A Comparative Study of Application Layer Multicast Protocols

Suman Banerjee, Bobby Bhattacharjee

Abstract—**Due to the sparse deployment of IP multicast in the Internet today, some researchers have proposed application layer multicast as a new approach to implement wide-area multicast services. In this approach multicast functionality is implemented at the end-hosts instead of network routers. Unlike network-layer multicast, application layer multicast requires no infrastructure support and can be easily deployed in the Internet. In this paper, we describe a set of application layer multicast protocols that have been proposed in recent literature, classify them based on some properties and present a comparison of performance and applicability of these schemes.**

I. Introduction

Multicasting is defined as the distribution of content to more than one host. In the Internet architecture, the network layer implements the data forwarding functionality between two hosts that are not located in the same Local Area Network (LAN). In tune with this approach, Deering [7] proposed the IP Multicast architecture, where the multicast functionality was added to the IP layer. In the IP Multicast architecture, the routers of the network distributedly define a data delivery tree. As multicast packets flow on this tree, they are appropriately replicated by the routers at the different branch points of the tree. IP Multicast is the most efficient way to perform group data distribution, as it is able to reduce packet replication on the wide-area network to the minimum necessary.

However, more than a decade after its initial proposal, deployment of IP Multicast has been limited and sparse due to a variety of technical and non-technical reasons. First, IP Multicast requires routers to maintain per group state (and in some proposals per source state in for each multicast group). The routing and forwarding table at the routers now need to maintain an entry corresponding to each unique multicast group address. However, unlike unicast addresses, these multicast group addresses are not easily aggregatable. This increases the overheads and complexities at the routers. Second, there is a dearth of experience with additional mechanisms like reliability and congestion control on top of IP Multicast, which makes the ISPs wary of enabling multicasting at the network layer. Although there exists proposals for such mechanisms over IP Multicast (e.g. (SRM [8] and RMTP [12] for reliability and MTCP [17] and PGMCC [18] for congestion control), the impact of these solutions on the wide-area Internet is not clear. Congestion control for multicast applications acquires far greater importance than the unicast case, and therefore, needs to be well understood before wide-scale deployment. Third, the pricing model for multicast traffic is not yet well-defined.

Therefore some researchers in the recent past have revisited the issue whether the network layer is necessarily the best layer for implementing multicast functionality and have proposed application layer multicast as an alternate technique for multicasting. As the name suggests, in application layer multicast, the multicasting functionality is implemented at the application layer, i.e. at the end-hosts instead of the network routers.

The basic idea of application layer multicast is shown in Figure 1. Unlike network layer multicast (Panel 0) where data packets are replicated at routers inside the network, in application layer multicast data 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 may send data multiple times over the same link, they are less efficient than network layer multicast. There are multiple intuitive metrics of "goodness" to evaluate the performance of an application layer multicast scheme.

1) *Quality of the data path:* is evaluated using two metrics, which are:
   Stress: This metric is defined per link or router of the topology and counts the number of identical packets sent by the protocol over that link or node. For network layer multicast there is no redundant packet replication and hence in this case, the stress metric is one at each link or node of the network.
   Stretch: This metric is defined per-member and is the ratio of the path length along the overlay from the source to the member to the length of the direct unicast path. Clearly, a sequence of direct unicasts from the source to all the other members (Panel 1, Figure 1) has unit stretch for each member.
2) *Control Overheads:* Each member on the overlay exchanges refresh messages with all its peers on the

The authors are with the Department of Computer Science, University of Maryland, College Park, MD 20742, USA. Emails: {suman,bobby}@cs.umd.edu

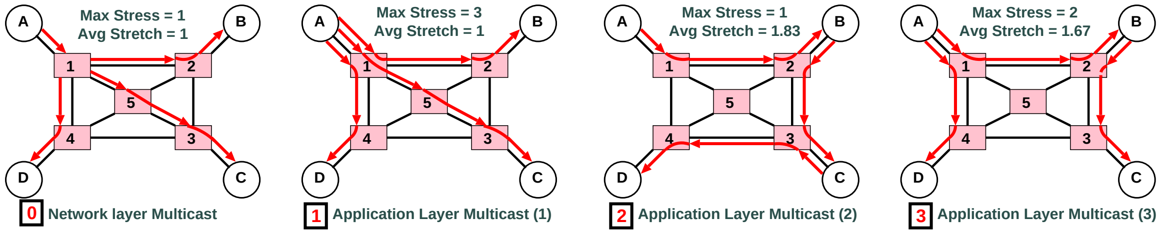

Fig. 1. Network-layer and application layer multicast. Square nodes are routers, and circular nodes are end-hosts.

overlay. These messages constitute the control overheads at the different routers, links and members of the multicast group. The control overheads are an important metric to consider from the scalability standpoint.

