Scalable Application Layer Multicast

Suman Banerjee, Bobby Bhattacharjee, Christopher Kommareddy
Department of Computer Science,University of Maryland, College Park, MD 20742, USA
Email: {suman,bobby,kcr}@cs.umd.edu

UMIACS-TR 2002-53 and CS-TR 4373
May 2002

***Abstract*— We describe a new scalable application-layer multicast protocol, specifically designed for low-bandwidth data streaming applications with large receiver sets. Our scheme is based upon a hierarchical clustering of the application-layer multicast peers and can support a number of different data delivery trees with desirable properties.**

We present extensive simulations of both our protocol and the Narada application-layer multicast protocol over Internet-like topologies. Our results show that for groups of size 32 or more, our protocol has lower link stress (by about 25%), improved or similar end-to-end latencies and similar failure recovery properties. More importantly, it is able to achieve these results by using orders of magnitude lower control traffic.

Finally, we present results from our wide-area testbed in which we experimented with 32-100 member groups distributed over 8 different sites. In our experiments, average group members established and maintained low-latency paths and incurred a maximum packet loss rate of less than 1% as members randomly joined and left the multicast group. The average control overhead during our experiments was less than 1 Kbps for groups of size 100.

I. Introduction

Multicasting is an efficient mechanism for packet delivery in one-many data transfer applications. It eliminates redundant packet replication in the network. It also decouples the size of the receiver set from the amount of state kept at any single node and therefore, is an useful primitive to scale multi-party applications. However, deployment of network-layer multicast [10] has not been widely adopted by most commercial ISPs, and thus large parts of the Internet are still incapable of native multicast more than a decade after the protocols were developed. *Application-Layer Multicast* protocols [9], [11], [6], [13], [14], [23], [17] do not change the network infrastructure, instead they implement multicast forwarding functionality exclusively at end-hosts. Such application-layer multicast protocols are increasingly being used to implement efficient commercial content-distribution networks.

In this paper, we present a new application-layer multicast protocol which has been developed in the context of the NICE project at the University of Maryland [1]. NICE is a recursive acronym which stands for NICE is the Internet Cooperative Environment. In this paper, we refer to the NICE application-layer multicast protocol as simply the NICE protocol. This protocol is designed to support applications with large receiver sets. Such applications include news and sports ticker services such as Infogate (http://www.infogate.com) and ESPN Bottomline (http://www.espn.com); real-time stock quotes and updates, e.g. the Yahoo! Market tracker, and popular Internet Radio sites. All of these applications are characterized by very large (potentially tens of thousands) receiver sets and relatively low bandwidth soft real-time data streams that can withstand some loss. We refer to this class of large receiver set, low bandwidth real-time data applications as *data stream* applications. Data stream applications present a unique challenge for application-layer multicast protocols: the large receiver sets usually increase the control overhead while the relatively low-bandwidth data makes amortizing this control overhead difficult. NICE can be used to implement very large data stream applications since it has a provably small (constant) control overhead and produces low latency distribution trees. It is possible to implement high-bandwidth applications using NICE as well; however, in this paper, we concentrate exclusively on low bandwidth data streams with large receiver sets.

A. Application-Layer Multicast

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 multicast data packets are replicated at end hosts.

[1] See http://www.cs.umd.edu/projects/nice

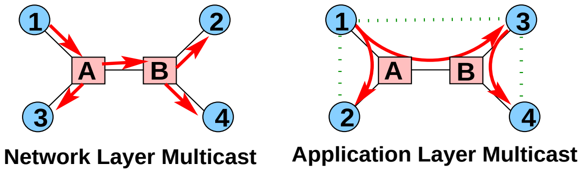

Fig. 1. Network-layer and application layer multicast. Square nodes are routers, and circular nodes are end-hosts. The dotted lines represent peers on the overlay.

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 "goodness" for application layer multicast overlays, namely *stress* and *stretch*, were defined in [9]). 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 the member along the overlay to the length of the direct unicast 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 does so at a cost of $O(N)$ stress at links near the source (N is the number of group members). It also requires $O(N)$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.

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 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 protocol to restore delivery to the other members. We present the first comparison of this aspect of application-layer multicast protocols.
- *Control overhead:* For efficient use of network resources, the control overhead at the members should be low. This is an important cost metric to study the scalability of the scheme to large member groups.

B. NICE Trees

Our goals for NICE were to develop an efficient, scalable, and distributed tree-building protocol which did not require any underlying topology information. Specifically, the NICE protocol reduces the worst-case state and control overhead at any member to $O(\log N)$, maintains a constant degree bound for the group members and approach the $O(\log N)$ stretch bound possible with a topology-aware centralized algorithm. Additionally, we also show that an average member maintains state for a *constant number of other members*, and incurs *constant control overhead* for topology creation and maintenance.

In the NICE application-layer multicast scheme, we create a hierarchically-connected control topology. The data delivery path is implicitly defined in the way the hierarchy is structured and no additional route computations are required.

Along with the analysis of the various bounds, we also present a simulation-based performance evaluation of NICE. In our simulations, we compare NICE to the Narada application-layer multicast protocol [9]. Narada was first proposed as an efficient application-layer multicast protocol for small group sizes. Extensions to it have subsequently been proposed [8] to tailor its applicability to high-bandwidth media-streaming applications for these groups, and have been studied using both simulations and implementation. Lastly, we present results from a wide-area implementation in which we quantify the NICE run-time overheads and convergence properties for various group sizes.

C. Roadmap

The rest of the paper is structured as follows: In Section II, we describe our general approach, explain how different delivery trees are built over NICE and present theoretical bounds about the NICE protocol. In Section III, we present the operational details of the protocol. We present our performance evaluation methodology in Section IV, and present detailed analysis of the NICE protocol through simulations in Section V and a wide-area implementation in Section VI. We elaborate on related work in Section VII, and conclude in Section VIII.

II. 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 implicitly defines the multicast overlay data paths, as described later in this section. The member hierarchy is crucial for scalability, since most members are in the bottom of the hierarchy and only maintain state about a constant number of other

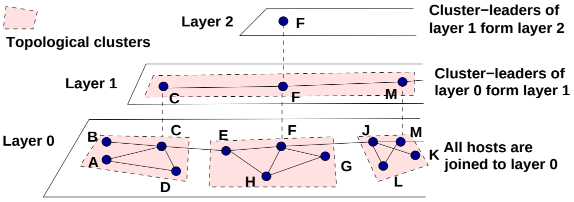

Fig. 2. Hierarchical arrangement of hosts in NICE. The layers are logical entities overlaid on the same underlying physical network.

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.

The NICE hierarchy described in this paper is similar to the member hierarchy used in [3] for scalable multicast group re-keying. However, the hierarchy in [3], is layered over a multicast-capable network and is constructed using network multicast services (e.g. scoped expanding ring searches). We build the necessary hierarchy on a unicast infrastructure to provide a multicast-capable network.

In this paper, we use end-to-end latency as the distance metric between hosts. While constructing the NICE 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.

