraction of members that correctly receive the data packets over this duration. Both Narada-5 and NICE have similar performance, and on average, both protocols restore the data path to all (remaining) receivers within 30 seconds, and on correctly serve over 90% of the members. We also ran the same experiment with the 30 second refresh period for Narada. The lower refresh period caused significant disruptions on

Group	Router Stress		Link Stress		Path Length		Bandwidth Overheads (Kbps)	
Size	Narada-5	NICE	Narada-5	NICE	Narada-5	NICE	Narada-30	NICE
8	1.55 (1.30)	3.51 (3.30)	**1.19** (0.39)	3.24 (2.90)	25.14 (9.49)	**12.14** (2.29)	0.61 (0.55)	1.54 (1.34)
16	1.84 (1.28)	2.34 (2.16)	**1.34** (0.76)	1.86 (1.39)	**19.00** (7.01)	20.33 (6.75)	2.94 (2.81)	0.87 (0.81)
32	2.13 (2.17)	2.42 (2.60)	**1.54** (1.03)	1.90 (1.82)	20.42 (6.00)	**17.23** (5.25)	9.23 (8.95)	1.03 (0.95)
64	2.68 (3.09)	2.23 (2.25)	1.74 (1.53)	**1.63** (1.39)	22.76 (5.71)	**20.62** (7.40)	26.20 (28.86)	1.20 (1.15)
128	3.04 (4.03)	2.36 (2.73)	2.06 (2.64)	**1.63** (1.56)	**21.55** (6.03)	21.61 (7.75)	65.62 (92.08)	1.19 (1.29)
256	3.63 (7.52)	2.31 (3.18)	2.16 (3.02)	**1.63** (1.63)	**23.42** (6.17)	24.67 (7.45)	96.18 (194.00)	1.39 (1.76)
512	4.09 (10.74)	2.34 (3.49)	2.57 (5.02)	**1.62** (1.54)	24.74 (6.00)	**22.63** (6.78)	199.96 (55.06)	1.93 (3.35)
1024	-	2.59 (4.45)	-	1.77 (1.77)	-	25.83 (6.13)	-	2.81 (7.22)
1560	-	2.83 (5.11)	-	1.88 (1.90)	-	24.99 (6.96)	-	3.28 (9.58)
2048	-	2.92 (5.62)	-	1.93 (1.99)	-	24.08 (5.36)	-	5.18 (18.55)

Table 1: Data path quality and control overheads for varying multicast group sizes

the tree with periods of over 100 seconds when more than 60% of the tree did not receive any data. Lastly, we note that the data distribution tree used for NICE is the least connected topology possible; we expect failure recovery results to be much better if structures with alternate paths are built atop NICE.

In Figure 15, we show the byte-overheads for control traffic at the access links of the end-hosts. Each dot in the plot represents the sum of the control traffic (in Kbps) sent or received by each member in the group, averaged over 10 second intervals. Thus for each 10 second time slot, there are two dots in the plot for each (remaining) host in the multicast group corresponding to the control overheads for Narada and NICE. The curves in the plot are the average control overhead for each protocol. As is evident from the plot, for groups of size 128, NICE has an order of magnitude lower average overhead, e.g. at simulation time 1000 seconds, the average control overhead for NICE is 0.97 Kbps versus 62.05 Kbps for Narada. At the same time instant, Narada-30 (not shown in the figure) had an average control overhead of 13.43 Kbps. Lastly, we note that the NICE control traffic includes all protocol messages, including messages for cluster formation, cluster splits, merges, layer promotions, and leader elections.

4.3.2 Aggregate Results

We present a set of aggregate results as the group size is varied. The purpose of this experiment is to understand the scalability of the different application-layer multicast protocols. The entire set of members join in the first 200 seconds, and then we run the simulation for 1800 seconds to allow the topologies to stabilize. The data path tree degree was low for both the protocols, and typically varied between 3 and 5 for the non-leaf members on the distribution tree.

In Table 1, we compare the stress on network routers and links, the overlay path lengths to group members and the average control traffic overheads at the network routers. For each metric, we present the both mean and the standard deviation.

As we showed in our first experiment, Narada and NICE tend to converge to trees with similar path lengths. The stress metric for both network links and routers, however, is consistently lower for NICE when the group size is large (64 and greater). It is interesting to observe the standard deviation of stress as it changes with increasing group size for the two protocols. The standard deviation for stress increased for Narada for increasing group sizes. In contrast, the standard deviation of stress for NICE remains relatively constant; the topolgy-based clustering in NICE distributes the data path more evenly among the different links on the underlying links regardless of group size.