While these are some of the more important metrics for application layer multicast protocols, it is clearly not an exhaustive list. Other metrics of interest include degree distribution of members on the data delivery path, bandwidth requirements at the access links of the members, etc.

Different application layer multicast protocols will create overlay paths that have different properties for these different metrics. In Figure 1, we show three example application layer multicast overlays on the same topology of routers and hosts. Let us assume that each link on the topology is of unit length. Panel 1 shows the overlay corresponding to a *sequence of direct unicasts* from the source (A) to all the other members. In this case, the stretch to each member is unity (since the direct unicast paths are used). Link $\langle A, 1\rangle$ experiences a stress of 3, while all other links experience unit stress. In general, for a group of N members, using a sequence of direct unicasts is one extreme case where the maximum stress at a link is $O(N)$ (at the data source) and the average stretch of members 1. Also, the source peers with all the other members in the multicast group and exchanges refresh messages with them. Therefore the worst case control overhead in this scheme is $O(N)$.

Panel 2 shows the overlay corresponding to *ring multicast.* This is the other extreme case where the maximum stress is 1 while the average stretch at members is $O(N)$. The worst case control overhead at members is also a constant.

Finally, Panel 3 shows another configuration of the overlay, which is an intermediate between the two extremes.

There has been a number of application layer multicast protocols proposed in the literature and it is not possible to describe all of them, in detail, in this survey. Instead we classify the set of proposed application layer multicast schemes into three different categories and only describe a few representative protocols from each of these categories. Next, we present a comparative study of these different schemes with respect to the different metrics of interest.

II. Application Layer Multicast Approaches

All application layer multicast protocols organize the group members into two topologies, namely the control topology and the data topology. Members that are peers on the control topology exchange periodic refresh messages to identify and recover from "ungraceful" departures from the group. (An ungraceful departure is one when the member departs from the group without informing its peers through control messages.) The data topology is usually a subset of the control topology and identifies the data path for a multicasted packet on the overlay. In fact the data topology is a tree, while the control topology has greater connectivity between members. Therefore, in many protocols the control topology is called a mesh and the data topology is called a tree.

Depending on the sequence of construction of the control and data topologies, we classify classify the different proposed application layer multicast techniques into three different categories — mesh-first, tree-first and implicit approaches.

In the mesh-first approach, group members first distributedly organize themselves into the overlay mesh topology. Multiple paths exist on the mesh between a pair of members. Each member participates in a routing protocol on this control topology to distributedly compute unique overlay paths to every other member. A source-specific tree rooted at any member can then be created using the well-known Reverse Path Forwarding (RPF) based construction used by many IP multicast protocols e.g. DVMRP [20].

In contrast, protocols based on the tree-first approach distributedly construct a shared data delivery tree first directly. Subsequently, each member discovers a few other members of the multicast group that are not its neighbors on the overlay tree and establishes and maintains additional control links to these members. This enhanced over-

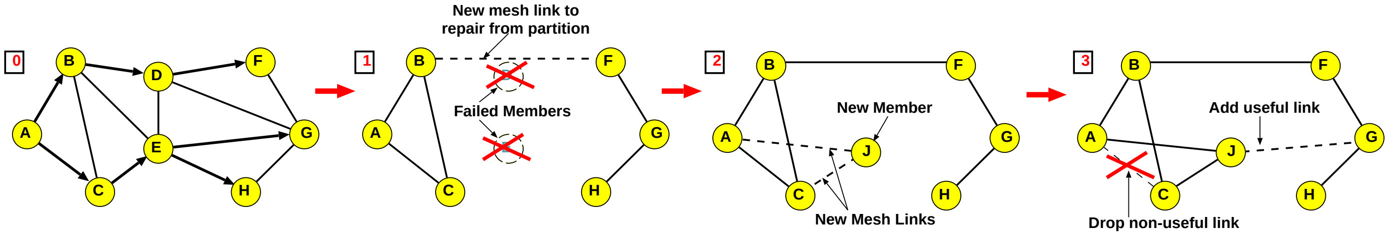

Fig. 2. Control and data paths in Narada. Neighbors on the control path (mesh) are connected by edges. In Panel 0, the thicker edges (marked with arrows) indicate the multicast data path when A is the source of the data. These edges are also part of the mesh. In Panel 1, two members, D and E, leave the group which leads to a mesh partition. The remaining members repair this partition. In Panel 2, a new member joins the mesh and sets up mesh-neighbor links with two randomly chosen members. Periodically members evaluate the utility of different links on the mesh. In Panel 3, a non-useful link is deleted from the mesh, and a potentially useful link is added to it.

lay (the data delivery tree with the additional control links) is the control topology in the tree-first approach.