A. *Hierarchical Arrangement of 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 $3k-1$, where k is a constant, and consists of a set of hosts that are close to each other. We explain our choice of the cluster size bounds later in this paper (Section III-B.1). Further, each cluster has a cluster leader. The protocol distributedly chooses the (graph-theoretic) center of the cluster to be its leader, i.e. the cluster leader has the minimum maximum distance to all other hosts in the cluster. This choice of the cluster leader is important in guaranteeing that a new joining member is quickly able to find its appropriate position in the hierarchy using a very small number of queries to other members.

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 single cluster, [CFM], in layer L_1. F is the center of this cluster, and hence its leader. Therefore F belongs to layer L_2 as well.

The NICE clusters and layers are created using a distributed algorithm described in the next section. 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 $j > i$.
- Each cluster has its size bounded between k and $3k-1$. The leader is the graph-theoretic center of the cluster.
- 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, X. Assume that host, X, belongs to layers $L_0, \ldots, L_{i-1}$ and no other layer, and let [..XYZ..] be the cluster it belongs it in its highest layer (i.e. layer L_{i-1}) with Y its leader in that cluster. Then, the super-cluster of X is defined as the cluster, in the next higher layer (i.e. L_i), to which its leader Y belongs. It follows that there is only one super-cluster defined for every host (except the host that belongs to the top-most layer, which 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 hosts B, A, and D. In NICE each host maintains state about all the clusters it belongs to (one in each layer to which it belongs) and about its super-cluster.

B. *Control and Data Paths*

The host hierarchy can be used to define different overlay structures for control messages and data delivery paths. The neighbors on the control topology exchange periodic soft state refreshes and do not generate high volumes of

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

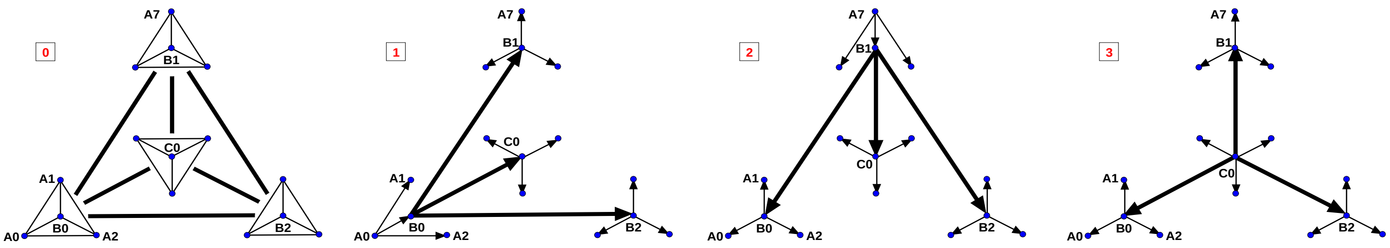

Fig. 3. Control and data delivery paths for a two-layer hierarchy. All A_i hosts are members of only L_0 clusters. All B_i hosts are members of both layers L_0 and L_1. The only C host is the leader of the L_1 cluster comprising of itself and all the B hosts.

traffic. Clearly, 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 the choices of control and data paths using clusters of size 4. The edges in the figure indicate the peerings between group members on the overlay topology. 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. In the rest of the paper, we use $Cl_j(X)$ to denote the cluster in layer L_j to which member X belongs. It is defined if and only if X belongs to layer L_j.

The control topology for the NICE protocol is illustrated in Figure 3, Panel 0. Consider a member, X, that belongs only to layers $L_0, \ldots, L_i$. Its peers on the control topology are the other members of the clusters to which X belongs in each of these layers, i.e. members of clusters $Cl_0(X), \ldots, Cl_i(X)$. Using the example (Figure 3, Panel 0), member A_0 belongs to only layer L_0, and therefore, its control path peers are the other members in its L_0 cluster, i.e. A_1, A_2 and B_0. In contrast, member B_0 belongs to layers L_0 and L_1 and therefore, its control path peers are all the other members of its L_0 cluster (i.e. A_0, A_1 and A_2) and L_1 cluster (i.e. B_1, B_2 and C_0). In this control topology, each member of a cluster, therefore, exchanges soft state refreshes with all the remaining members of the cluster. This allows all cluster members to quickly identify changes in the cluster membership, and in turn, enables faster restoration of a set of desirable invariants (described in Section II-D), which might be violated by these changes.

The delivery path for multicast data distribution needs to be loop-free, otherwise, duplicate packet detection and suppression mechanisms need to be implemented. Therefore, in the NICE protocol we choose the data delivery path to be a tree. More specifically, given a data source, the data delivery path is a source-specific tree, and is implicitly defined from the control topology. Each member

Procedure : *MulticastDataForward*(h, p)
    $\{\ h \ \in$ *layers* $L_0, \ldots, L_i$ *in clusters* $\mathrm{Cl}_0(h), \ldots, \mathrm{Cl}_i(h)\ \}$
for j in $[0, \ldots, i]$
    if $(p \notin Cl_j(h))$
        ForwardDataToSet$(Cl_j(h) - \{h\})$
    end if
end for

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

executes an instance of the Procedure *MulticastDataForward* given in Figure 4, to decide the set of members to which it needs to forward the data. Panels 1, 2 and 3 of Figure 3 illustrate the consequent source-specific trees when the sources are at members A_0, A_7 and C_0 respectively. We call this the *basic data path*.

To summarize, in each cluster of each layer, the control topology is a clique, and the data topology is a star. It is possible to choose other structures, e.g. in each cluster, a ring for control path, and a balanced binary tree for data path.

C. Analysis

Each cluster in the hierarchy has between k and $3k-1$ members. Then for the control topology, a host that belongs only to layer L_0 peers with $O(k)$ other hosts for exchange of control messages. In general, a host that belongs to layer L_i and no other higher layer, peers with $O(k)$ other hosts in each of the layers $L_0, \ldots, L_i$. Therefore, the control overhead for this member is $O(k\ i)$. Hence, the cluster-leader of the highest layer cluster (Host C_0 in Figure 3), peers with a total of $O(k \log N)$ neighbors. This is the worst case control overhead at a member.

It follows using *amortized cost* analysis that the control overhead at an average member is a constant. The number of members that occur in layer L_i and no other higher layer is bounded by $O(N/k^i)$. Therefore, the amortized control overhead at an average member is

$$\leq \frac{1}{N}\sum_{i=0}^{\log N}\frac{N}{k^i}k.i = O(k) + O(\frac{\log N}{N}) + O(\frac{1}{N}) \rightarrow O(k)$$

with asymptotically increasing N. Thus, the control overhead is $O(k)$ for the average member, and $O(k \log N)$ in the worst case. The same holds analogously for stress at members on the basic data path [3]. Also, the number of application-level hops on the basic data path between any pair of members is $O(\log N)$.

While an $O(k \log N)$ peers on the data path is an acceptable upper-bound, we have defined enhancements that further reduce the upper-bound of the number of peers of a member to a constant. The stress at each member on this enhanced data path (created using local transformations of the basic data path) is thus reduced to a constant, while the number of application-level hops between any pair of members still remain bounded by $O(\log N)$. We outline this enhancement to the basic data path in the Appendix.

