A Proposal for A Scalable Internet Multicast Architecture

Authors: Sherlia Shi

We propose a new network and system architecture for multicast in the Internet. Our main objectives are to find a cost-effective way to scale to a large number of multicast groups whose members are geographically dispersed, and to enable small and less capable devices to participate in group communications. In order to preserve the efficiency of data distribution gained by multicast, while avoiding the control complexity previously exhibited by IP multicast, we propose the use of an overlay network for multicast services. We construct "virtual" multicast trees, which consist of unicast connections joining multicast servers in the network. These servers represent their end users who are interested in communicating together, and replicate and forward data on their behalf. A network design problem is presented to study the implication of such overlay network. On top of this network communication channel, we devise a transport layer and system architecture to provide crucial functionalities to applications. In addition, we propose a multicast mechanism that involves only end-systems. This allows accelerated application deployment without the need of network...

Read complete abstract on page 2.
A Proposal for A Scalable Internet Multicast Architecture

Complete Abstract:

We propose a new network and system architecture for multicast in the Internet. Our main objectives are to find a cost-effective way to scale to a large number of multicast groups whose members are geographically dispersed, and to enable small and less capable devices to participate in group communications. In order to preserve the efficiency of data distribution gained by multicast, while avoiding the control complexity previously exhibited by IP multicast, we propose the use of an overlay network for multicast services. We construct "virtual" multicast trees, which consist of unicast connections joining multicast servers in the network. These servers represent their end users who are interested in communicating together, and replicate and forward data on their behalf. A network design problem is presented to study the implication of such overlay network. On top of this network communication channel, we devise a transport layer and system architecture to provide crucial functionalities to applications. In addition, we propose a multicast mechanism that involves only end-systems. This allows accelerated application deployment without the need of network infrastructure support. To gain experience of the interfacing between applications and the multicast communication channel, we implement a multicast middleware package and prototype applications for future experiments.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/246
A Proposal for A Scalable Internet Multicast Architecture

Sherlia Shi

WUCS-01-03

June 2001

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130
A Proposal for A Scalable Internet Multicast Architecture

Sherlia Shi
Department of Computer Science
Washington University in St. Louis
sherlia@arl.wustl.edu

Abstract
We propose a new network and system architecture for multicast in the Internet. Our main objectives are to find a cost-effective way to scale to a large number of multicast groups whose members are geographically dispersed; and to enable small and less capable devices to participate in group communications. In order to preserve the efficiency of data distribution gained by multicasts, while avoiding the control complexity previously exhibited by IP multicast, we propose the use of an overlay network for multicast services. We construct "virtual" multicast trees, which consist of unicast connections joining multicast servers in the network. These servers represent their end users who are interested in communicating together, and replicate and forward data on their behalf. A network design problem is presented to study the implication of such overlay network. On top of this network communication channel, we devise a transport layer and system architecture to provide crucial functionalities to applications.

In addition, we propose a multicast mechanism that involves only end-systems. This allows accelerated application deployment without the need of network infrastructure support. To gain experience of the interfacing between applications and the multicast communication channel, we implement a multicast middleware package and prototype applications for future experiments.

1 Introduction

With the enormous advances in network computing and communication technologies, the Internet has become essential for information exchanges in many parts of the world today. Yet, today's web and email based networking is just the beginning of an upcoming information age, with the ultimate technology wave still preparing its entrance. The next generation of the Internet will ride on the vast progress on network infrastructure, which enables two important advances. First, it enables high speed real-time multimedia applications to be carried over the commodity Internet; Second, broadband access reaches to millions of households enabling person-to-person network communications in a cheaper but better way.

However, today's computer-supported communication is largely limited to data exchange between two computers, or point-to-point communications. Group communication, on the other hand, is minimally supported, even though it is an equally important and natural model of communication in people's day-to-day experiences. Students going to classes, professionals going to staff meetings, friends getting together watching a game, are all different forms of group communication. Unfortunately, most of these applications are still little developed or are only supported in very limited scales. This lack of support coincides with the limited and expensive network infrastructure we have today, but as stated earlier, this will change shortly and the availability of rich-media network communication and high-speed network access for households and business corporations, with the appropriate development of applications, will drive the demands for system and network support for group communications.