Protocols using the implicit approach create a control topology with some specific properties. The data delivery path is *implicitly* defined on this control topology by some packet forwarding rule which leverages the specific properties of the control topology to create loop-free multicast paths. Thus, in the implicit approach the mesh and the tree are simultaneously defined by the protocol, and no additional member interactions are needed to generate one from the other. All the example protocols developed using the implicit approach have been specifically designed to scale to multicast groups with large number of members.

In the next three sections, we describe representative protocols from each of these approaches in turn.

III. Mesh-first Approach

In this section, we describe the Narada protocol [6], [5] as an example of the mesh-first approach.

A. Narada

The Narada protocol [6], [5] was one of the first application layer multicast protocols that demonstrated the feasibility of implementing multicast functionality at the application-layer.

Narada defines a special designated host, called the Rendezvous Point (RP), that is used to boot-strap the join procedure of a new member. In fact, all application layer multicast protocols use an entity equivalent to the RP in Narada to initiate the join mechanism. In this paper, we use this nomenclature to denote the boot-strapping host for all the application layer multicast protocols.

Mesh construction: When a new member wants to join the multicast group, it first obtains a list of group members that are already joined to the mesh. This information can typically be obtained from the RP, which maintains state about all members joined to the multicast group. The new member then randomly selects a subset of these members and attempts to join the mesh as neighbors of these members. The join procedure succeeds when at least one of these members accepts the new member as its mesh-neighbor.

After joining the mesh, the new member starts exchanging periodic refresh messages with its mesh-neighbors. Whenever a new member joins or an existing member leaves the group (and the mesh), this group change information is propagated through the mesh to all the other members. Thus, each member in the group keeps state about all other members that are part of the group. This information is also periodically refreshed. Distribution of such state information about each member to all other members leads to relatively high control overhead ($O(N^2)$ aggregate control overhead, where N is the group size). Therefore, the Narada protocol is effective only when the multicast group size is small. However, this was an explicit design choice for the Narada protocol, where the authors traded off high control overheads for greater robustness in recovery from member failures in some specific cases. This is described in the following example.

When members D and E simultaneously fail, the mesh partitions into two parts. As a consequence, members A, B and C stop receiving state refresh messages from members F, G and H, and vice versa. Each member then probabilistically probes the members from which it has stopped receiving refresh messages to establish new links and repair the partition (Panel 1). Note that the recovery does not require the intervention of the RP and, therefore, works even when the RP fails.

Data Delivery Path: The members of the group run a routing protocol to compute unicast paths between all pair of members on the mesh. The multicast data delivery path with any specific member as the source can be then computed using the well-known Reverse Path Forwarding check employed by IP multicast protocols (e.g. DVMRP [20]). An example of such a data path is shown

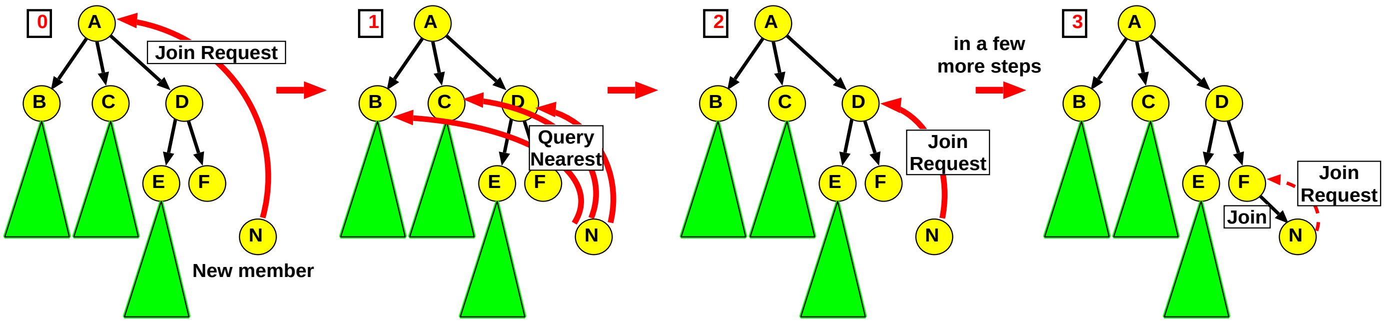

Fig. 3. Tree join procedure in HMTP. The new member, N, discovers the root, A, by querying the RP. For this example we assume that members A and D have a maximum degree of 3 (and therefore cannot support any other member as children). N recursively finds a nearby potential parent by querying down the tree.

in Panel 0, Figure 2. The specific links on the data path from source A is highlighted with thicker directed edges.