D. *Invariants*

All the properties described in the analysis hold as long as the hierarchy is maintained. 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 partitioned into clusters of size between k and $3k-1$.
- 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 form the immediate higher layer.

III. Protocol Description

In this section we describe the operations of the NICE protocol. We assume the existence of a special host that all members know of a-priori. Using nomenclature developed in [9], 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 complexity further). For an application such as streaming media delivery, the RP could be a distinguished host in the domain of the data source.

[3] Note that the stress metric at members is equivalent to the degree of the members on the data delivery tree.

The NICE protocol itself has three main components: initial cluster assignment as a new host joins, periodic cluster maintenance and refinement, and recovery from leader failures. We discuss these in turn.

A. *New Host Joins*

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 5. Assume that host A_{12} wants to join the multicast group. First, it contacts the RP with 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_{12} amongst layer L_2 members. Host C_0 informs A_{12} of the three other members (B_0, B_1 and B_2) in its L_1 cluster. A_{12} 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. The outline of this operation are presented in pseudocode as Procedure *BasicJoinLayer* in Figure 6.

We assume that all hosts are aware of only a single well-known host, the RP, from which they initiate the join process. Therefore, overheads due to join query-response messages is highest at the RP and descreases down the layers of the hierarchy. Under a very rapid sequence of joins, the RP will need to handle a large number of such join query-response messages. Alternate and more scalable join schemes are possible if we assume that the joining host is aware of some other "nearby" host that is already joined to the overlay. In fact, both Pastry [18] and Tapestry [22] alleviate a potential bottleneck at the RP for a rapid sequence of joins, based on such an assumption.

1) Join Latency: The joining process involves a message overhead of $O(k \log N)$ query-response pairs. The

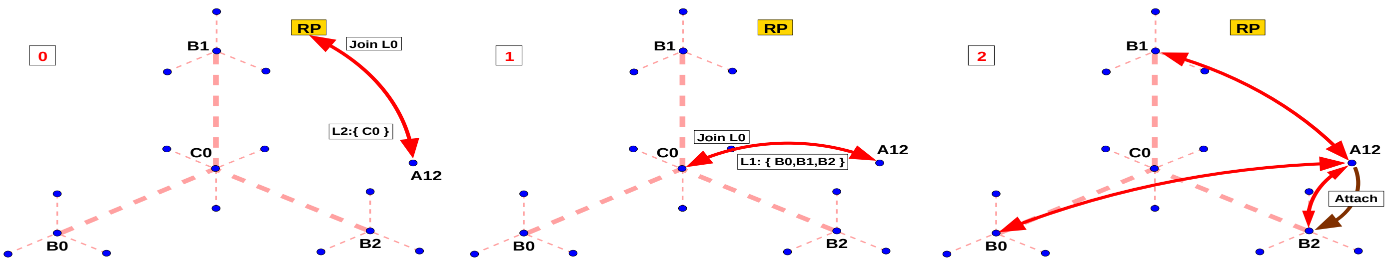

Fig. 5. Host A_{12} joins the multicast group.

Procedure : *BasicJoinLayer*(h, i)
Cl$_j \leftarrow$ *Query*$(RP, -)$
while $(j > i)$
  Find y s.t. dist$(h, y) \leq$ dist$(h, x), x, y \in$ *Cl*$_j$
  Cl$_{j-1}(y) \leftarrow$ *Query*$(y, j-1)$
  Decrement j, *Cl*$_j \leftarrow$ *Cl*$_{j-1}(y)$
endwhile
JoinCluster$(h$,*Ldr*$($*Cl*$_j), L_i)$

Fig. 6. Basic join operation for member h, to join layer L_i. $i = 0$ for a new member. If $i > 0$, then h is already part of layer L_{i-1}. *Query*$(y, j-1)$ seeks the membership information of *Cl*$_{j-1}(y)$ from member y. *Query*$(RP, -)$ seeks the membership information of the topmost layer of the hierarchy, from the RP. *JoinCluster*(x, y, L_j) operation sends an appropriate message to add new member x to a cluster in layer L_j. The message is sent to y, the leader of the cluster.

join-latency depends on the delays incurred in this exchanges, which is typically about $O(\log N)$ round-trip times. In our protocol, we aggressively locate possible "good" peers for a joining member, and the overhead for locating the appropriate attachments for any joining member is relatively large.

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 5, when A_{12} 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.

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. Consider a change in leadership of a cluster, C, in layer L_j. The current leader of C removes itself from all layers L_{j+1} and higher to which it is attached. A new leader is chosen for each of these affected clusters. For example, a new leader, h, of C in layer L_j is chosen which is now required to join its nearest L_{j+1} cluster. This is its current super-cluster (which by definition is the cluster in layer L_{j+1} to which the outgoing leader of C was joined to), i.e. the new leader replaces the outgoing leader in the super-cluster. However, if the super-cluster information is stale and currently invalid, then the new leader, h, invokes the join procedure to join the nearest L_{j+1} cluster. It calls *BasicJoinLayer*$(h, j+1)$ and the routine terminates when the appropriate layer L_{j+1} cluster is found. Also note that the *BasicJoinLayer* requires interaction of the member h with the RP. The RP, therefore, aids in repairing the hierarchy from occasional overlay partitions, i.e. if the entire super-cluster information becomes stale in between the periodic *HeartBeat* messages that are exchanged between cluster members. If the RP fails, for correct operation of our protocol, we require that it be capable of recovery within a reasonable amount of time.

B. Cluster Maintenance and Refinement

Each member x of a cluster C, sends a *HeartBeat* message every τ seconds to each of its cluster peers (neighbors on the control topology). The message contains the distance estimate of x to each other member of C. It is possible for x 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 periodically 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.

All of the cluster member state is sent via unreliable messages and is kept by each cluster member as soft-state, refreshed by the periodic *HeartBeat* messages. A member x is declared no longer part of a cluster independently

Procedure : *ClusterSplit*(C)
  $\{ \ |C| \geq 3k \ \}$
$\sigma \leftarrow \{Q | Q \subset C \ \wedge \ |Q|, |C - Q| \geq \lfloor 3k/2 \rfloor\}$
Let $R(Q) \equiv \max($*radius*(Q),*radius*$(C - Q))$
Find Q^* s.t. $R(Q^*) \leq R(Q)$ where $Q, Q^* \in \sigma$
LeaderTransfer(*Ldr*(C), Q^*, *Ldr*(Q^*))
LeaderTransfer(*Ldr*(C), $C - Q^*$, *Ldr*($C - Q^*$))

Fig. 7. Cluster split operation for cluster, C, which exceeds the size bound. The operation is invoked by the leader of cluster, C. The appropriate partitions (Q^* and $C - Q^*$) can be naively implemented in with a running time of $O(|C|^3)$. *radius*(Q) defines the graph-theoretic radius of the set of members, Q. *LeaderTransfer*(x, C, y) sends appropriate messages to transfer the leadership of the cluster C from current leader, x, to new leader, y. The *LeaderTransfer* message can be sent as a regular *HeartBeat* message with appropriate flags set to indicate the transfer.

