ALMI: An Application Level Multicast Infrastructure

Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, Marcel Waldvogel
dpendarakis@tellium.com, sherlia@arl.wustl.edu,
dverma@watson.ibm.com, mwa@arl.wustl.edu

WUCS-00-17

September 25, 2000

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

ALMI: An Application Level Multicast Infrastructure

Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, Marcel Waldvogel
dpendarakis@tellium.com, sherlia@arl.wustl.edu, dverma@watson.ibm.com, mwa@arl.wustl.edu

Abstract

The IP multicast model allows scalable and efficient multi-party communication, particularly for groups of large size. However, deployment of IP multicast requires substantial infrastructure modifications and is hampered by a host of unresolved open problems such as reliability, flow and congestion control, security and access control. Motivated by these problems, we have designed and implemented ALMI, an application level group communication middleware, which does not rely on network infrastructure support and thus, allows accelerated deployment and simplified configuration at the cost of a relatively small increase in traffic load.

ALMI is tailored toward support of multicast groups of relatively small size (several 10s of members) with many to many semantics. Participants of a multicast session are connected via a virtual multicast tree, i.e., a tree that consists of unicast connections between end hosts. The tree is formed as a minimum spanning tree (MST) where the cost of each link is an application-specific performance metric. The shift of multicast to end systems introduces certain performance penalties, such as duplicate packets on physical links and larger end-to-end delay than IP multicast. Using simulation, we show that the performance tradeoff is quite small and that ALMI multicast trees approach the efficiency of IP multicast trees. We have also implemented ALMI as a Java based middleware package and performed experiments over the Internet. The experimental results show that ALMI is able to cope with network dynamics and keep the multicast tree efficient.

1. Introduction

This work is motivated by the need to support group communication among a small group of hosts without relying on the IP multicast model. Traditional IP multicast, as defined by IGMP and related standards, provides an excellent solution to the communication needs of multicast groups with a large number of members. However, it requires fairly elaborate control support from network devices, such as IP routers, in particular membership management (IGMP) and multicast routing protocols. Since IP routers maintains separate routing state for each multicast group, the model is relatively less scalable with respect to the number of concurrently active multicast groups. Widespread deployment of IGMP and routing protocols requires substantial infrastructure modifications, and complex modifications to IP routers' software. Some of the issues associated with IP multicast, e.g. end-to-end reliability, flow and congestion control schemes, offer significant challenges for which no clear solutions have emerged thus far.

Internet service providers and corporate network administrators have usually been reluctant to deploy IP multicast support on a wide-spread basis. This is due to a number of unresolved problems; there is a lack of effective access control policies to limit the amount of multicast traffic in the network, there is potential for traffic explosion and the networks could be vulnerable to denial of service attacks by unauthorized senders. As an example, the MBONE provides an excellent medium for video broadcast of special events such as IETF meetings, but many large corporations block its transmission on their internal networks, restricting such transmissions to a very small number of machines.

There are a large number of applications whose requirements are substantially different from the design point of IP multicast. Such applications include video-conferencing, multi-party games, private chat rooms, web cache replication and database/directory replication. These applications usually contain a small number of group members, and the groups (e.g. multi-party games) are often created and destroyed relatively dynamically. The number of such groups that are concurrently active can be fairly large. For a large number of such small and sparse groups, the benefits of IP multicast in terms of bandwidth efficiency and scalability are quite often outweighed by the control complexity associated with group set-up and maintenance.

Due to the increasing number of such applications, and a lack of ubiquitous deployment of IP multicast in all IP-based networks, there has been renewed interest in multicast protocols that can be supported without relying on the IP multicast infrastructure. Some of the work has been motivated by applications like Internet TV, which are single source applications with a very large group size. These schemes, which include *Simple Multicast* [16], *Express* [10] and very recently, *Source-Specific Multicast* [11], offer multicast routing schemes which solve some of the problems of their traditional IP multicast counterparts, such as address allocation and access control. Nevertheless, all these solutions require substantial changes to the network infrastructure and their adoption by the network community and deployment in the Internet is yet to be seen.

In order to meet the requirements of the emerging applications, we need a solution for multi-sender multicast communication which scales for a large number of communication groups with small number of members, and does not depend on multicast support in the routers. In this paper, we propose an *application level multicast infrastructure* that addresses these concerns. This solution provides a multicast middleware which is implemented above the sockets layer. Application level multicast offers accelerated deployment, simplified configuration and better access control at the cost of additional (albeit small) traffic load in the network. Since application level multicast is implemented in the user space, it allows more flexibility in customizing some aspects, e.g. data transcoding, error recovery, flow control, scheduling, differentiated message handling or security, on an application-specific basis.

In our scheme, participants of a multicast session are connected via a *virtual multicast tree*, i.e. a tree that consists of unicast connections between *end hosts*. The tree is formed as a *Minimum Spanning Tree (MST)*, where the cost of each link is an application specific metric. The implementation we describe in subsequent sections uses the round-trip application level delay between group members as this cost metric. However, a plug-in architecture enables this metric to be changed easily by applications. In this paper, we present the architecture of the multicast middleware we have developed, a Java based implementation, and the results of some performance experiments conducted over a local area network as well as over the Internet. We have called this Java based package, ALMI for *Application Level Multicast Infrastructure*.

The rest of the paper is organized as follows. We first present an overview of our architecture, including the operation of control and data planes in section 2, followed by a design of application specific components in section 3. Sections 4 and 5 present simulation analysis and experimental evaluation of ALMI, respectively. Section 6 describes related work; we conclude in section 7.

2. ALMI Architecture and Operations

In this section, we describe the communication channels provided by ALMI and its related protocol operations for both controller and group members. We also describe operations related to multicast tree generation and criteria of tree updates and its stability issues. One of the advantage gained in ALMI is its value-added application specific components. To simplify explanation, we defer our design of these functionalities to next section.

2.1. Overview of ALMI Communication Architecture

Figure 1 depicts a high-level view of ALMI. An ALMI session consists of a session controller and multiple session members. Session controller is a program instance, located at a place that is easily accessible by all members. It may be co-located with one of the session members, typically the session initializer, or it could reside on a special purpose server or a multicast proxy installed within a corporate or an ISP network. Session members are organized into a multicast tree. A link in the multicast tree (solid line) represents a unicast connection between two members. Session data is disseminated along this multicast tree, while control messages are unicast between each member and the controller. The multicast tree is a shared-tree amongst members with bi-directional links. In order to avoid loops, two members incident on a link receive a designation of parent and child. This parent-child relation only distinguishes the two member for reasons we will explain later in this section; it does not indicate direction of data flow.