Mesh Refinement: The data delivery paths in Narada are spanning trees of the mesh. Therefore, the quality of the data delivery path (i.e. the stress and stretch properties) depends on the quality of links that are part of the mesh. When new members join, or when the mesh recovers from partitions, a random set of edges are added to the mesh. Thus, periodic refinements are made to mesh edges to improve the quality of data delivery paths. In Panel 3, Figure 2, we show such refinements. Adding the edge $\langle J, G\rangle$ is considered useful because a large number of shorter unicast paths can be created on the mesh using this new edge (for example between member sets $\{A, C, J\}$ and $\{G, H\}$, and between member sets $\{C, J\}$ and $\{F\}$). The edge $\langle A, C\rangle$ is removed from the mesh since was being used to create a single shortest path (that between members A and C). These decisions to add or drop edges from the mesh are made locally by the two end-points of the edge, through simple heuristics that compute a specific "utility" for the edge.

IV. Tree-first Approach

We describe two protocols, Yoid [9] and HMTP [21], as examples of the tree-first approach.

A. Yoid

Yoid, along with Narada, is one of the first application layer multicast protocols. Since Yoid directly creates the data delivery tree, it has a direct control over various aspects of the tree structure, e.g. the out-degree at members, the choice of tree neighbors, etc. This is in contrast to the mesh-first approach which has an indirect control on the tree structure and therefore, the quality of the data delivery tree depends on the quality of the mesh.

Tree Construction: All protocols based on the tree-first approach create a shared tree and each member is responsible for finding its appropriate parent on the tree. Each member on the tree has a degree bound, which limits the number of children on the tree it is willing to support. A new member, h, starts to find a new parent by querying the RP. The RP typically responds with a list of members that are already part of the multicast group and hence, joined to the tree. Member, h, then probes members from this list to find potential parents. A member, y can be a potential parent of x if the following two conditions hold: (a) If y is chosen as a parent of x it should not cause a loop in the tree, and (b) y should have available degree for new children. If no potential parent is found in this list, Yoid proposes different heuristics that can be used to find other potential parents on the tree. (The HMTP protocol, which we describe next, implements a specific example of such a heuristic.) If x eventually finds some potential parents, it chooses the "best" potential parent (with respect to the metric of interest) as its parent.

If x is unable to find any parent it eventually declares itself to be the root of the shared tree and informs the RP. In transience the tree may get partitioned and one member in each tree partition will declare itself to be the root. In such a case, the RP arbitrates in merging the different trees fragments. Periodically each members seeks other potential parents for better points of attachment in the shared tree. Yoid incorporates loop detection and avoidance mechanisms when members change parents in the tree.

Mesh Construction: Each member, h, on the shared tree finds a few other members at random which are not tree-neighbor's of h. These additional links along with the tree links, define the mesh. The additional mesh links are used for recovery from tree partitions.

B. HMTP

Host Multicast Tree Protocol (HMTP) is another application layer multicast protocol that uses the tree-first approach and has some similarities with the Yoid protocol.

Tree Construction: Like in Yoid, members in HMTP are responsible for finding parents on the shared tree. A joining member, h, finds its parent using the following heuristic: It first discovers the root of the shared tree by querying the RP. Starting from the root, at each level of the tree h tries to find a member, x, close to itself. If the number of children of x is less than its degree bound, h joins as a child of x. Or else it proceeds to the next level and tries to find a potential parent among the children of x. This is shown using an example in Figure 3. The new member, N probes the root, A. Since all its degree is filled, it then probes to find, D, the child of A which is closest to itself. However, since D also has no available degree, N proceeds to the next level and finds, F, the nearest child of D. F has available degree and so N joins as a child of F.

Members in HMTP maintain information about all members on its path to the root. Periodically, each member tries to find a better (i.e. closer) parent on the tree, by re-initiating the join process from some random member on its root path. Knowing the entire root path allows members to detect loops. HMTP employs a loop detection and resolution mechanism, instead of loop avoidance.