1.1 Group Communication Applications and Their Characteristics

There is a diverse range of applications that inherently require group communication and collaboration: video conferencing, distance learning, distributed databases, data replication, multi-party games and distributed
simulation, network broadcast services and many others. The diversity of these applications demands versatile support from the underlying system in many dimensions. Examples of these dimensions include the amount of data that need to be delivered (bandwidth requirement), the timeliness of their delivery (latency requirement), the reliability of their delivery (reliability requirement), and the idiosyncrasy of a group such as the number of data sources (multi-source requirement), the number of recipients to be reached (scalability requirement), and the frequency of members joining or leaving the group (dynamics requirement). Table 1 summarizes the individual characteristics of these next-generation applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Multi-source</th>
<th>Scalability</th>
<th>Dynamics</th>
<th>Bandwidth</th>
<th>Latency</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Conference</td>
<td>all</td>
<td>small</td>
<td>low</td>
<td>medium</td>
<td>critical</td>
<td>no</td>
</tr>
<tr>
<td>Distance Learning</td>
<td>one or few</td>
<td>medium</td>
<td>low</td>
<td>medium</td>
<td>critical</td>
<td>no</td>
</tr>
<tr>
<td>Distributed Cache Update</td>
<td>few or all</td>
<td>medium</td>
<td>low</td>
<td>high</td>
<td>non-critical</td>
<td>yes</td>
</tr>
<tr>
<td>Internet TV/Broadcast</td>
<td>one</td>
<td>huge</td>
<td>high</td>
<td>high</td>
<td>critical</td>
<td>no</td>
</tr>
<tr>
<td>Multi-party Games</td>
<td>all</td>
<td>large</td>
<td>high</td>
<td>low</td>
<td>critical</td>
<td>yes</td>
</tr>
<tr>
<td>Distributed Simulation</td>
<td>all</td>
<td>large</td>
<td>low</td>
<td>high</td>
<td>depends</td>
<td>yes</td>
</tr>
<tr>
<td>Distributed Instrumentation</td>
<td>all</td>
<td>huge</td>
<td>low</td>
<td>low</td>
<td>non-critical</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 1: Application Characteristics for Group Communication

Supporting these applications has imposed a serious challenge to our current communication systems. Due to the prevalence of underlying point-to-point connectivities, communication systems are quickly reaching their limit. A typical example is that requests to a popular web server usually experience long response time due to server overload since it has to establish individual connections for each incoming data request, even for requests for the same objects. The inadequacy of unicast-only systems is more significant for these forward-looking applications, especially in distributed systems where data needs to be constantly updated and synchronized.

As a result, multicast is considered as a better and more efficient transmission mechanism to convey information among a group of people. Instead of transmitting data separately to each member of a group, a multicast data source only sends one copy of data which is replicated as necessary when propagating in the network towards the receivers. This is extremely helpful for small and less capable devices to disseminate data to a large set of receivers, since the intelligence in the network helps the source to reduce the load on both its CPU and its access link. Scalability is another center of interest for multicast, as it reduces the amount of total traffic injected into the network by each multicast session. This allows multicast to scale to very large group sizes and enables group communication without traffic explosion in the network as in the case of unicast.

For the past ten years, multicast has been an active research area, with many standards proposed, applications developed and even a few multicast sessions transmitted. Nevertheless the current Internet multicast, or IP multicast, has left many unresolved issues hindering its own development as well as the development and deployment of multicast applications.

### 1.2 Issues in IP Multicast

The major deficiency of IP multicast can be traced to its roots in layer 2 multicast mechanisms, intended for shared medium networks like Ethernet and FDDI. In this model, both senders and receivers are not known to each other or to the network, and anyone is allowed to send to and receive from a multicast group at any time. This uncontrolled form of group semantics causes tremendous difficulties for routing and session management protocols to achieve scalability and efficiency. Since a multicast address is basically a random
number that has no geographical or topological meaning, it is therefore necessary to store group information globally in the network even at places where it is not needed. Consequently, global coordinations within the network, e.g. address allocation and join/leave notifications, etc., are hard to achieve in a fast and scalable way without incurring large control traffic overhead.