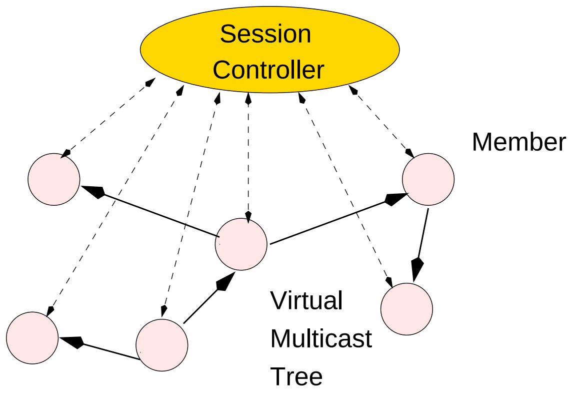

Figure 1: ALMI Architecture Overview

The session controller handles member registration and maintains the multicast tree. In order to achieve the latter, the controller performs two important functions:

- It ensures connectivity of the multicast tree when members join and/or leave the session and when network or host failures occur.
- It ensures the efficiency of the multicast tree by periodically calculating a minimum spanning tree based on the measurement updates received from all members. To collect measurements the controller essentially instructs each member to monitor a set of other members.

A session member receives and sends data as it would in an IP multicast session; in addition, it also forwards data to designated adjacent neighbors. Data eventually reaches all session members through this relaying process, assuming all members are cooperative. Coupled with ALMI's access control module, the assumption that a group does not contain any malicious member who intentionally blocks the data flow, is reasonable for our targeted applications such as video conferencing, web replication, etc. Furthermore, since final responsibility for distribution tree maintainance lies with the controller, a member that violates the cooperation requirement can be identified by the controller, based on feedback from other members, and purged from the tree. Besides forwarding data on the data plane, a session member also monitors the performance of unicast paths to and from a subset of other session members. This is achieved by periodically sending probes to these members and measuring an application level performance metric; in the current implementation the roundtrip response delay. Delay measurements are then reported to the controller and serve as the costs used to calculate a Minimum Spanning Tree.

ALMI takes the *centralized control* approach to maintain tree consistency and efficiency. This design choice is made for better reliability and reduced overhead during a change of membership or a recovery from node(i.e. end system) failure. On the other hand, the session controller manifests itself only in the control path, and does not obstruct high data rate transmissions among session members. We believe this centralized approach is adequate and efficient for a large range of multicast applications. However, a centralized controller architecture has obvious implications in control plane reliability and fault tolerance. Clearly, a single controller would constitute a single point of failure for all control operations related to the group. Two points should be made in this respect. First, the centralized session controller could be augmented with multiple *back-up controllers*, operating in "stand-by" mode, with addresses which are well known to all session members. In this case the "stand-by" controllers periodically receive state from the primary controller, which would include recent measurements, tree topology and current membership information. Second, even in the event that no control operation is possible, the existing ALMI tree, and hence data path, will remain unaffected and will continue operation until a membership change or a critical failure occurs. Therefore a transient controller (or its network) failure can be tolerated. In summary, we believe the benefit of simplicity offered by the centralized controller approach far outweigh any negative implications from the fault tolerance perspective.

2.2. Control Plane Operation

ALMI relies on a *control protocol* for communication between session controller and session members. This protocol handles tasks related to membership management, performance monitoring and routing. ALMI uses a common packet format to carry both data and control packets, shown in Figure 2.

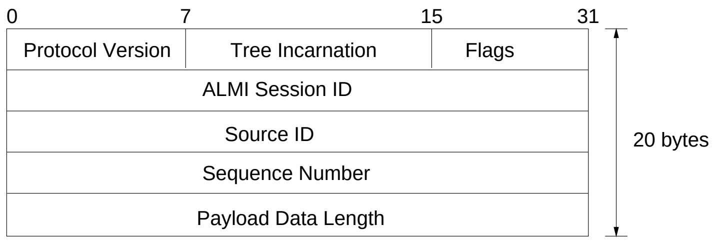

Figure 2: ALMI Packet Header Format

The content of this packet header is rather straight forward. *Session ID* and *Source ID* are generated by controller and guaranteed to be collision free. The *flag* field in the header defines various types of operation messages, including:

- Registration messages addressed from hosts to the controller. When a host joins a session, the controller returns a list of peering points from which the member should accept connection requests and the parent to which the new member is to initiate a connection.
- Connection request and acknowledgment between parent and child. This message exchanges parent and child data port numbers, which are locally bound with either TCP accept sockets or with UDP sockets. Members use these ports to initialize future data connection.
- Performance monitoring messages reported from members to session controller, such as pair-wise delay measurements between group members. In each update message, members include a list of <*current neighbor ID*, *delay measurement*> pairs. If the controller detects inconsistency between a member's view of neighbors with its own recording, it sets the corresponding measurement entry for this pair of neighbors to infinity.
- Distribution tree messages, generated by the controller, are used to inform members of their peering points in the data distribution tree. This message informs members of their new parent and children's ID. If a member detects that it needs to switch to a new parent, it sends a connection request to the new parent and closes the connection to the old parent at the mean time. On the other hand, if a parent detects a child is no longer in its children list, it will close the corresponding data connection as well.
- Neighbor monitoring update messages, which are sent by the controller to members to inform them a new list of neighbors they need to monitor. This message is triggered if the controller detects the number of current monitoring pairs has dropped below a threshold due to accumulated network errors. Or it is triggered due to the unsatisfaction of the current state of the multicast tree.
- Departure messages, are sent from group members to the controller and their current parent and children. If a child member receives such a message from its parent, it needs to contact the controller again to rejoin the group.

The *Tree Incarnation* field is to prevent loops and partitions in the multicast tree. Since a session multicast tree is calculated centrally by the controller, assuming correct controller operation, a loop free topology will always be generated. However, since tree update messages are independently disseminated to all members, there is always a possibility that some messages might be lost or received out-of-order by different groups members. In addition members might act on update messages with varying delay. All of these events could result in loops and/or tree partition. In order to avoid these transient phenomena, the controller assigns a monotonically increasing version number to each newly generated multicast tree. To avoid loops, a source generating packets includes its latest tree incarnation in the packet header. In order to guarantee tree consistency and at the mean time ensure delivery of most packets, each ALMI node maintains a small cache of recent multicast tree incarnations. Thus, an ALMI node simultaneously keeps state about multiple trees, each with the corresponding list of adjacent nodes. The number of cache entries is configurable. When receiving a packet with tree version contained in the cache, the receiving node forwards it across the interfaces corresponding to this tree version. Packets with tree versions not contained in the cache are discarded. On the other hand, if a member receives a data packet with a newer tree version, it detects that its information is not up to date and therefore re-registers itself with the controller to receive the new tree information.