Unlike Yoid, HMTP does not explicitly create a mesh. However, each member periodically discovers and caches information about a few other members that are part of the tree. In the specific case when the RP is unavailable, the knowledge of such members is used to recover the tree from partitions.

V. Implicit Approach

In this section we describe three different protocols that use the implicit approach to create application layer multicast overlays, namely NICE, Scribe, and CAN-multicast.

A. NICE

The NICE protocol [2] arranges the set of members into a hierarchical control topology. As new members join and existing members leave the group, the basic operation of the protocol is to create and maintain the hierarchy. The hierarchy implicitly defines the multicast overlay data paths and is crucial for scalability of this protocol to large groups. The members at the bottom of the hierarchy maintain (soft) state about a constant number of other members, while the members at the top maintain such state for about $O(\log N)$ other members.

The NICE hierarchy is created by assigning members to different levels (or layers) as illustrated in Panel 0, Figure 4. Layers are numbered sequentially with the lowest layer of the hierarchy being layer zero (denoted by L_0). Members 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 members that are close to each other. 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 members 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.

The members are assigned to the different layers as follows: All members are part of the lowest layer, L_0. A distributed clustering protocol at each layer partitions these members into a set of clusters with the specified size bounds. The protocol also chooses the member which is the graph theoretic center of the cluster, to be the leader of the cluster. The cluster leaders of all the clusters in layer L_i join layer L_{i+1}.

Control and Data Topologies: The member hierarchy is used to define both the control and data overlay topologies. In the control topology, all members of each cluster peer with each other and exchange periodic refreshes between them. The data topology is defined by the following forwarding rule on the control topology:

> The source member sends a data packet to all its peers on the control topology. Consider an intermediate member, h that belongs to layers $L_0 \ldots L_j$ that receives the data packet from another member, say p. Then p and h belong to the same cluster in some layer, say L_i. Member h will forward the data packet to all other members of cluster $C_k, k \neq i$ (where C_k corresponds to its cluster in layer L_k) if and only if h is the cluster leader of C_k.

The ensuing data topologies are shown in Panels 1, 2 and 3, Figure 4 for different sources.

Join Procedure: A new member joins a L_0 cluster that is closest to itself with respect to the distance metric. Locating this L_0 cluster is approximated by a sequence of refinement steps, where the joining member starts with the topmost layer and sequentially probes one cluster in each layer to find the "closest" member in that layer.

B. CAN-Multicast

Content-Addressable Network (CAN) [15] is an application-level infrastructure where a set of end-hosts

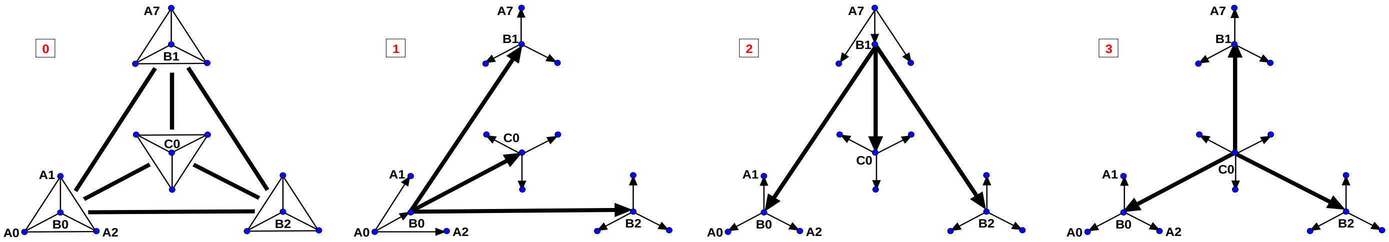

Fig. 4. NICE hierarchy and control and data topologies for a two-layer hierarchy. All A_i hosts are members of only L_0 clusters. All B_i hosts are members of both layers L_0 and L_1. The only C host is the leader of the L_1 cluster comprising of itself and all the B hosts.