Secondly, the anonymity of the IP multicast model raises security concerns, since it magnifies the impact of denial-of-service attacks. Only recently did the new IGMP V3 [4] address this issue by adding source filtering support, where a list of sources can be specified as "included" or "excluded" senders for the group. However, this method is useful only when the malicious attackers are identified a priori. Overall, the open architecture of IP multicast lacks adequate access control and authentication mechanisms to protect network and session members from malicious attacks.

Lastly, at the system level, the lack of end-to-end transmission control, i.e. error control and congestion control, and the lack of adequate system APIs have hindered the development of multicast applications. Although many error-control mechanisms have been devised, a deployable congestion control protocol remains an open research issue. Above all, a flexible framework that captures the transmission requirements of different multicast applications is still lacking.

1.3 Proposed Approaches using Overlay Network

We propose an Advanced Multicast Infrastructure (AMcast) as an alternative to the current global domain IP Multicast. AMcast consists of servers deployed in the network as multicast proxies which provide connectivity and management functionality to end users. These servers are geographically dispersed in metropolitan areas where most network traffic is generated. For each multicast session, proxies representing session users form an overlay multicast tree, where each tree branch consists of a unicast connection between two member proxies. Session data is then relayed on this "virtual" multicast tree among proxies and disseminated from each proxy to its local users.

The most noticeable advantages of AMcast are: a) its ability to reduce the load on data sources and load on their network access bandwidths, achieving just the same ability as in native multicast but without the need of explicit network support; b) its ability to provide application-customizable routing and transporting flexibilities which are far more complicated to implement at the network layer; c) its scalability of handling a large amount of sessions, as contrasted to IP multicast which does not scale well with number of multicast groups; and d) its ease of deployment and immediate availability since it does not require modification to the network as IP multicast does. The tradeoff in achieving these functionalities is the packet duplications at proxy servers' interfaces and close-by network links, resulting in less data bandwidth efficiency compared to native IP multicast. For sessions that have very high fanout and bandwidth intensity, such as Internet TV, this inefficiency may become significant enough to hinder overall performance. However, for the majority of other applications, the gain on the reduced source "last-mile" load and additional transport support far out-weighs the loss of bandwidth.

From a network design point of view, the cost model of AMcast differs significantly from traditional networks. We will elaborate on this point later in the following section. In this proposal, we focus on issues in dimensioning access bandwidth of AMcast servers and propose load balancing routing algorithms. We also provide initial investigations of simulated performance of our network design process. Additionally, we propose a flexible transport layer which deals with error control and flow control in the AMcast architecture.

To facilitate development of multicast applications for small groups, we also propose an Application-level Multicast Infrastructure (ALMI), as an end-system only multicast enabling mechanism. ALMI is a middleware communication package that utilizes centralized routing control; however the data path still uses the multicast transmission paradigm, thus achieving better data efficiency than point-to-point communication. The focus on ALMI is to experiment the interfacing between applications and a rich set of middleware functionalities, including data reliability, transcoding and session security.
2 Related Work

Amid the inability of the delivery of IP multicast and the emergence of network services such as content services and application services, there is an increasing demand for self-organized overlay networks. However, in spite of the much hyped content distribution networks and a flurry of peer-to-peer applications, little is done in the research community to formally study the implication of overlay network from a network design point of view and investigate the potential new opportunities of providing better services for the next generation of network applications. It is in this context that we believe AMcast and ALMI, as well as less than a handful of other research proposals, are among the first to advocate application-level multicast (or overlay multicast) as an alternative to IP multicast.

Yalcast [13], aims to extend the Internet multicast architecture and defines a set of protocols 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 to inform new members about several current members in the tree but the rendezvous host is not connected to the multicast data paths. Yalcast 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, Yalcast envisions the deployment of IP multicast within small “network islands” and provides a rudimentary architecture for global multicast.

In contrast to Yalcast, Endsystem Multicast [6] is aiming towards small and sparse group communication applications much like ALMI does. In Endsystem Multicast, group members are self-organized into multicast trees using a routing protocol similar to DVMRP [11] that creates source-based multicast trees. It requires 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.