The control overhead numbers in the Table are different than the ones in Figure 15; the column in the table is the average control traffic *per network router* as opposed to control traffic at an end-host. Since the control traffic gets aggregated inside the network, the overhead at routers is significantly higher than the overhead at an end-host. For these router overheads, we report the values of the Narada-30 version in which the route update frequency set to 30 seconds. Recall that this protocol, Narada-30 performs relatively poorly when members leave, but is much more efficient (specifically 5 *times* less overhead with groups of size 128) than the Narada-5 version. The refresh messages in NICE were still sent at 5 second intervals. Even with this disparity, the average control traffic byte overheads at a router for the different group sizes is an order of magnitude lower for NICE.

5 Related Work

There are a few closely related projects which explore implementing multicast at the application layer. They can be classified into two broad categories: mesh-first (Narada [7], Gossamer [4]) and tree-first protocols (Yoid [9], ALMI [12]). Yoid defines a distributed tree building protocol between the end-hosts, while ALMI uses a centralized algorithm to create a minimum spanning tree rooted at a designated single source of multicast data distribution. The JungleMonkey project[7] is a similar effort to create an application layer multicast overlay using a tree-first approach.

Bayeux [16] in another architecture for application layer multicast, where the end-hosts are organized into a hierarchy as defined by the Tapestry overlay location and routing system [15]. A level of the hierarchy is defined by a set of hosts that share a common suffix in their host IDs. Such a technique was proposed by Plaxton et.al. [13] for locating and routing to named objects in a network.

The Overcast [11] protocol organizes a set of similar proxies (called Overcast nodes) into a distribution tree rooted at a central source for single source multicast. A distributed tree-building protocol is used to create this source specific tree, in a manner similar to Yoid. RMX [5] provides support for reliable multicast data delivery to end-hosts using a set of such proxies, called Reliable Multicast proXies. Application end-hosts are configured to affiliate themselves with the nearest RMX. The architecture assumes the existence of an overlay construction protocol, using which these proxies organize themselves into an appropriate data delivery path. TCP is used to provide reliable communication between each pair of peer proxies on the overlay.

6 Conclusions

In this paper, we have presented a new protocol for application-layer multicast. Our main contribution is an extremely low overhead hierarchical control structure over which different data distribution paths can be built. Our results show that it is possible to build and maintain application-layer multicast trees with very little overhead. Clearly, existing protocols like Narada are very useful for small group sizes, but impose too much overhead for applications that require large groups. We believe that the results of this paper are a significant first step towards contructing large wide-area applications over application-layer multicast.

References

[1] D. G. Andersen, H. Balakrishnan, M. Frans Kaashoek, and R. Morris. The Case for Resilient Overlay Networks. In *Proc. HotOS VIII, Schloss Elmau, Germany*, May 2001.

[2] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT): An Architecture for Scalable Multicast Routing. In *Proceedings of ACM Sigcomm*, 1995.

[3] K. Calvert, E. Zegura, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom*, 1996.

[4] Y. Chawathe. Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service. *Ph.D. Thesis, University of California, Berkeley*, December 2000.

[7] http://www.junglemonkey.net

[5] Y. Chawathe, S. McCanne, and E. A. Brewer. RMX: Reliable Multicast for Heterogeneous Networks. In *Proceedings of Infocom*, 2000.

[6] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proceedings of ACM SIGCOMM*, August 2001.

[7] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.

[8] S. Deering and D. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. In *ACM Transactions on Computer Systems*, May 1990.

[9] P. Francis. Yoid: Extending the Multicast Internet Architecture, 1999. White paper http://www.aciri.org/yoid/.

[10] A. Gupta. Steiner points in tree metrics don't (really) help. In *Symposium of Discrete Algorithms*, January 2001.

[11] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, 2000.

[12] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An Application Level Multicast Infrastructure. In *Proceedings of 3rd Usenix Symposium on Internet Technologies & Systems*, March 2001.

[13] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[14] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collin s, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: A Case for Informed Internet Routing and Transport. In *IEEE Micro*, 1999.

[15] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. In *Tech. Report UCB/CSD-01-1141, University of California, Berkeley*, April 2001.

[16] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area dat a dissemination. In *Eleventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)*, 2001.