Procedure : *ClusterMerge*(C)
  $\{ \ |C| < k$ *and* L_i *is the layer to which* C *belongs* $\}$
$l \leftarrow$ *Ldr*(C)
Find y s.t. $\text{dist}(l, y) < \text{dist}(h, x), x, y \in$ *Cl*$_{i+1}(l)$
ClusterMergeRequest(l, y, L_i)
LeaderTransfer(l, C, y)

Fig. 8. Cluster merge operation is invoked by l, the leader of the cluster C. The size of C is $< k$. l finds y, the leader of another cluster in layer L_i and sends the *ClusterMergeRequest* message.

by all other members in the cluster if they do not receive a message from x for a configurable number of *HeartBeat* message intervals.

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. A cluster that exceeds the cluster size upper bound, $3k - 1$ is split into two clusters each of which has at least $\lfloor 3k/2 \rfloor$ members. This is described in pseudo-code in Figure 7.

For correct operation of the protocol, we could have chosen the cluster size upper bound to be any value $\geq 2k - 1$. However, if $2k - 1$ was chosen as the upper bound, then the cluster would require to split when it exceeds this upper bound (i.e. reaches the size $2k$). Subsequently, an equal-sized split would create two clusters of size k each. However, a single departure from any of these new clusters would violate the size lower bound and require a cluster merge operation to be performed. Choosing a larger upper bound (e.g. 3k-1) avoids this problem. When the cluster exceeds this upper bound, it is split into two clusters of size at least $\lfloor 3k/2 \rfloor$, and therefore, requires at least $\lceil k/2 \rceil$ member departures before a merge operation needs to be invoked.

Procedure : *ClusterRefine*(z)
  $\{ \ L_i$ *is highest layer to which* z *belongs* $\}$
$l \leftarrow$ *Ldr*(*Cl*$_i(z)$); $C \leftarrow$ *Cl*$_{i+1}(l)$
Find y s.t. $\text{dist}(z, y) < \text{dist}(z, x), x, y \in C$
if $(y \neq l)$
  LeaveCluster(z, l, L_i)
  JoinCluster(z, y, L_i)
endif

Fig. 9. Cluster refine operation by member z which belongs to layers $L_0, \ldots, L_i$ and no other higher layer. If it finds another appropriate cluster in the same layer, it leaves its current cluster and joins the other cluster. *LeaveCluster*(x, y, L_j) sends an appropriate message from a departing member x to its cluster leader y in layer L_j.

The cluster leader initiates this 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) but with an additional size constraint. We use an approximation strategy — the leader splits the current cluster into two clusters, each of size at least $\lfloor 3k/2 \rfloor$, 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,(in layer L_i) with leader l, falls below k, l initiates a cluster merge operation (shown in pseudo-code in Figure 8). Note, l itself belongs to a layer L_{i+1} cluster, *Cl*$_{i+1}(l)$. l chooses its closest cluster-peer, y, in *Cl*$_{i+1(l)}$. y is also the leader of a layer L_i cluster, *Cl*$_i(y)$. l initiates the merge operation of C with *Cl*$_i(y)$ by sending a *ClusterMergeRequest* message to y. l updates the members of C with this merge information. y similarly updates the members of *Cl*$_i(y)$. Following the merge, l removes itself from layer L_{i+1} (i.e. from cluster *Cl*$_{i+1}(l)$.

When a member is joining a layer, it may not always be able to locate the closest cluster in that layer (e.g. due to lost join query or join response, etc.) and instead attaches to some other cluster in that layer. Therefore, each member, z, 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 the closest member (say y) to itself in the super-cluster. If y is not the leader of the L_i cluster to which z belongs then such an inaccurate attachment is detected. In this case, z leaves its current layer L_i cluster and joins the layer L_i cluster of which y is the leader. This cluster refinement process is shown in pseudo-code in Figure 9.

C. Host Departure and Leader Selection

When a host x leaves the multicast group, it sends a *Remove* message to all clusters to which it is joined. This is a graceful-leave. However, if x fails without being able to send out this message all cluster peers of x detects this departure through non-receipt of the periodic *HeartBeat* message from x. If x was a leader of a cluster, this triggers a new leader selection in the cluster. Each remaining member, y, of the cluster independently select a new leader of the cluster, depending on who y estimates to be the center among these members. Multiple leaders are re-conciled into a single leader of the cluster through exchange of *LeaderTransfer* message between the two candidate leaders, when the multiplicity is detected.

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 restores the hierarchy invariants and reconciles the cluster view for all members.

IV. Experimental Methodology

We have analyzed the performance of the NICE protocol using detailed simulations and a wide-area implementation. In the simulation environment, we compare the performance of NICE to three other schemes: multi-unicast, native IP-multicast using the Core Based Tree protocol [2], and the Narada application-layer multicast protocol (as given in [9]). In the Internet experiments, we benchmark the performance metrics against direct unicast paths to the member hosts.

Clearly, native IP multicast trees will have the least (unit) stress, since each link forwards only a single copy of each data packet. Unicast paths have the lowest latency and so we consider them to be of unit stretch [4]. They provide us a reference against which to compare the application-layer multicast protocols.

A. Data Model

In all these experiments, we model the scenario of a data stream source multicasting to the group. We chose a single end-host, uniformly at random, to be the data source generating a constant bit rate data. Each packet in the data sequence, effectively, 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] There are some recent studies [19], [1] to show that this may not always be the case; however, we use the native unicast latency as the reference to compare the performance of the other schemes.

B. 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. In our experiments, we modeled member departures from the group as *ungraceful departures*, i.e. members fail instantly and are unable to send appropriate leave messages to their existing peers on the topology. Therefore, in transience, 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 study these effects, we measured the fraction of hosts that correctly receive the data packets sent from the source as the group membership changed. 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 mean, variance and the distribution of the control bandwidth overheads at both routers and end hosts.

V. Simulation Experiments

We have implemented a packet-level simulator for the four different protocols. Our network topologies were generated using the Transit-Stub graph model, using the GT-ITM topology generator [4]. 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 routers, chosen 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 lossless 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; in our implementation, the performance is also affected by other factors such as additional link latencies due to congestion and drops due to cross-traffic congestion.

A. Our implementation of Narada

We have implemented the entire Narada protocol from the description given in [9]. We did not implement the Narada high bandwidth extensions described in [8].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 [9], 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 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.

B. Simulation Results

We have simulated a wide-range of topologies, group sizes, member join-leave patterns, and protocol parameters. For NICE, we set the cluster size parameter, k, to 3 in all of the experiments presented here. 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.
- NICE protocol demonstrates that it is possible to provide these performance with orders of magnitude lower control overhead for groups of size > 32.

We begin with results from a representative experiment that captures all the of different aspects comparing the various protocols.

1) Simulation Representative Scenario: This experiment has two different phases: a join phase and a leave phase. In the join phase a set of 128 members[5] 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 until 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 until 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 16 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 rates 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

[5] We show results for the 128 member case because that is the group size used in the experiments reported in [9]; NICE performs increasingly better with larger group sizes.