Scattercast [5] is an application-level infrastructure service engineered for content distribution. It uses shortest path routing to build source-rooted distribution trees. In order to build a routing table at the application level, a mesh is first built among multicast proxies using a protocol called Gossamer for neighbor discovery. Additionally, a customizable transport is defined in Scattercast, which prioritizes application data based on their content, for example text data is prioritized for reliability, while losses in image data are ignored to some extent.

All three of the above schemes try to leverage the existing multicast routing protocols and re-apply them at the application level. Although, at the application level, the complexity of IP routing is greatly reduced, since the number of nodes involved is much fewer than the number of routers all over the Internet, there is additional complexity introduced by the sensitivity of end host measurement and the potential of end host unreliability. Additionally, the cost of building an application-level multicast tree differs greatly from the cost of building a network level multicast tree and results in very different network design and routing perspectives. And it is these aspects that we plan to evaluate carefully and explore fully in this thesis work.

3 Multicast Overlay Network Service

In this section, we describe in detail the design of the AMcast overlay network, including its cost model, load balancing routing algorithms and the procedure of network capacity assignments. We also present transport layer components and their functions in AMcast as well as proposing a system architecture that encompasses AMcast routing, data forwarding and transport services using crossbow plug-ins [10].

3.1 Overview of AMcast Architecture and Design Issues

Figure 1 illustrates the network interconnections of AMcast servers with ISP networks and end users. An AMcast server entity is a cluster of servers physically co-located in a geographic region, typically a metropolitan area where most network traffic is generated. To obtain network connectivity, an AMcast server establishes service level agreements with its regional ISPs. The obtained interface bandwidths are used in two ways: one is to access the backbone network and participate in the cross-region multicast spanning trees for sessions it
is a member of; the other is to distribute multicast data to regional subscribers over a star of unicast connections centered at the server or over a locally available IP multicast group. The interface to the regional network is typically separate from the backbone access interface. Moreover, network structures similar to cable networks can be used effectively for regional data distribution. Therefore, our design problem focuses on the overlay network consisting of AMcast servers and their interconnections to the backbone network.

![Diagram of AMcast architecture]

Figure 1: Overview of AMcast architecture

Unlike IP multicast, AMcast creates a virtual multicast tree among participants, with each tree branch as an end-to-end unicast connection through the underlying Internet infrastructure. Each session server not only serves as a data recipient, but also as a data forwarder to its next hops in the multicast tree. These differences cause unique network design perspectives with regards to the cost metric and network routing constraints.

- **Network reachability**: The overlay network among AMcast servers is a fully meshed network, as each node is able to reach everybody else in the network via unicast connections. Therefore, unlike in IP multicast where a path from one router to another is confined by its physical connectivity, an $n$-node application multicast session could have $n^{n-2}$ different spanning trees. A cost effective network design will have to consider load distribution on all of these alternative routes.

- **Network cost**: Historically, the cost of a network is the summation of tariffs paid for each individual link in the network. This is certainly true for network provisioners who have to physically deploy the links or lease them from others. But from an application or application server's point of view, network cost is actually the total amount paid to gain *access bandwidth* at each service provider's site to the backbone network. This divergence of the cost metric has a deep impact on both design and routing strategies: a minimum cost network no longer consists of minimum spanning trees but of trees that make the best use of the access bandwidth at each node.

- **Routing constraints**: Traditional IP multicast routes through a shortest path tree to minimize delay from source to members, i.e. reducing the number of links needed to carry session traffic. Building the network at the application layer surely gives the flexibility of tailoring application needs to routing strategies. One of these metrics is the network delay experienced by an application. For applications such as streaming media or conferencing, satisfying constraints on the maximum delay between any pairs of participants translates to quality assurance for the application.

Therefore, we state the main design problem of the AMcast overlay network as follows: *Given a fixed total cost of obtaining access bandwidths at AMcast servers, how to assign bandwidths to individual servers and how
to route data traffic such that we will achieve good load balance among servers and attain high utilization across all server interfaces as well as small end-to-end delay. The two parts of the problem, bandwidth dimensioning and multicast routing, are tightly coupled. The routing algorithm is directed by the difference between a server's assigned capacity and its actual traffic load when creating a multicast spanning tree. On the other hand, the dimensioning process must know the intrinsic property of a routing algorithm such as the possible traffic concentration points, and assign bandwidth to servers accordingly. There are two other important parameters involved in the design process: a) the location, or the topology of AMcast servers; and b) the traffic information used for bandwidth dimensioning. We will discuss our models for these two parameters later in this section.