2.3. Member Operation

Figure 3 shows a typical sequence of operations performed by a session member participating in an ALMI tree.

One of the first tasks a session member has to perform is to locate the session controller. It is assumed that initially, the session ID, the controller's address and port number are communicated or announced to members through online or offline schemes, such as a URL, a directory service or an email message. A session member is identified by its network address and port number, the combination of which will subsequently be referred to as the member's *address*. Members register by sending a *JOIN* message to the session controller. A member accepted to the group receives from the controller its *member ID*, as well as the ID and address of its parent. The member then sends a *GRAFT* message to its parent and in response obtains the data ports on which it receives and sends data.

Data distribution along the multicast session tree occurs on a hop by hop fashion. Depending on the application, data transfer between two adjacent members can be reliable or unreliable by deploying TCP (e.g. data replication services) or UDP (e.g. stream-based applications), respectively. There are clear advantages in being able to use existing, widely deployed protocols: first, it reduces system administration and configuration cost; and second, use of TCP and its associated congestion mechanism offers hop-by-hop reliability and provides compatibility in sharing bandwidth with regular flows. We stress that the last property is rather convenient since multicast congestion control is an extremely hard problem especially for its deployment viability. Additionally, applying TCP on a hop-by-hop basis implicitly creates back pressure for the source to slow down, resulting in end-to-end, albeit simplistic, congestion management.

When TCP is used, a connection has to be established between two adjacent nodes with one end initiating and the other end accepting the connection. Therefore, ALMI controller assigns parent and child labels to two adjacent nodes: a TCP connection is always initialized in the direction from a child to the parent. The parent-child relationship is also used in monitoring connectivity; if a

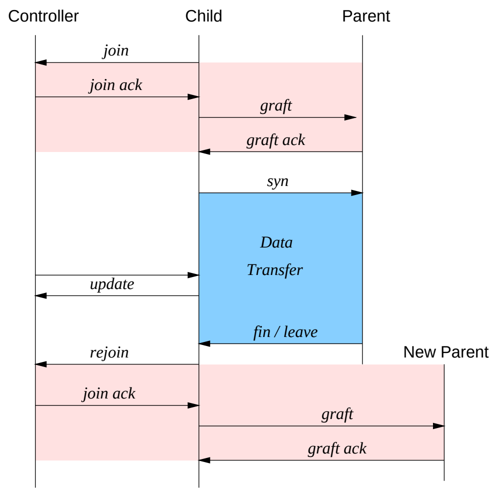

Figure 3: ALMI Member Operation

child detects failure of the connection to its parent, it considers itself disconnected from the graph and sends a *REJOIN* message to the controller. On the other hand, if the parent detects a child connection failure, it simply closes the connection. This relationship does not indicate directions of data flows, however, once the multicast tree is formed, each member forwards data to all adjacent members, including all children and the parent, except the one on which data is received.

As part of the evolving tree dynamics, a session member might be required to switch to a new parent. Such an event can be initiated by either the controller ("push") or the member ("pull"). In the former case, the controller instructs the member to switch to a new parent because a substantially better MST has been computed or a failure in the existing tree has been detected. In the latter case, the member detects through the monitoring process that it's parent is not responding or receives a message directly from the parent informing it that the parent is leaving the group. It then issues a *REJOIN* message to the controller, repeating the steps as when joining an ALMI group. In both cases, determination of a new parent is made by the controller.

2.4. Multicast Tree Generation and Update

We now turn to the computation of the ALMI distribution tree. A session multicast tree is formed as a virtual *Minimum Spanning Tree* that connects all members. The minimum spanning tree calculation is performed at the session controller and results are communicated to all members in the form of a *(parent, children)* list. Link costs are representative of an application specific performance metric which is computed by members in a distributed fashion and reported to the controller. Our current implementation uses the roundtrip delay, measured at the ALMI layer, as the performance metric. Each group member calculates roundtrip delay as an *exponential weighted moving average* to smooth jitters in the measurements. We choose latency as metric because it is important to most applications and is relatively easy to monitor. However, some applications may find other metrics such as available link bandwidth, more useful and better suited to match its performance measure. As an example, a bandwidth intensive application may prefer a high bandwidth, high delay link to

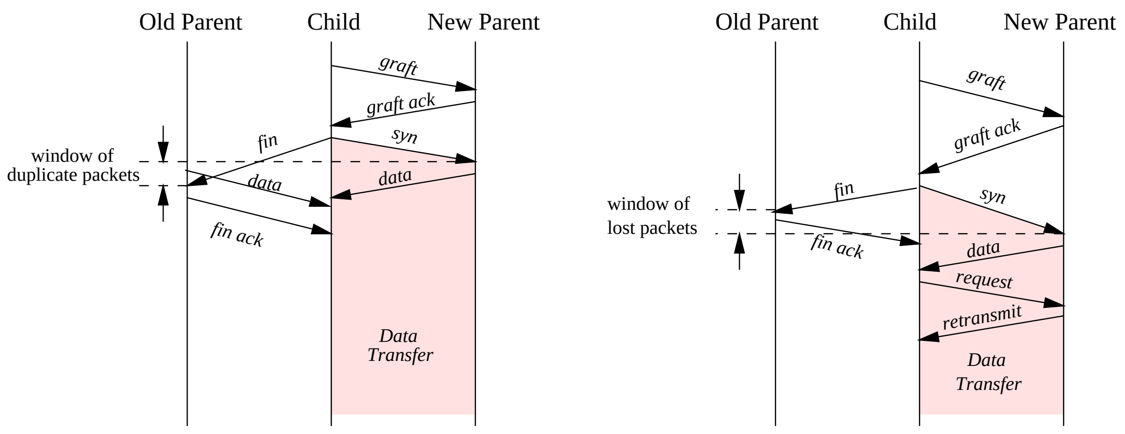

Figure 4: Member Operation during Tree Transitions

a low delay, low bandwidth link to carry its traffic. Design and development of these type of tools to obtain more sophisticated measurements helps ALMI to provide more flexible services and these tools can be easily plugged in as a module to ALMI. Nevertheless such instrumentation in a wide area network is non-trivial and it is beyond the scope of this paper to discuss these mechanisms. In the rest of this paper, we will simply use delay as the default performance metric.