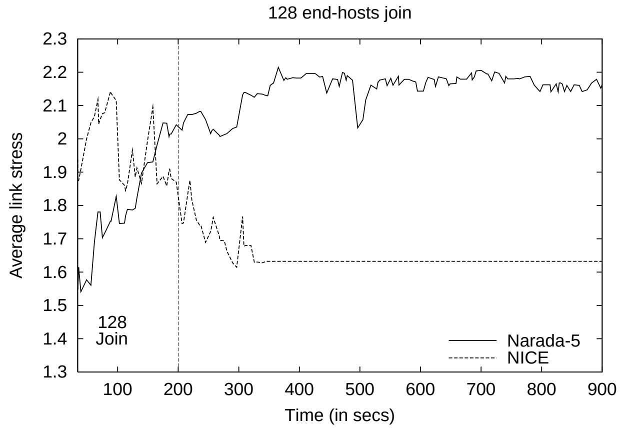

Fig. 10. Average link stress (simulation)

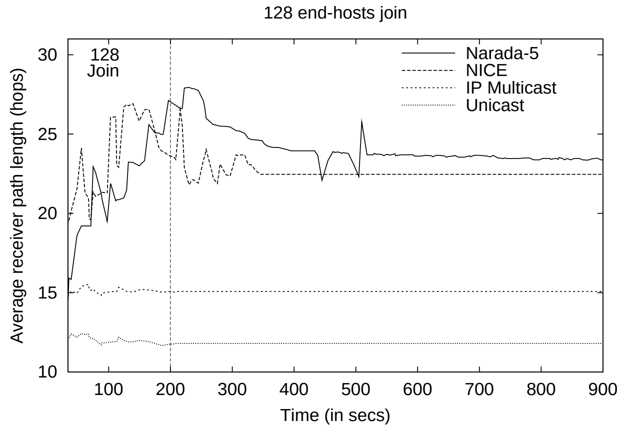

Fig. 11. Average path length (simulation)

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 [9]; 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 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 V-B.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.

The distribution of stress on links for the multi-unicast scheme has a significantly large tail (e.g. links close to the source has a stress of 127). This should be contrasted with better stress distribution for both NICE and Narada. 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 has a heavier-tail than NICE. 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.

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

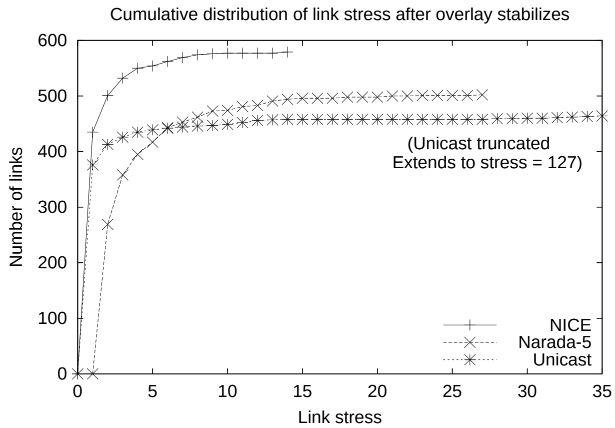

Fig. 12. Stress distribution (simulation)

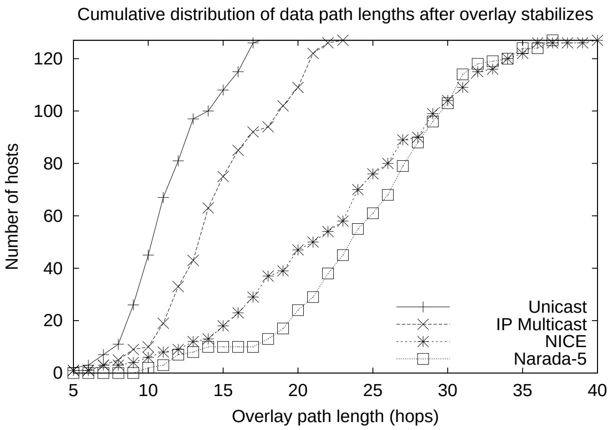

Fig. 13. Path length distribution (simulation)

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. We also ran the same experiment with the 30 second refresh period for Narada. The lower refresh period caused significant disruptions on 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 can be expected, 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. Note that the NICE control traffic includes all protocol messages, including messages for cluster formation, cluster splits, merges, layer promotions, and leader elections.

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. In Table I, 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. Note, that the Narada protocol involves an aggregate control overhead of $O(N^2)$, where N is the size of the group. Therefore, 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.

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 topology-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 the Narada-30 version has poor failure recovery performance, but is much more efficient

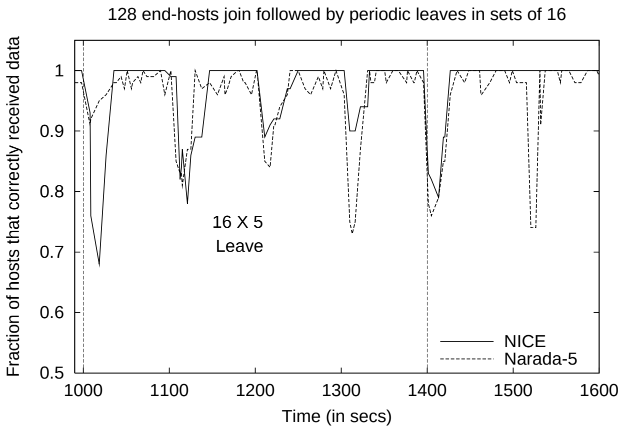

Fig. 14. Fraction of members that received data packets over the duration of member failures. (simulation)

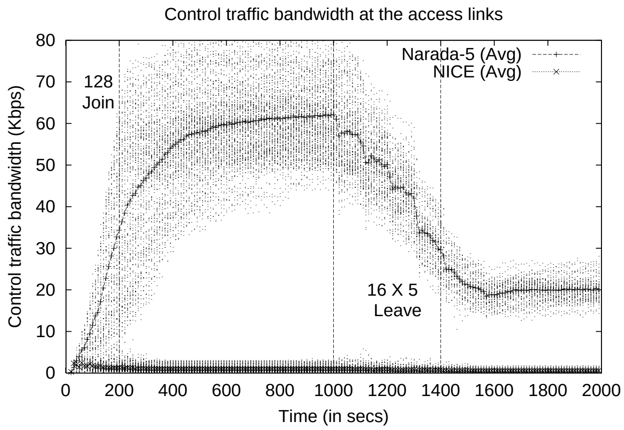

Fig. 15. Control bandwidth required at end-host access links (simulation)

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 I

DATA PATH QUALITY AND CONTROL OVERHEADS FOR VARYING MULTICAST GROUP SIZES (SIMULATION)

(specifically 5 *times* less overhead with groups of size 128) than the Narada-5 version. The *HeartBeat* messages in NICE were still sent at 5 second intervals.

For the NICE protocol, the worst case control overheads at members increase logarithmically with increase in group size. The control overheads at routers (shown in Table I), show a similar trend. Thus, although we experimented with upto 2048 members in our simulation study, we believe that our protocol scales to even larger groups.