3.2 Multicast Routing in Overlay Network

There are two key parameters that characterize the performance of a multicast routing algorithm in the context of overlay networks. One is the diameter of a session multicast tree which determines the maximum end-to-end delay experienced by users; the other is the residual bandwidth at session servers which indicates the balance of traffic load across the servers. Intuitively, we can see that it is difficult to optimize both parameters at the same time. Figure 2 illustrates an example, assuming all nodes are of equal capacity. The minimization of multicast tree diameter results in higher traffic concentration on geocentric servers; while the optimal load allocation results in high end-to-end delay. In order to compensate for both parameters, we formalize two routing algorithms, each of which treats one parameter as a routing constraint and optimizes the other.

(a) Optimize on diameter  (b) Optimize on load-balancing (c) Optimize on both

Figure 2: An Example of Tradeoff Between Diameter and Load-Balancing

Definition 3.1 Minimum diameter, degree-bounded spanning tree (MDDBST)

Given an undirected complete graph $G = (V, E)$, a degree bound $d_{\text{max}}(v) \in N$ for each vertex $v \in V$; a cost $c(e) \in Z^+$ for each edge $e \in E$. Find a spanning tree $T$ of $G$ such that for each $v \in T$, degree of $v$ satisfies $d_T(v) \leq d_{\text{max}}(v)$ and the diameter of $T$, $\text{dia}(T)$ which is the cost of the longest simple path in $T$, is minimized.

The above problem is NP-complete, since its special case when every node has a degree constraint of two, is the same as the Traveling Salesman Problem (TSP) [15]. We have developed a heuristic algorithm for the MDDBST problem, which is a greedy algorithm similar to Prim's algorithm for Minimum Spanning Tree [8]. Figure 3 shows the steps of the algorithm. We denote $\delta(v)$ as the longest path of $v$ to any other nodes in $T$. Similarly to Prim's algorithm, we start from a single root node. At each step when adding a new node $v$ to the existing component $T$, we select the node that has the smallest $\delta(v)$. Then, we update the nodes in the existing component who have changed their longest path because of the new node, $\delta(u) = \max(\delta(u), \text{dist}_T(u, v))$. Finally, for each node $v$ not in the component, we update its parent to node $u$ which, without violating the degree constraint, gives $v$ the smallest longest path. In terms of the tree diameter, the performance ratio over the optimal solution is $O(\varepsilon \log n)$ provided that the ratio of the longest edge to the shortest is bound by $\varepsilon \in Z^+$. The proof can be found in [25].

The algorithm fails when it finishes with some vertices having $\delta(v) = \infty$, meaning that we cannot build a spanning tree with the specified set of degree constraints. There are two occasions for this to happen: one
is that the total degree constraints can be less than \(2 \times (|V| - 1)\), which is the minimum total degree required for a session spanning tree; the other is that during the progress of the algorithm, a leaf node may be added to the current tree component and consumes all the spare degrees of the component, leaving the rest of the nodes unconnected. Both of these failures do not occur very often in a real system, as the degree constraints are usually generous enough to avoid them. Only when the system is extremely highly loaded, some or all of the nodes may have stringent degree constraints which cause the algorithm to fail. We can perform a simple feasibility test on the summation of the degree constraints to identify the first type of failure. To remedy the second type of failure, we can add a count of the spare degree of the tree component and defer the addition of a leaf node if it reduces the count to zero.