Neighbor monitoring graph

In order to obtain monitoring results, ALMI connects all group members into a monitoring graph. Members send ping messages to measure round trip delay to its neighbors in the graph. For small groups, it is possible to create a mash and have $O(n^2)$ message exchanges to compute the best multicast tree. However, as group size grows, it becomes unscalable to have large number of message exchanges since the monitoring process is periodic and continuous through the whole multicast session. To reduce control overhead, we limit the degree of each node in the graph, i.e. the number of neighbors monitored by a member, to be constant so as to reduce the number of message exchanges to $O(n)$. The consequent spanner graph results in sub-optimal multicast tree since it does not have a complete view of all possible paths and its set of edges may not be a super set of all edges in MST. Such sub-optimality is reduced, however, by occasionally purging the currently known bad edges from the graph and updating it with edges currently not in the graph. Over time, the graph converges to include all edges in the optimal degree-bounded spanning tree. Likewise, in a dynamic environment, the graph updates to trace the better set of edges and to produce a more favorable multicast tree.

Multicast tree and its stability

Once members start to report monitoring results to their session controller, ALMI is able to improve the multicast tree from its initial random tree.[1] As described above, an ALMI multicast tree is a degree-bounded optimal spanning tree. Since most end hosts tend to be on access links

[1]By default, the set of neighbors in the multicast tree is a subset of neighbors in the monitoring graph, so a re-computation can only result in performance improvement.

rather than at network core, it is desirable to confine the number of packet copies traversing through access links to be small, i.e a small degree bound. On the other hand, if servers use ALMI to construct a multicast session and they have access to high speed network, the degree bound can be correspondingly configured higher.

A more crucial issue is how to achieve stability of the multicast tree since a change of tree is associated with operational cost such as *GRAFT*, *GRAFT ACK* and re-initiation of the data connection. More over, data packet may be lost or duplicated during a tree transition, as shown in figure 4, and recovery process can be expensive for it incurs additional delay and data buffering at the application. Therefore, our goal of improving the performance of multicast tree is only on a long term basis and any potential path oscillations are prevented. The controller calculates the overall performance gain of the new multicast tree and switches tree only if the overall gain exceeds a threshold. Both the frequency and threshold of switching tree are user configurable parameters.

3. Design of Application Specific Components in ALMI

Previous sections presented the architecture of control and data planes in ALMI. One of the advantages in ALMI is its ease of deploying value-added services for applications, such as end-to-end reliability, data integrity and authentication, and quality of service. A complete design of building blocks to fulfill these requirements is outside the scope of this paper. This section discusses briefly design points in supporting some of these components and in particular, we present our design and protocols for a reliable data distribution service which we have recently implemented.

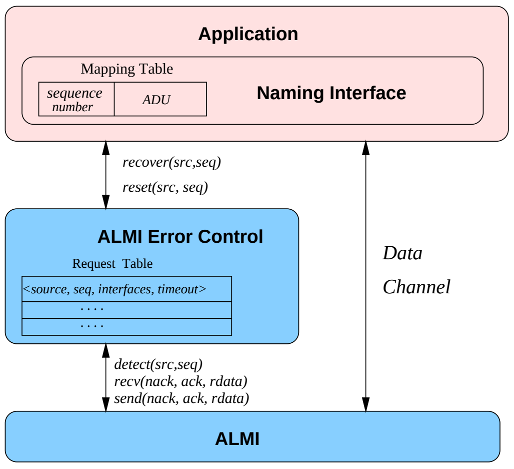

Figure 5: ALMI Error Control and Naming Architecture

3.1. End to End Data Reliability

Content distribution applications typically require data consistency and reliability. TCP has successfully satisfied these requirements for unicast connectivity; a TCP-equivalent reliable transport protocol for multicast communication has been the subject of active research in recent years [12].

In an ALMI multicast group, the end-to-end reliability problem still exists; however, the cause of the problems differs greatly from that over IP multicast. In ALMI, unicast TCP connections provide data reliability on a hop-by-hop basis, which implies that packet losses due to network congestion and transmission errors are eliminated. Instead, the main reason for packet losses in ALMI are due to multicast tree transitions, transient network link failures, or node failures.

In ALMI, implosion and exposure control happens naturally, it efficiently aggregate requests and retransmit data without the need for router support as in [19, 13] or knowledge of session topology as in [14]. Upon loss detection, a session member sends a request onto the interface where data is received from. Requests are then aggregated at each hop so that only one of them escapes the loss subtree. When applications can buffer data or regenerate data from disk, retransmission can happen locally. In this case, the node above the lossy link will retransmit data to the requesting subtree. Otherwise, when upstream node has reset its application naming states(explained below) and can no longer retransmit data locally, a *NODATA* packet is sent back to the requestor, i.e. the head of the loss subtree. The requestor then initiates an *out-of-band connection* directly to the source, and subsequent request and retransmit are conducted over this out-of-band connection. In both local and out-of-band retransmission, upon receiving retransmitted packets, requestor forwards them to downstream requestors. The out-of-band connection is torn down after fulfilling the request. The choice of out-of-band request versus relaying request and retransmissions hop-by-hop is due to ALMI's loss characteristics: they are infrequent but usually happen in bulk. Typically, once a node loses its connection, it takes about 3 round trip time to re-connect to the multicast tree and detect packet losses. Although relaying request all the way up to the source can sometime aggregate more independent loss requests at higher up the tree, it adds per-hop processing and transmission delay for each request and retransmission packet, and also disrupts the normal data distribution process. On the contrary, an out-of-band connection separates data distribution from retransmissions and have much less processing delay.

Additionally, ALMI also deploys *ACKs* to synchronize data reception states at members. This is necessary for applications that require total reliability but have limited buffer space. Before resetting their buffers, members need to ensure all packets in buffer are correctly received by all members. An *ACK* is a list of <source, sequence number> pairs, where sequence number is the highest contiguous sequence number received locally from a data source. Initiated from leaf nodes, *ACKs* are sent upstream towards the root. At each intermediate node, once a member received *ACKs* from all its children, it forwards upstream an *ACK* containing the minimum of sequence numbers for each source. When the *ACK* reaches root, it is multicasted back downstream and reset every nodes' state to their common minimum. A member is then free to clear up all packet buffers with sequence number less than the minimum. The frequency of the *ACK* process depends on both the data rate and the smallest buffer space at a member application.

3.2. Data Naming

An important question related to error recovery is that of *data naming.* Applications and ALMI require a commonly understood naming convention so that they can communicate which data is requested. Since losses in ALMI group are more likely to occur in batches over dispersed time intervals rather than isolated packets on regular time intervals, sequence numbers as used by TCP, are insufficient to specify a member's data reception state and could hinder a members' ability to request and retransmit data efficiently. Furthermore, an application may decide to ignore certain packets, for example, packets containing out-of-date information, and only recover others. A data naming component is thus more desirable since it allows flexibility in tailoring application reliability semantics.