VI. WIDE-AREA IMPLEMENTATION

We have implemented the complete NICE protocol and experimented with our implementation over a one-month period, with 32 to 100 member groups distributed across 8 different sites. Our experimental topology is shown in Figure 16. The number of members at each site was varied between 2 and 30 for different experiments. For example, for the 32 member experiment reported in this section, we had 2 members each in sites B, G and H, 4 each at A, E and F, 6 in C and 8 in D. Unfortunately, experiments with much larger groups were not feasible on our testbed. However, our implementation results for protocol overheads closely match our simulation experiments, and we believe our simulations provide a reasonable indication of how the NICE implementation would behave with larger group sizes.

A. Implementation Specifics

We have conducted experiments with data sources at different sites. In this paper, we present a representative set of the experiments where the data stream source is located at site C in Figure 16. In the figure, we also indicate the typical direct unicast latency (in milliseconds) from the site C, to all the other sites. These are estimated one-way latencies obtained using a sequence of application layer (UDP) probes. Data streams were sent from the source host at site C, to all other hosts, using the NICE overlay topology. For our implementation, we experimented with different *HeartBeat* rates; in the results presented in this section, we set the *HeartBeat* message period to 10 seconds.

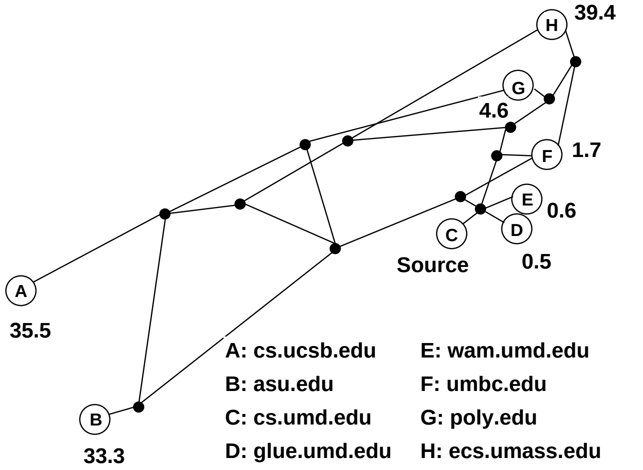

Fig. 16. Internet experiment sites and direct unicast latencies from C

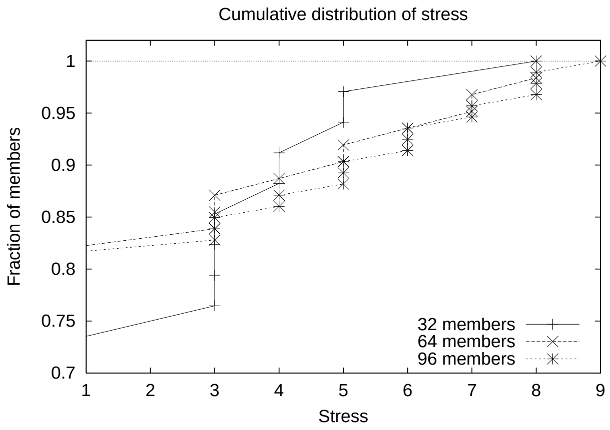

Fig. 17. Stress distribution (testbed)

In our implementation, we had to estimate the end-to-end latency between hosts for various protocol operations, including member joins, leadership changes, etc. We estimated the latency between two end-hosts using a low-overhead estimator that sent a sequence of application-layer (UDP) probes. We controlled the number of probes adaptively using observed variance in the latency estimates. Further, instead of using the raw latency estimates as the distance metric, we used a simple binning scheme to map the raw latencies to a small set of equivalence classes. Specifically, two latency estimates were considered equivalent if they mapped to the same equivalence class, and this resulted in faster convergence of the overlay topology. The specific latency ranges for each class were 0-1 ms, 1-5 ms, 5-10 ms, 10-20 ms, 20-40 ms, 40-100 ms, 100-200 ms and greater than 200 ms.

To compute the stretch for end-hosts in the Internet experiments, we used the ratio of the latency from between the source and a host along the overlay to the direct unicast latency to that host. In the wide-area implementation, when a host A receives a data packet forwarded by member B along the overlay tree, A immediately sends back a overlay-hop acknowledgment back to B. B logs the round-trip latency between its initial transmission of the data packet to A and the receipt of the acknowledgment from A. After the entire experiment is done, we calculated the overlay round-trip latencies for each data packet by adding up the individual overlay-hop latencies available from the logs at each host. We estimated the one-way overlay latency as half of this round trip latency. We obtained the unicast latencies using our low-overhead estimator immediately after the overlay experiment terminated. This guaranteed that the measurements of the overlay latencies and the unicast latencies did not interfere with each other.

B. Implementation Scenarios

The Internet experiment scenarios have two phases: a join phase and a rapid membership change phase. In the join phase, a set of member hosts randomly join the group from the different sites. The hosts are then allowed to stabilize into an appropriate overlay delivery tree. After this period, the rapid membership change phase starts, where host members randomly join and leave the group. The average member lifetime in the group, in this phase was set to 30 seconds. Like in the simulation studies, all member departures are ungraceful and allow us to study the worst case protocol behavior. Finally, we let the remaining set of members to organize into a stable data delivery tree. We present results for three different groups of size of 32, 64, and 96 members.

Data Path Quality

In Figure 17, we show the cumulative distribution of the stress metric at the group members after the overlay stabilizes at the end of the join phase. For all group sizes, typical members have unit stress (74% to 83% of the members in these experiments). The stress for the remaining members vary between 3 and 9. These members are precisely the cluster leaders in the different layers (recall that the cluster size lower and upper bounds for these experiments is 3 and 9, respectively). The stress for these members can be reduced further by using the high-bandwidth data path enhancements, described in the Appendix. For larger groups, the number of members with higher stress (i.e. between 3 and 9 in these experiments) is more, since the number of clusters (and hence, the number of cluster leaders) is more. However, as expected, this increase is only logarithmic in the group size.

In Figure 18, we plot the cumulative distribution of the stretch metric. Instead of plotting the stretch value for each

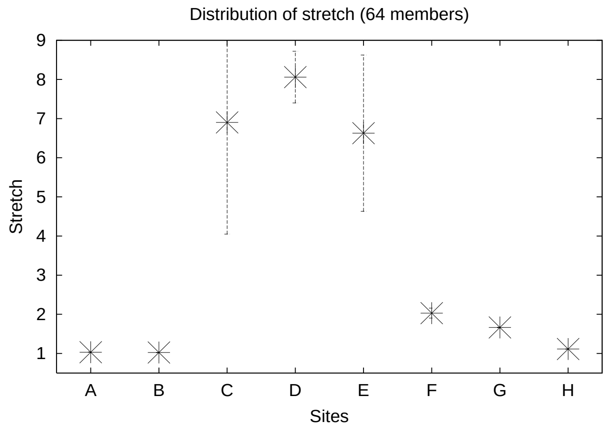

Fig. 18. Stretch distribution (testbed)