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G = (V, E))</td>
</tr>
<tr>
<td>Edge cost (c(u,v)), for (u,v \in V)</td>
</tr>
<tr>
<td>Degree constraints (d_{\text{max}}(v))</td>
</tr>
<tr>
<td>Output: (T) with the smallest diameter</td>
</tr>
<tr>
<td>foreach (v \in V)</td>
</tr>
<tr>
<td>(\delta(v) = c(r,u);)</td>
</tr>
<tr>
<td>(p(u) = r;)</td>
</tr>
<tr>
<td>(T = \langle w = {r}, L = {}\rangle;)</td>
</tr>
<tr>
<td>while ((W \neq V))</td>
</tr>
<tr>
<td>let (u) be the vertex in (V - W) with the smallest (\delta(u));</td>
</tr>
<tr>
<td>(W = W \cup {u}; L = L \cup {u, p(u)});</td>
</tr>
<tr>
<td>foreach (v \in W \setminus {u})</td>
</tr>
<tr>
<td>(\delta(v) = \max(\delta(u), \text{dist}_T(u,v));)</td>
</tr>
<tr>
<td>foreach (v \in V - W)</td>
</tr>
<tr>
<td>(\delta(v) = \infty;)</td>
</tr>
<tr>
<td>foreach (q \in W)</td>
</tr>
<tr>
<td>if (\text{degree}(q) &lt; d_{\text{max}}(q)) and (c(v, q) + \delta(q) &lt; \delta(v))</td>
</tr>
<tr>
<td>(\delta(v) = c(v, q) + \delta(q);)</td>
</tr>
<tr>
<td>(p(v) = q;)</td>
</tr>
</tbody>
</table>

| Figure 3: Heuristic Algorithm for MDBBST |

**Definition 3.2 Bounded diameter, residual-balanced spanning tree (BDRBST)**

Given an undirected complete graph \(G = (V, E)\), a degree bound \(d_{\text{max}}(v)\) for each \(v \in V\); a cost \(c(e) \in Z^+\) for each \(e \in E\); a bound \(B \in Z^+.\) Find a spanning tree \(T\) of \(G\) for which \(d_T(v) \leq d_{\text{max}}(v)\), for each \(v \in V\) and diameter of \(T, \text{dia}(T) < B\) and which maximizes \(\min_v(d_{\text{max}}(v) - d_T(v))\).

The above problem is also NP-complete, since its special case when every node has a degree of two, corresponds to the decision version of the TSP problem.

The difference between BDRBST and MDBBST is that instead of trying to minimize the multicast tree diameter, we try to distribute the work of heavily loaded servers to others who are nearby and are less loaded. Overall, this increases the total load that the system can handle at the cost of increased end-to-end delay. In order to distribute load while still satisfying the constraint of end-to-end delay, we introduce a balance factor \(M\) to denote the tradeoff between diameter and load balancing. We vary the previous MDBBST algorithm to take into account this balance factor: at each step when adding a new node, instead of selecting the one node that has the smallest \(\delta(v)\), we select a set of \(M\) smallest nodes and choose one of them that maximizes the minimal residual bandwidth of these \(M\) nodes and their parent nodes. If \(M = 1\), this algorithm is the same as the one shown in Figure 3. On the other hand, if \(M\) equals the number of servers in a multicast session, then the algorithm considers load balancing as the sole routing criteria and serves as an approximation algorithm for BDRBST. For intermediate values of \(M\), it takes both parameters
into account. We have found that small values of $M$ (e.g. 5) provide good load balance while still meeting the diameter bound.

3.3 Dimensioning Server Access Bandwidth

In order to evaluate the routing algorithms experimentally, we need a realistic network environment in which to perform the evaluation. Such an environment includes both a model for the network traffic and a network topology and configuration suitable to carry the traffic. In this section, we discuss our assumptions on the network model used in the simulation and methodologies in assigning access bandwidth to AMcast servers.

In our simulation, we assume that there is an AMcast server site in each of the 50 largest US metropolitan areas and that the amount of traffic generated at each site is proportional to the population of the metro area. We allocate a given total access bandwidth among these servers based on their characterized traffic loads during a dynamic Poisson process. We use a Binomial distribution as the fanout of the multicast sessions.

There are two factors impact the traffic load distribution across the servers. One is the population of the area which determines the number of sessions a server will participate in; the other is the geographical location of a server which decides the degree of a server in a multicast tree, where each tree branch utilizes one additional bandwidth unit at a server’s interface. However, to determine a server’s degree, we must first construct the multicast tree. As the servers are not yet assigned their shares of access bandwidth, the quality of the multicast tree is characterized only by its diameter. Therefore, we construct the multicast tree as a minimum-diameter spanning tree because it provides the best performance tree for a session with minimum maximum end-to-end delay. Once the traffic load is computed at each server, we allocate bandwidth proportionally to each server subject to a total affordable capacity. Realistically, the costs of bandwidth in different metro areas could be different or non-linear depending on the supply and demand in one particular area. To simplify matters and ease understanding, we assume costs per unit bandwidth are equal for all servers.