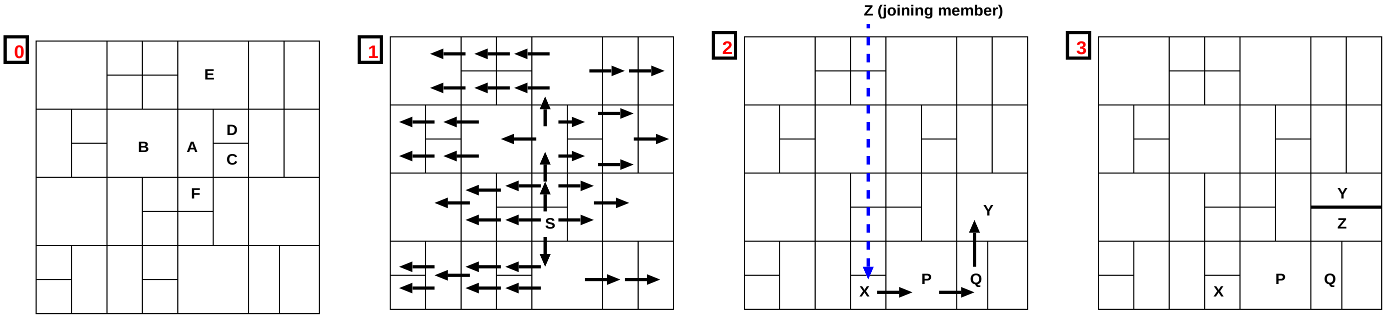

Fig. 5. Structure of a CAN built using a 2-dimensional coordinate space and the corresponding control and data topologies.

implement a distributed hash table on an Internet-wide scale. The constituent members of the CAN form a virtual d-dimensional Cartesian coordinate space, and each member "owns" a portion of this space. For example, Panel 0, Figure 5 shows a 2-dimensional coordinate space partitioned into zones by 34 CAN members of which 6 of the members (A - F) are marked in their respective zones. In [16], the authors propose an application layer multicast scheme based on the CAN architecture.

Control and data topologies: In the control topology two members peer with each other if their corresponding regions in the d-dimensional space abut each other. For example, in Panel 0, Figure 5, member A has 5 neighbors on the control topology, B, C, D, E and F. The data topology is implicitly defined by performing directed flooding on the control topology (e.g. Panel 1, Figure 5). In [16], one such data topology is defined by the following forwarding rule:

> The source forwards a data packet to all its control topology neighbors. Consider a member, h, that receives a data packet from another member, p. Members h and p are neighbors on the control topology. Member h will forward this packet to a neighbor, n, on the control topology if and only if (a) $n \neq p$ and (b) the packet has not traversed half of the coordinate space towards the dimension along with h and n abut. The latter condition ensures that packets do not loop around the CAN. Each member also maintains a packet cache to identify and discard any duplicate packets.

Join Procedure: When a new member, Z wants to join the CAN, it queries the RP to find at least one existing member, X, that is already joined to the CAN. Z picks a random point in the coordinate space (say a point which is owned by member Y). The goal of the joining member is to find the member Y which owns this randomly chosen point. This is done by routing through the CAN, as shown in Panel 2, Figure 5. The protocol then splits the zone owned by Y into two, and the ownership of one of the halves is transferred to Z (Panel 3, Figure 5).

The assignment procedure of zones of the coordinate space to members of the CAN, as described, ignores the relative distances between the members in constructing the overlay. As a consequence, neighbors on the CAN may be far apart and thus, the multicast overlay paths can have high stretch. To remedy this situation, the authors in [16] suggest the use of a "distributed binning" scheme by which members that are close to each other are assigned nearby zones in the coordinate space.

C. *Scribe*

Scribe [3] is a large-scale event notification system that uses application layer multicast to disseminate data on topic-based publish-subscribe groups. Scribe is built on

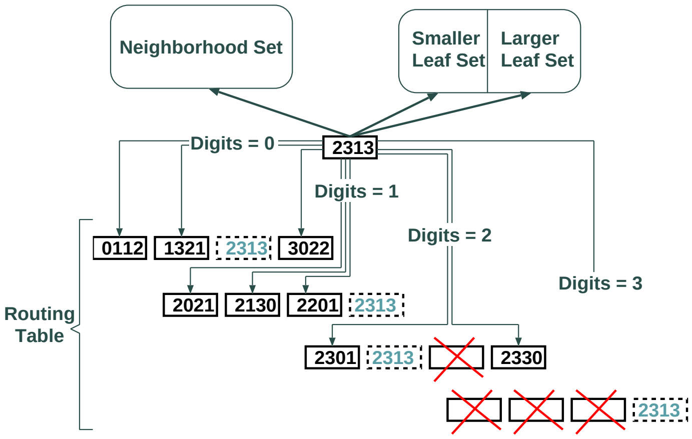

Fig. 6. Neighborhood of a Pastry member, with identifier 2313. All numbers are in Base-4. The routing table has 4 rows. Each member in the routing table share a common prefix with the member 2313. Row 2 in the routing table, for example, has members (2021, 2130 and 2201) that share the first digit (2) of their node identifiers. Additionally each of these members in the same row have a different digit at the second position (i.e. 0, 1 and 2 respectively). The fourth member in this row should have the same 1-digit prefix and the digit 3 in the second position. It is the member 2313 itself (and so it is not added to the routing table).

top of Pastry [19] which is a peer-to-peer object location and routing substrate overlaid on the Internet.