In ALMI's data naming interface, an application can specify the mapping between its *application data units* and ALMI packet sequence numbers. An *ADU* is solely defined by application protocol, for example, for some database applications, it can be an object ID; or for a ftp application, a tuple containing <file name, offset, length>. Other more sophisticated mechanisms such as hierarchical data naming schemes [17, 5] can be incorporated as well, to achieve better flexibility and efficiency.

3.3. Other Components

There are many other functionalities that could be incorporated into ALMI, such as delay constraints for real-time sessions, access control for private multicast sessions and etc. In ALMI, an application delay bounds can be achieved by constraining the diameter of the computer MST tree. Similarly, the multicast tree can be computed with constraints on the degree of session members, in order to achieve better load balancing. Regarding access control, the session controller is naturally capable of controlling which members are allowed to join; furthermore, the controller can act as a key distribution center, distributing symmetric keys to encrypt the data, as well as certificates and signed public keys that should be used for data authentication. We are currently underway adding these components to ALMI.

4. Simulation Analysis of ALMI Multicast Tree Efficiency

While ALMI achieves group communication without relying on network layer multicast support and reduces the control load associated with group set-up and maintainance, it is bound to exhibit lower transmission efficiency since nodes on the distribution tree have to be ALMI capable and, thus currently confined to end hosts. Moreover, packet processing and forwarding at the application layer typically incurs higher delay when compared to router processing at the IP layer. In this section we investigate the extent of these ALMI performance constraints by conducting experiments which compare ALMI to IP multicast. Results obtained provide insight onto the trade-offs associated with ALMI and allow us to decide the applicability of ALMI for specific applications and deployment settings.

We examine the relative cost of an ALMI tree to those of source-rooted shortest path multicast trees as well the cost of a mesh of unicast connections which would have to be used in the absence of any multicast support. Trees are generated and costs computed over a set of random graphs with a variable number of multicast group members. The algorithms for generating random graphs are similar to those in [22], where a connected graph is generated with a specified edge connectivity probability.

In comparing the cost of an ALMI multicast tree to that of source-rooted shortest path multicast trees we note that since ALMI constructs a shared multicast tree, the cost of distributing data is the same independently of the location of the sender(s). However, this property does not hold for source-rooted trees, in which data originating at different nodes will traverse paths of differing cost to reach all group members. Therefore, to achieve a meaningful comparison, the cost of an ALMI multicast tree is compared with the average cost of all shortest path trees rooted at each group member.

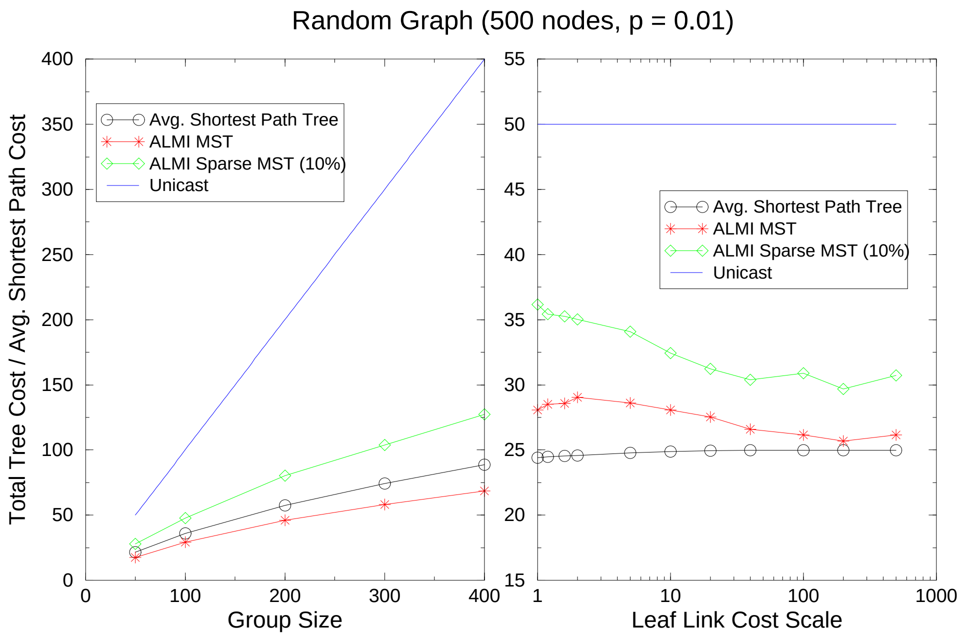

Figure 6: Cost Comparison of ALMI MST and Shortest Path Tree in Random Graph

As mentioned in Section 2, ALMI provides a mechanism to further reduce control traffic load by allowing members to collect delay measurements to only a subset of other group members. Obviously, performing the MST calculation on a (connected) subgraph results in a sub-optimal ALMI distribution tree. In this section, we analyze quantitatively the impact of this mechanism in terms of how much it increases the cost of the actual ALMI multicast tree. The cost of an ALMI tree is defined to be the sum of delays on each link of the shared multicast tree; all link delays are assumed to be symmetric.

Figures 6 and 7 depict multicast tree cost in a random graph and a transit-stub graph, respectively. Each data point is derived by averaging over the results of 10 graphs. Random graphs in Figure 6 consist of 500 nodes with an average node degree of 5, and transit-stub graphs in Figure 7 consist of about 6000 nodes, with an average node degree of 3. More details about the formation of transit-stub graphs can be found in [22]. Link costs are uniformly distributed in the interval $[0, 1]$.

In both figures, the x-axis of the graph on the left depicts multicast group size; groups of variable size are formed by selecting a random subset of network nodes as group members. It is assumed that every network node can be co-located with a host. The graphs on the left plot the average cost of all source-rooted trees, one for each multicast group node, the ALMI MST cost and the cost of a mesh of $O(n^2)$ unicast connections among all group members. We also compute the cost of an ALMI multicast tree calculated from incomplete information, denoted as “ALMI sparse MST”. This tree