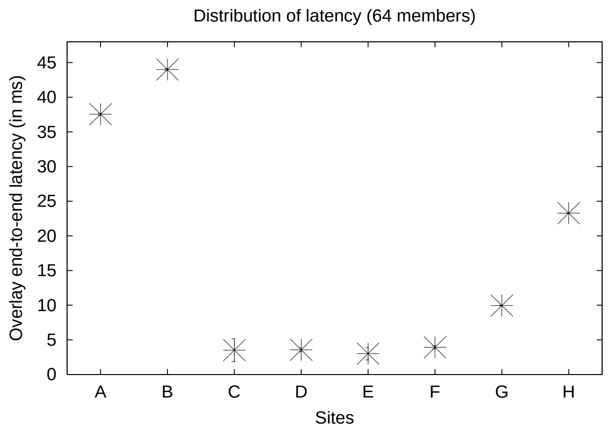

Fig. 19. Latency distribution (testbed)

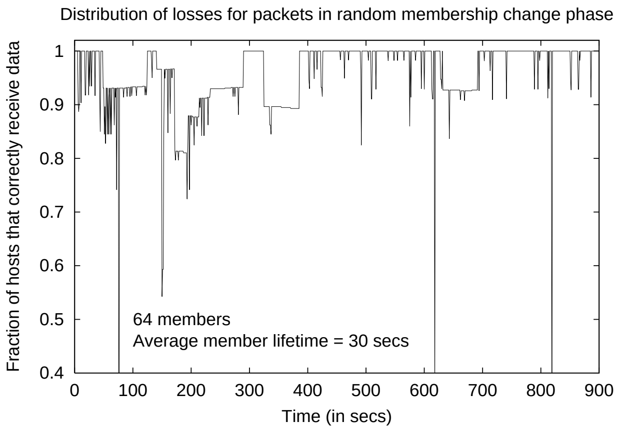

Fig. 20. Fraction of members that received data packets as group membership continuously changed (testbed)

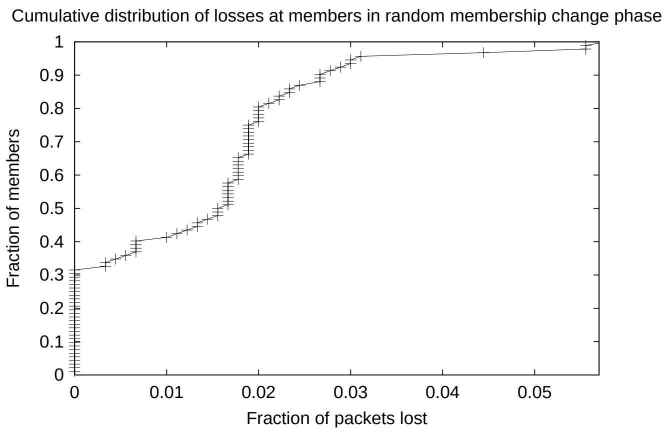

Fig. 21. Cumulative distribution of fraction of packets lost for different members out of the entire sequence of 900 packets during the rapid membership change phase (testbed)

single host, we group them by the sites at which there are located. For all the member hosts at a given site, we plot the mean and the 95% confidence intervals. Apart from the sites C, D, and E, all the sites have near unit stretch. However, note that the source of the data streams in these experiments were located in site C and hosts in the sites C, D, and E had very low latency paths from the source host. The actual end-to-end latencies *along the overlay paths* to all the sites are shown in Figure 19. For the sites C, D and E these latencies were 3.5 ms, 3.5 ms and 3.0 ms respectively. Therefore, the primary contribution to these latencies are packet processing and overlay forwarding on the end-hosts themselves.

In Table II, we present the mean and the maximum stretch for the different members, that had direct unicast latency of at least 2 ms from the source (i.e. sites A, B, G and H), for all the different sizes. The mean stretch for all these sites are low. However, in some cases we do see relatively large worst case stretches (e.g. in the 96-member experiment there was one member that for which the stretch of the overlay path was 4.63).

Failure Recovery

In this section, we describe the effects of group membership changes on the data delivery tree. To do this, we observe how successful the overlay is in delivering data during changes to the overlay topology. We measured the number of correctly received packets by different (remaining) members during the rapid membership change phase of the experiment, which begins after the initial member set has stabilized into the appropriate overlay topology. This phase lasts for 15 minutes. Members join and leave the grou at random such that the average lifetime of a member in the group is 30 seconds.

In Figure 20 we plot over time the fraction of members that successfully received the different data packets. A total of 30 group membership changes happened over the duration. In Figure 21 we plot the cumulative distribution of packet losses seen by the different members over the entire 15 minute duration. The maximum number of packet losses seen by a member was 50 out of 900 (i.e. about 5.6%), and 30% of the members did not en-

Group Size	Stress Mean	Stress Max.	Stretch Mean	Stretch Max.	Control overheads (Kbps) Mean	Control overheads (Kbps) Max.
32	1.85	8.0	1.08	1.61	0.84	2.34
64	1.73	8.0	1.14	1.67	0.77	2.70
96	1.86	9.0	1.04	4.63	0.73	2.65

TABLE II

AVERAGE AND MAXIMUM VALUES OF OF THE DIFFERENT METRICS FOR DIFFERENT GROUP SIZES(TESTBED)

counter any packet losses. Even under this rapid changes to the group membership, the largest continuous duration of packet losses for any single host was 34 seconds, while typical members experienced a maximum continuous data loss for only two seconds — this was true for all but 4 of the members. These failure recovery statistics are good enough for use in most data stream applications deployed over the Internet. Note that in this experiment, only three individual packets (out of 900) suffered heavy losses: data packets at times 76 s, 620 s, and 819 s were not received by 51, 36 and 31 members respectively.

Control Overheads

Finally, we present the control traffic overheads (in Kbps) in Table II for the different group sizes. The overheads include control packets that were sent as well as received. We show the average and maximum control overhead at any member. We observed that the control traffic at most members lies between 0.2 Kbps to 2.0 Kbps for the different group sizes. In fact, about 80% of the members require less than 0.9 Kbps of control traffic for topology management. More interestingly, the average control overheads and the distributions do not change significantly as the group size is varied. The worst case control overhead is also fairly low (less than 3 Kbps).

VII. RELATED WORK

A number of other projects have explored implementing multicast at the application layer. They can be classified into two broad categories: mesh-first (Narada [9], Gossamer [6]) and tree-first protocols (Yoid [11], ALMI [14], Host-Multicast [21]). Yoid and Host-Multicast 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 Overcast protocol [13] organizes a set of 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 [7] provides support for reliable multicast data delivery to end-hosts using a set of similar 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.

Some other recent projects (Chord [20], Content Addressable Networks (CAN) [16], Tapestry [22] and Pastry [18]) have also addressed the scalability issue in creating application layer overlays, and are therefore, closely related to our work. CAN defines a virtual d-dimensional Cartesian coordinate space, and each overlay host "owns" a part of this space. In [17], the authors have leveraged the scalable structure of CAN to define an application layer multicast scheme, in which hosts maintain $O(d)$ state and the path lengths are $O(dN^{1/d})$ application level hops, where N is the number of hosts in the network. Pastry [18] is a self-organizing overlay network of nodes, where logical peer relationships on the overlay are based on matching prefixes of the node identifiers. Scribe [5] is a large-scale event notification infrastructure that leverages the Pastry system to create groups and build efficient application layer multicast paths to the group members for dissemination of events. Being based on Pastry, it has similar overlay properties, namely $(2^b - 1)\log_{2^b} N$ state at members, and $O(\log_{2^b} N)$ application level hops between members [6]. Bayeux [23] 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 [22]. 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. [15] for locating and routing to named objects in a network. Therefore, hosts in Bayeux maintain $O(b \log_b N)$ state and end-to-end overlay paths have $O(\log_b N)$ application level hops. As discussed in Section II-C, our proposed NICE protocol incurs an amortized $O(k)$ state at members and the end-to-end paths between members have $O(\log_k N)$ application level hops. Like Pastry and Tapestry, NICE also chooses overlay peers based on network locality which leads to low stretch end-to-end paths.