Each member in Pastry is assigned a random node identifier, which may be generated by computing cryptographic hash of the member's public key. Pastry organizes the members into an overlay in which messages can be routed from a member to any other member by knowing the node identifier of the latter. This organization is shown in Figure 6. The members are represented by rectangular boxes and their node identifiers are marked inside the box. The node identifiers are thought of as a sequence of digits with base 2^b, where b is a small constant. In the example in Figure 6, $b = 2$. In this section we refer to members by their node identifiers.

Each member has a routing table, a neighborhood set and a leaf set. The routing table for a member, h, contains information about a set of members in the overlay with which the member, h, shares a common prefix. All members in row i of the routing table of h have the same $i-1$ digits in their node identifier prefix (as shown in Figure 6). There are 2^b-1 such entries in each row. If N is the size of the node identifier space, the total number of rows in the routing table is $\log_{2^b} N$, which corresponds to the number of digits in the node identifier. If this member is not aware of any member with a matching prefix of some given size, the corresponding entry in the routing table is empty. The neighborhood set of the member, h, has members that are close to h based on the distance metric. The leaf set of h contains members for which the node identifiers are numerically close to the node identifier of h. It is partitioned into two equal-sized sets with one corresponding to numerically smaller node identifiers and the other corresponding to numerically larger ones.

Control and Data Topologies: The Scribe application layer multicast protocol uses Pastry as the underlying routing substrate to provide multicast services. Therefore the control topology in Scribe is the same as the control topology in Pastry. The neighbors of any member on the control path include all its routing table, neighborhood set and leaf set entries.

Unicast paths to specific destination identifiers in Pastry are defined by the following rule:

> A message with destination identifier, y, is routed towards a member which has y as its node identifier. If no such member exists on the overlay, the message is routed to a member, z, whose node identifier is numerically closest to y. This routing is performed as follows — each member forwards the message to a member in its routing table that shares a longer common prefix with the destination identifier than its own identifier. When no such member can be found in the routing table, the message is forwarded to a member in the leaf set that is numerically closer to the destination identifier than its own identifier.

A multicast group in Scribe typically consists of a subset of the members that have already joined the Pastry overlay. Each multicast group in Scribe has its own identifier (and is called the topic identifier in [3]). The member whose node identifier is numerically closest to the multicast group identifier becomes the RP for that group. The data topology for a multicast group in Scribe is the union of the Pastry unicast paths from the different group members to the RP. The state for this data path is set up during the join procedure as described next.

Join Procedure: In Scribe, when a member on the overlay joins a multicast group, it routes a join message using the multicast group identifier as the destination identifier. This message gets routed by the Pastry substrate towards the RP. All members on this unicast path that are not already a part of the multicast data delivery tree for the group add themselves to the tree. A member needs to be joined to the Pastry group to be able to join a Scribe multicast group. The join procedure for Pastry is discussed in detail in [19].

VI. Other Application Layer Multicast Protocols

Among the different application layer multicast protocols proposed in literature we have described only a few

Scheme	Type	Tree-type	Max. Path length	Max. Tree degree	Avg. Control Overheads
Narada	Mesh-first	Source-specific	Unbounded	Approx. bounded	$O(N)$
HMTP/Yoid	Tree-first	Shared	Unbounded	$O(\text{max. degree})$	$O(\text{max degree})$
Bayeux/Scribe	Implicit	Source-specific	$O(\log N)$	$O(\log N)$	$O(\log N)$
CAN-multicast	Implicit	Source-specific	$O(dN^{1/d})$	constant	constant
NICE	Implicit	Source-specific	$O(\log N)$	$O(\log N)$	constant

TABLE I

A COMPARISON OF DIFFERENT APPLICATION LAYER MULTICAST SCHEMES.

examples in the prior sections. In this section we briefly mention other such protocols, all of which can be classified into one of the three approaches.