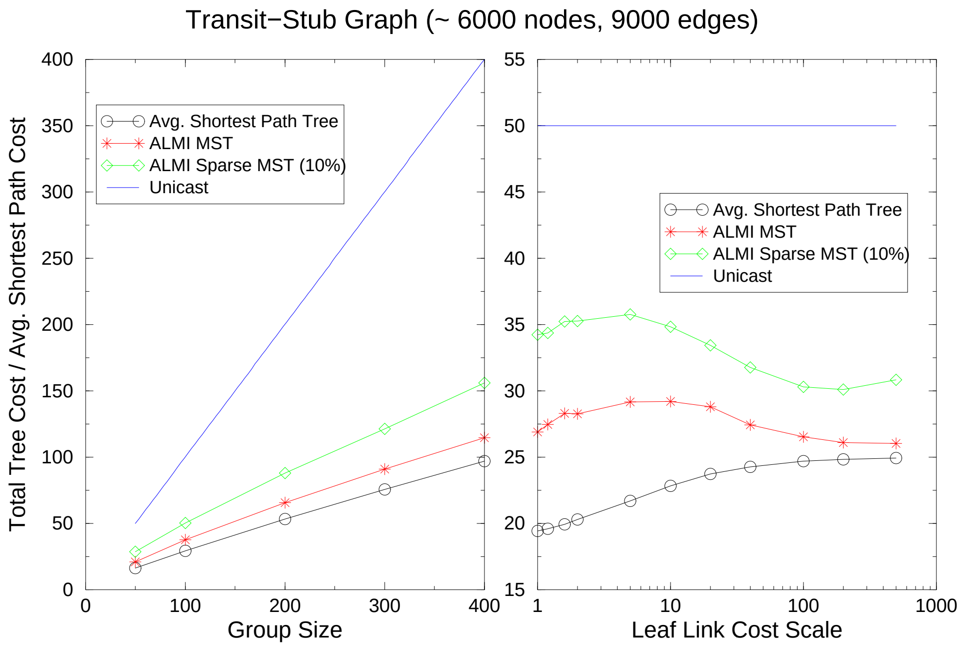

Figure 7: Cost Comparison of ALMI MST and Shortest Path Tree in Transit-Stub Graph

corresponds to the case where every ALMI node monitors the delay to just 10% of the total number of group nodes.

We first concentrate on the results depicted in the left graphs of figures 6 and 7. It is interesting to observe that for the random graph, at all group sizes the ALMI MST cost is smaller than the average source-based tree cost. This is essentially due to the fact that an ALMI multicast tree is an MST tree; optimal source based trees are computed based on information local to each node and, therefore, are not globally optimal. On the other hand, in a transit-stub graph, the ALMI multicast tree is about 20% more expensive. This difference is due to the distinct characteristics of the two types of graphs. Since an ALMI multicast tree consists of a collection of unicast paths between hosts, some network links will be typically traversed multiple times. In a transit-stub graph, since hosts reside in stub networks, the links between transit domains and stub domains will most certainly be traversed multiple times, whereas in the random graph topology, since hosts are co-located with network nodes and uniformly distributed throughout the graph, the number of such links are fewer, hence lowering the cost of the ALMI multicast tree. Finally, as expected, the ALMI sparse MST has a higher total cost since it is derived using a subset of link metrics. Still, the cost difference in all cases is within 50%, which could be considered a reasonable price to pay for a 90% reduction in performance monitoring traffic.

Thus far, we have assumed that all network links have equal cost and that hosts are co-located with network nodes; in other words host are attached to the network with zero cost. In practice, however, this assumption might not be accurate; typically *"last mile"* links have lower bandwidth and thus result in higher delays and MST costs. Higher "last mile" costs could adversely impact ALMI, since all data flows in and out of non-leaf nodes in the ALMI tree at least twice and hence, the cost of link connecting hosts to a network aggregation point will contribute more to the total tree cost. In the right side graphs of Figure 6 and 7, we plot tree costs against the cost of the "last-mile" links. We include the same comparisons; ALMI MST, ALMI "sparse MST", average of all shortest path trees and meshed unicast connections. In this simulation, multicast group size is fixed to 50 and the "last-mile" link cost is uniformly distributed between 0 and $scale$, shown on the x-axis.

The results demonstrate that, even for a moderate group size of 50 members, the benefit of ALMI over pure unicast is still significant, reducing tree cost to only half. Furthermore, it is observed that as the cost of "last-mile" links increases, ALMI multicast tree cost decreases and approaches the cost of the average shortest path tree. This is due to the fact that MST calculation results in a tree which tends to prefer inclusion of low-cost links. This is similar to the behavior that would be observed if servers were deployed in the network to help relay data to other parts of the network. Overall, the simulation clearly shows the advantage of an ALMI multicast tree over $O(n^2)$ unicast connections. The fact that ALMI is almost as efficient as the shortest path trees even in the presence of incomplete measurements, argues that it is a rather attractive solution for many multicast applications.

In this simulation, we have focused on comparison of ALMI multicast tree with source-rooted shortest path trees. Compliment to SPTs, shared multicast tree, as constructed from CBT [1] and PIM-SM [7] optimizes the total cost of the multicast tree. Although it is known that finding the optimal center for the multicast group is an NP-complete problem, there are heuristic placement strategies to select one of the group member or network node to be the core. In [21], it shows that a resulting shared multicast tree from a feasible heuristic method has an average cost of 95% of the cost of shortest path tree for a varied number of group sizes, average node degree and different node distributions. Therefore, we infer that the cost difference between ALMI multicast tree and CBT or PIM-SM will be comparably small as well.

5. Experimental Evaluation of ALMI

We have implemented ALMI as a Java-based middleware package using JDK1.2 [20]. In the next two experiment sets, we evaluate the performance of an actual operational group of ALMI nodes over either a WAN or a LAN. These two scenarios have fundamental differences; in a LAN environment most of the delay between two ALMI nodes is due to host processing while over a wide area network, delay is mostly due to transmission, propagation and queuing delay over the network.

5.1. Experiment Over WAN

Over a wide area network, ALMI has to cope with the dynamics of network paths, such as distortion of delay measurements and transient link failures. ALMI needs to prevent the multicast tree from diverging from an efficient construction. To demonstrate that ALMI is able to achieve a cost-efficient tree, we have conducted experiments over 9 sites scattered in both US and Europe. Figure 8 shows an example topology during the experiment. Link delays is measured from traceroute output.

The experiment was run as follows. We started ALMI at all 9 sites and configured the ALMI controller to re-calculate the multicast tree every 5 minutes. Simultaneously, we run traceroute from each site to every other site periodically, every 5 minutes. The output from traceroute provides us with a benchmark of the network delay experienced between nodes during our experiment. We

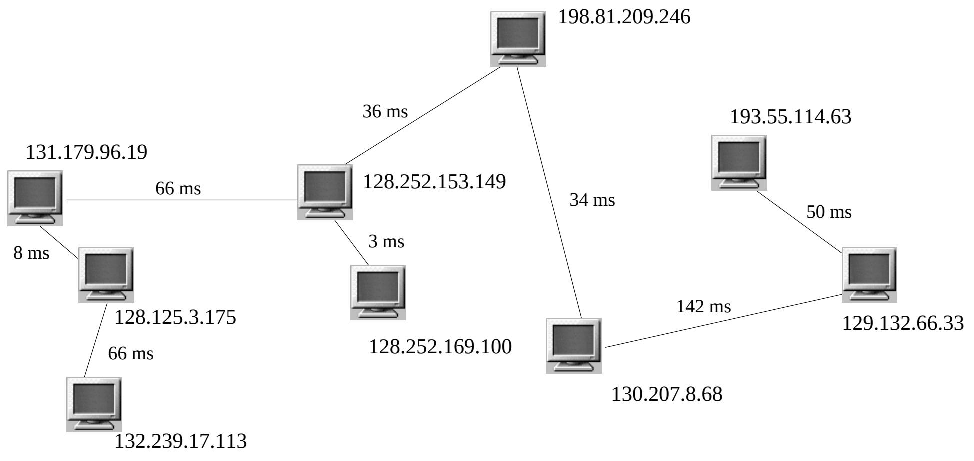

Figure 8: Example WAN Topology (Path delay measured from traceroute)