We summarize the above as follows: For both NICE and CAN-multicast, members maintain constant state for other members, and consequently exchange a constant amount of periodic refreshes messages. This overhead is logarithmic for Scribe and Bayeux. The overlay paths for

[6] b is a small constant.

NICE, Scribe, and Bayeux have a logarithmic number of application level hops, and path lengths in CAN-multicast asymptotically have a larger number of application level hops. Both NICE and CAN-multicast use a single well-known host (the RP, in our nomenclature) to bootstrap the join procedure of members. The join procedure, therefore, incurs a higher overhead at the RP and the higher layers of the hierarchy than the lower layers. Scribe and Bayeux assume members are able find different "nearby" members on the overlay through out-of-band mechanisms, from which to bootstrap the join procedure. Using this assumption, the join overheads for a large number of joining members can be amortized over the different such "nearby" bootstrap members in these schemes.

VIII. 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. While the focus of this paper has been low-bandwidth data stream applications, our scheme is generalizable to different applications by appropriately choosing data paths and metrics used to construct the overlays. We believe that the results of this paper are a significant first step towards constructing large wide-area applications over application-layer multicast.

IX. Acknowledgments

Srinivas Parthasarathy implemented a part of the Narada protocol used in our simulation experiments. We thank Kevin Almeroth, Lixin Gao, Jorg Liebeherr, Steven Low, Martin Reisslein and Malathi Veeraraghavan for providing us with user accounts at the different sites for our wide-area experiments and Peter Druschel for providing many useful comments on a previous version of this paper.

References

[1] D. Andersen, H. Balakrishnan, M. Frans Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of 18th ACM Symposium on Operating Systems Principles*, Oct. 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] S. Banerjee and B. Bhattacharjee. Scalable Secure Group Communication over IP Mulitcast. In *Proceedings of Internation Conference on Network Protocols*, Nov. 2001.

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

[5] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 2002. To appear.

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

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

[8] 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*, Aug. 2001.

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

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

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

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

[13] 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*, Oct. 2000.

[14] 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.

[15] 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*, June 1997.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM Sigcomm*, Aug. 2001.

[17] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of 3rd International Workshop on Networked Group Communication*, Nov. 2001.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.

[19] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: A Case for Informed Internet Routing and Transport. *IEEE Micro*, 19(1), Jan. 1999.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H.Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM Sigcomm*, Aug. 2001.

[21] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proceedings of IEEE Infocom*, June 2002.

[22] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical report, UCB/CSD-01-1141, University of California, Berkeley, CA, USA, Apr. 2001.

[23] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Eleventh International Workshop on*

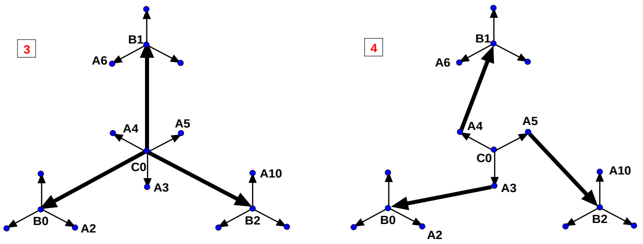

Fig. 22. Data path enhancements using delegation.

Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001), 2001.

APPENDIX

DATA PATH ENHANCEMENTS FOR HIGH BANDWIDTH APPLICATIONS

The basic data path in NICE imposes more data forwarding responsibility onto the cluster leaders. As a consequence, members that are joined to higher layers are cluster leaders in all the lower layers that they are joined to. Therefore, they are required to forward higher volume of data than those members that are joined to only the lower layers. This data forwarding path, is therefore, not suited for high bandwidth applications (e.g. video distribution). 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. In this paper, we only explain data path delegation, assuming that data is originating from the leader of the highest cluster in the topology. However, the same delegation mechanism is equally applicable for data originating from any member (with minor modifications).

Consider a host h that belongs to layers, $L_0, \ldots, L_i$ and no other higher layer. The corresponding clusters in these layers are: $Cl_0(h), \ldots, Cl_i(h)$. In the basic data path (described in Section II-B), h receives the data from the leader, p, of cluster $Cl_i(h)$, i.e. its topmost layer. It is also responsible for forwarding data to all the members in the clusters $Cl_0(h), \ldots, Cl_{i-1}(h)$, i.e. the clusters in the remaining layers.

In the enhanced data path, h forwards data to only the other members of $Cl_0(h)$, i.e. its cluster in the lowest layer (L_0). Additionally, it delegates the responsibility of forwarding data to members in $Cl_j(h)$ to members in $Cl_{j-1}(h)$, for all $1 \leq j \leq i-1$. Since the cluster sizes are bounded between k and $3k-1$, each member of $Cl_{j-1}(h)$ gets a *delegated* forwarding responsibility to at most three members in $Cl_j(h)$. Only the cluster leader can delegate forwarding responsibility to another member of its cluster. A member that belongs to multiple layers belongs to only a single cluster in each layer and is the leader of the respective clusters in all but one layer. It is not a leader in its topmost cluster. Therefore, each member can be delegated forwarding responsibilities for at most 3 new peers. The member, h receives data from a member, q to which p (the leader of its topmost cluster, $Cl_i(h)$) has delegated the forwarding responsibility.

This data path transformation is illustrated with an example in Figure 22. Consider the basic data path with host C_0 as the source (Panel 3). Host C_0 is the leader of both its L_0 and L_1 clusters. Therefore, in the basic data path, it is required to forward data to the other members both its clusters (A_3, A_4 and A_5 in layer L_0, and B_0, B_1 and B_2 in layer L_1). In the enhanced data path (Panel 4), it delegates the other members of its L_0 cluster to forward data to the other members of its L_1 cluster. In particular, it sets up the new data path peers as: $\langle A_3 \rightarrow B_0 \rangle$, $\langle A_4 \rightarrow B_1 \rangle$, and $\langle A_5 \rightarrow B_2 \rangle$. Members which are not leaders of their L_1 clusters, i.e. B_0, B_1 and B_2 now receive data not from the cluster leader (i.e. C_0), and instead receive data from the members delegated by C_0 as described.

Any member, in the enhanced data path, forwards data to all members of only one of its clusters (i.e. its L_0 cluster), and additionally may be delegated to forward data to two other members. This the total number of data path peers for any member in this enhanced data path is bounded by $3k$, a constant that depends on the cluster size. However, the number of application-level hops between any pair of members on the overlay is still bounded by $O(\log N)$.