Gossamer [4] is a protocol that uses the mesh-first approach that constructs a mesh overlay for a set of application-level proxies that are used to the protocol to the needs of different applications. The Overcast protocol [10] 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. ALMI [13] is centralized overlay construction protocol that also uses the tree-first approach.

Bayeux [23] is an application layer multicast scheme which uses the implicit approach. It is built on top of a peer-to-peer object location system called Tapestry [22]. The Tapestry overlay structure is similar to Pastry. Although their underlying routing substrates are similar, Bayeux and Scribe differ in the way the multicast data delivery paths are created. An application layer multicast scheme based on delaunay triangulations [11] has also been defined which constructs the data delivery overlay using an implicit approach.

VII. COMPARATIVE STUDY

Different application layer multicast protocols have various properties that make them suitable for different applications. In this section, we present a comparison of these different protocols and comment on their suitability to specific applications. In Table I we present this comparison of the different aspects of the protocols. The tree-first protocols (Yoid and HMTP) create shared trees. All the other remaining protocols create source-specific trees. However, these source-specific trees is not nescessarily the "best" possible tree for a given source. This is because the flexibility of choosing a specific tree for each source is limited by the structure of the control topology.

In general, it is difficult to analytically compute either the stretch or stress metrics for most of the protocols. In particular, an analysis of the stress metric significantly depends on the characteristics of the underlying topology. Analysis of Bayeux [14] and NICE [1] have shown that both these protocols can guarantee a constant stretch for a "dense distribution" of members. Simulations have shown reasonable to good stretch performance of all the other different protocols.

In the table we show the path length measured by the number of application-level hops and the maximum degree of a member on the data delivery tree. These metrics are indirectly related to the stress and stretch metrics and are easily analyzed for the different protocols. The number of application-level hops is unbounded for both mesh-first and tree-first approaches. CAN-multicast also has a large bound for the number of application-level hops on the data path.

In Yoid and HMTP, members themselves choose their degree, and hence provide an upper bound for the tree degree. Although Narada defines a notion of maximum degree on the mesh, sometimes it is required to relax this constraint to allow new members to join the mesh. Otherwise in some cases new members will suffer a long latency before they find an existing mesh member with available degree. Therefore, we say that the tree degree for Narada is approximately bounded.

The average control overheads is highest for the mesh-first protocols, since each member exchanges state information with each other member. The average overhead at members is constant for both NICE and CAN-multicast.

Based on these observations, we make the following inferences on the applicability of the protocols to different applications:

- Mesh-first protocols are efficient for small multicast groups, while implicit protocols scale well with increasing group sizes.
- Tree-first protocols are less suited for latency-sensitive (e.g. real-time) applications but are useful to implement for high-bandwidth data transfers.
- Implicit protocols are particularly beneficial when the size of the multicast group is very large, and can be

adapted for both latency-sensitive applications (due to their short path lengths) and high-bandwidth applications (due to low tree degree).

VIII. Conclusions

Application layer multicast is a new approach to provide multicast services to group applications. In this peer-to-peer architecture, members organize themselves into an overlay topology for data delivery that adapts to the changing network conditions and group dynamics. All application layer multicast schemes can take advantage of network-layer multicast support where available. However, such additional network-layer capabilities are not essential for these protocols and therefore, can be easily deployed in the Internet today.

References

[1] S. Banerjee and B. Bhattacharjee. Analysis of the NICE Application Layer Multicast Protocol. Technical report, UMIACS-TR 2002-60 and CS-TR 4380, Department of Computer Science, University of Maryland, College Park, June 2002.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM Sigcomm*, August 2002.

[3] 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.

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

[5] 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.

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

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

[8] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6), December 1997.

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

[10] 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*, October 2000.

[11] J. Leibeherr and M. Nahas. Application-layer Multicast with Delaunay Triangulations. In *Global Internet Symposium, Globecom*, November 2001.

[12] J.C. Lin and S. Paul. RMTP: a reliable multicast transport protocol. In *Proceedings of IEEE Infocom*, March 1996.

[13] 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.

[14] 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.

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

[16] 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*, November 2001.

[17] I. Rhee, N. Ballaguru, and G.N. Rouskas. MTCP: Scalable TCP-like congestion control for reliable multicast. In *Proceedings of ACM Sigcomm*, August 1998.

[18] L. Rizzo. pgmcc: a TCP-friendly single-rate multicast congestion control scheme. In *Proceedings of ACM Sigcomm*, August 2000.

[19] 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)*, November 2001.

[20] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. *RFC 1075*, November 1998.

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