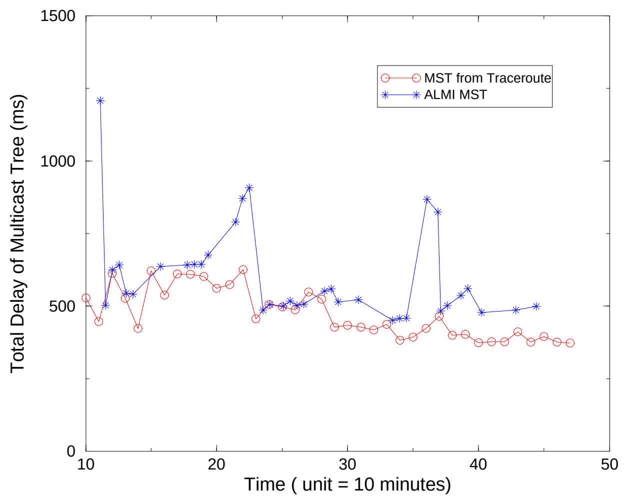

Figure 9: Evaluation of ALMI MST in WAN Test

then compare the total delay of an MST computed from the traceroute measurements to that of the ALMI multicast tree computed by the ALMI controller. For this experiment, we used the traceroute measured delay as the ALMI tree link cost in order to achieve a fair comparison. In other words, the comparison reflects only the difference of tree composition, excluding the distortion caused by delay measurements at the application level.

Figure 9 shows the result of a six hour test run of a single multicast session. Initially, the cost of ALMI multicast tree is very high, since the ALMI controller does not have *a priori* topological knowledge about group members and randomly connects members to each other at the beginning of the session. However, the ALMI tree cost was quickly brought down at the next re-calculation of the tree and stays close to the real MST cost, as the controller periodically gathers measurement reports from group members and updates the ALMI MST. There are two spikes in the ALMI MST, at time units 22 and 36 respectively. Analyzing the traces, we found that both points are caused by transient network failures. In the first case, one of a pair of two nodes, who are very close to each other, detects the other end as unreachable and connects to a much higher cost neighbor. In the second case, one node experiences temporary network failure and is timed out at the controller. The network recovers after approximately 15 minutes and the node re-joins the group but is randomly assigned a new parent. The presence of a new member, either at the session beginning or during the session, always introduces sub-optimality of the tree since they are randomly connected to the rest of the ALMI multicast tree. A more intelligent controller may be able to use one of the Internet services such as in [9, 18, 15] to estimate the topological information of a new member and initialize its connection more efficiently. We conclude from this experiment that ALMI is able to use application perceived delay to construct an efficient multicast distribution tree in a highly dynamic network environment.

5.2. Experiment Over LAN

In this experiment, we test a scenario where network bandwidth is higher than what end-systems can consume, and test the forwarding processing delay caused by ALMI processing. We used a Sun Ultra-1 attached to a 10Mb/s Ethernet network as a source sending data to several Pentium III - class PCs connected over a 100 Mb/s LAN. We vary the number of intermediate data relaying hops and measure the throughput at the last hop. In this experiment, we use TCP connection between nodes and confine the controller to connect members as a chain in order to capture the effect of ALMI member node forwarding.

packet size	**Zero Hop** (KB/S)	**One Hop** (KB/S)	**Two Hops** (KB/S)
64	156.83	154.83	153.994
128	278.57	209.98	190.56
256	489.26	439.19	422.69
512	657.81	642.83	609.13
1024	752.47	732.85	769.74
2048	800.55	797.33	788.63
4096	813.84	813.18	836.82

Table 1: Experiment of ALMI forwarding delay in end systems

From Table 1, we observe that the throughput achieved in all cases remains stable regardless of the number of intermediate hops. This shows that ALMI processing delay does not increase with the higher number of data relaying hops. From a scalability point of view, this means that the overall TCP throughput achieved in a session is decided by the slowest network path or intermediate hop, but is not affected by the aggregation of bottlenecks if there are multiple. On the other hand, if we look at Table 1 vertically, we see that the processing delay associated with each packet is relatively high, especially for small size packets. This is due to the fact that the Java virtual machine is still comparably slow even in the presence of JIT. However, we believe this gap will be reduced in the near future with the advances of better compilers and faster CPUs.

6. Related Work

Challenging the conventional wisdom of IP multicast, ALMI explores an alternative architecture to apply multicast paradigm in the current Internet. There are two closely related projects emerging independently at the same time which have very similar objectives as ALMI does. Yallcast [8], aims to extend the Internet multicast architecture and defines a set of protocol for host-based content distribution either through tunneled unicast connections or IP multicast wherever available. It uses a *rendezvous host* to bootstrap group members into the multicast tree. The functionality of the *rendezvous host* is similar to ALMI's group controller, it is only used to inform new members about several current members in the tree and is not connected to the multicast data paths. Yallcast creates a shared multicast tree using a distributed routing protocol. It also maintains a mesh topology among group members to ensure that the multicast group is not partitioned. Overall, Yallcast envisions the deployment of IP multicast into small and disjunct network islands and provides a rudimentary architecture for global multicast. In contrast to Yallcast, Endsystem Multicast [4] is more similar to ALMI in aiming towards small and sparse group communication applications. In Endsystem Multicast, group members are self-organized into multicast trees using a DVMRP [6] like routing protocol and creates source-based multicast tress. It require members to periodically broadcast refresh messages to keep the multicast tree partition free. A companion protocol of Endsystem Multicast is called Narada, which focuses on optimizing the efficiency of the overlay, in terms of delay bounds, based on end-to-end measurements. Both Yallcast and Endsystem Multicast are still in their initial evaluation stage and at this point, we are not aware of any performance reports. Although, Yallcast and Endsystem Multicast have their end goals align with those of ALMI, the tree construction algorithms are very different in all three protocols. Both Yallcast and Endsystem Multicast try to leverage the existing multicast routing protocols and re-apply them at the application level. However, we argue that one of the fundamental complexities comes with IP multicast is its complication in routing protocols. Although, at the application level, such complexity can be greatly reduced, due to the number of nodes involved is much fewer than the number of routers all over the Internet, a fully distributed algorithm may still cause excessive control overheads and incur reliability problems, which are the same problems as existed in current multicast routing protocols. A centralized control protocol as the one in ALMI, with careful design of redundancy, can simplify the matter greatly and provides a more reliable mechanism to prevent tree partitions and routing loops.

There are other relevant projects that also deploy multicast at the application level, with more emphasis on each specific applications. RMX [3] is a project that installs multicast proxies to connect islands of IP multicast with co-located homogeneous receivers. Besides relaying data, an RMX proxy also adapts to the heterogeneous environment using detailed application knowledge. For example, an RMX proxy can act as a transcoder to accommodate the low bandwidth receivers. The tree configuration among RMX proxies are static right now and there is no self-configuration and adaptation aspects of the multicast overlay as of this writing. AMRoute [2] is a protocol for host-based multicast over mobile wireless networks. It assumes the existence of an underlying broadcast mechanism for configuration purposes. AMRoute continuously creates a mesh of bidirectional tunnels between a pair of group members. Additionally, each multicast group has a *core node* which is responsible for the initial signaling and tree creation. The AMRoute core uses a source routing approach, where source is the core node itself, and selects a subset of the available virtual mesh links to form a multicast distribution tree. The core can also migrate dynamically according to group membership and network connectivity. Both of these projects bear similarities to ALMI, yet ALMI is defined as a more general infrastructure for a wide range of applications rather than for a specific application or environment.

7. Conclusions and Future Work

This paper presented ALMI, an application level multicast infrastructure, that has been designed and built to provide a solution for multi-sender multicast communication which scales to a large number of communication groups with small number of members, and does not depend on multicast support at the IP layer. This solution provides a multicast middleware which is implemented above the sockets layer. Application level multicast offers accelerated deployment, simplified configuration and better access control at the cost of small additional traffic load in the network. Simulation results, along with initial experimentation results indicate that the performance tradeoff is quite small and that ALMI multicast trees are close to the efficiency of IP multicast trees. Since application level multicast is implemented in the user space, it allows more flexibility in customizing some application related modules, e.g. data transcoding, error recovery, flow control, scheduling, differentiated message handling and security.

We plan to extend this work in multiple ways. We are enhancing the performance evaluation work to include experiments with a larger number of nodes, as well as integrating with real life applications so that, besides control, data performance characteristics can be studied in detail. In addition, we plan to implement and study in more detail application specific modules such as end-to-end reliability, naming and security. In terms of speeding the performance of our middleware, we will explore options of moving parts of the forwarding functionality to an OS kernel and defining an interface between ALMI and the OS specific parts.

8. Acknowledgment

Our sincere thanks to the following people for their generosity of letting us use their computing resources for conducting our experiments: Ernst Biersack of Eurecom, France; Andreas Festag of TU Berlin, Germany; Aiguo Fei of University of California, Los Angeles; Ralph Keller of ETH Zurich; Manamohan Mysore of University of California, San Diego; Christos Papadopoulos of University of Southern California; Injong Rhee of North Carolina State University; Ellen Zegura and Peter Wan of Georgia Institute of Technology. Without their help, this work would never have been completed.

References

[1] A. Ballardie. Core Based Trees (CBT version 2) Multicast Routing - Protocol Specification. RFC 2189, 1997.

[2] E. Bommaiah, L Mingyan, A. Mcauley, and R. Talpade. Amroute: Adhoc multicast routing protocol. Internet Draft, August 1998.

[3] Y. Chawathe, S. McCanne, and E. Brewer. RMX: Reliable Multicast in Heterogeneous Networks. In *Proc. IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[4] Y. Chu, S. Rao, and H. Zhang. A Case For EndSystem Multicast. Internet End-to-End Research Meeting, June 1999.

[5] J. Crowcroft, Z. Wang, A. Gosh, and C. Diot. RMFP: A Reliable Multicast Framing Protocol. Internet Draft, March 1997.

[6] Distance Vector Multicast Routing Protocol - DVMRP. RFC 1812.

[7] D. Estrin, V. Jacobson, D. Farinacci, L. Wei, S. Deering, M. Handley, D. Thaler, C. Liu, Sharma P., and A. Helmy. Protocol Independent Multicast-Sparse Mode (PIM-SM): Motivation and Architecture. Internet Engineering Task Force, August 1998.

[8] P. Francis. Yallcast: Extending the Internet Multicast Architecture. http://www.yallcast.com, September 1999.

[9] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. An Architecture for a Global Internet Host Distance Estimation Service. In *Proc. IEEE INFOCOM*, 1999.

[10] H. Holbrook and D. Cheriton. IP Multicast Channels: EXPRESS Support for Large-scale Single Source Applications. In *Proc. ACM SIGCOMM*, Boston, MA, September 1999.

[11] H. Hummel. Source Specific Multicast. IETF draft, draft-hummel-ssm-00.txt, June 2000.

[12] Katia Obraczka. Multicast Transport Mechanisms: A Survey and Taxonomy. In *IEEE Communications Magazine*, January 1998.

[13] C. Papadopoulos, G. Parulkar, and G. Varghese. An Error Control Scheme for Large-Scale Multicast Applications. In *Proc. IEEE INFOCOM*, 1998.

[14] S. Paul, K. K. Sabnani, J. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). In *Proc. IEEE INFOCOM*, 1996.

[15] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An Architecture for Large-Scale Internet Measurement. *IEEE Communications*, pages 48–54, August 1998.

[16] R. Perlman, C. Lee, A. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green. Simple Multicast: A Design for Simple, Low-Overhead Multicast. IETF draft, draft-perlman-simple-multicast-03.txt, October 1999.

[17] S. Raman and S. McCanne. Scalable Data Naming for Application Level Framing in Reliable Multicast. In *Proc. ACM Multimedia '98*, Bristol, UK, September 1998.

[18] S. Seshan, M. Stemm, and R. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proc 1st Usenix Symposium on Internet Technologies and Systems (USITS '97)*, Monterey, CA, December 1997.

[19] T. Speakman, D. Farinacci, S. Lin, and A. Tweekly. PGM Reliable Transport Protocol. *Internet Draft*, August 1998.

[20] Java^{TM} 2 Platform. http://www.javasoft.com.

[21] L. Wei and D. Estrin. The Trade-offs of Multicast Trees and Algorithms. In *Proc. of IC3N*, 1994.

[22] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. IEEE INFOCOM*, San Francisco, CA, 1996.