![Figure 4: Dimension of Sever Access Bandwidth Using Minimum Diameter Tree](image)

Figure 4 shows a set of calibrated servers simulated after the aforementioned procedures. Servers on the x-axis are sorted according to their populations. Clearly, those servers with large populations and those at geographic center locations receive more bandwidth since they either handle more locally generated traffic.
due to a large population, or more transient traffic due to their center locations. For example, Chicago has about 43% of New York’s population but receives 1.3 times more bandwidth.

3.4 Preliminary Evaluations

In this section, we present some preliminary simulation results for our dimensioning process and routing algorithms. The simulation is set up based on a topology consisting of the 50 largest metropolitan areas in the United States [28]. Link cost is measured as the geographic distances between each pair of servers; and each session is assumed to consume one unit bandwidth per link. The diameter constraint on the BDRBST algorithm is 8000 km, which amounts to about 40 ms one-way propagation delay.

3.4.1 Performance On Multicast Tree Diameter

Figure 5 shows the simulated multicast tree diameter for both MDDBST and BDRBST. In this simulation, we do not dimension the network but only restrain a node’s degree by a Binomial random distribution with mean $p$. For MDDBST, we vary the mean of the Binomial distribution while for BDRBST, we fix $p = 7$ and vary the balance factor $M$. In each simulation run, we randomly select over the 50 cities a fixed fanout (or session size), shown as the x-axis. The y-axis shows the ratio of the multicast tree diameter over the geographical distance of two cities furthest apart in a session. This distance is also the lower bound of end-to-end delay for the session.

![Diagram](image)

Figure 5: Performance of MDDBST and BDRBST on Multicast Tree Diameter

The results clearly show that the MDDBST algorithm works very well creating multicast trees with the largest end-to-end delay no more than 1.25 times the absolute optimal. When applied with a smaller degree bound, the spanning tree is more likely to be long legged, resulting in longer diameter. As the session size increases, the density of nodes increases. A session is more likely to have a geographic center which connects to as many other nodes as allowed by its degree bound, resulting in slightly shorter diameter. For BDRBST, the tree diameter increases with the increase of $M$; but overall it is well within the delay constraint. These results suggest that both algorithms are able to route sessions successfully satisfying end-to-end delay requirements.

3.4.2 Performance On Server Load Balancing

In this section, we evaluate the load balancing performance of MDDBST and BDEBST routing algorithms over a dimensioned network. The measure of load balancing is the total session rejection rate under a certain offered load. A good load balancing scheme should produce a higher overall system utilization at an
acceptable rejection rate, for example one rejection every 10,000 requests. We model the dynamic session requests and removals as Poisson session arrivals with Pareto session service time. Session fanout is generated as a Binomial distribution with mean equals to 10. The results are compared with a theoretical lower bound on system utilization, which is computed from a single queuing system where the total server bandwidth amounts to the queue size and each session corresponds to a request whose size equals to the session size.

Additionally, we have tried three alternative bandwidth allocation policies to quantify server traffic load as below.

a) We continuously generate sessions and assign to each selected servers bandwidth amounts to their degree in the multicast tree, we refer to this case as static as sessions are only added but never removed, and network is configured in accordance with its transition from zero traffic to full traffic;
b) We add and remove sessions dynamically and assign bandwidth according to the average carried load at each server during the whole simulation time;
c) In addition to bandwidth assignment in (b), we route the traffic once more on the previously configured network using the actual load balancing routing algorithm. Let $c_i$ and $c'_i$ be the capacity assigned to each server after one and two rounds, respectively. We re-assign bandwidth capacity $c'_i = L_i + \sum \frac{R^n_i}{n}$, where $n$ is the number of servers, $L_i$ the carried load of each server during the second round of dimensioning and $R^n_i$ the average residual bandwidth of server $i$ in second round. Intuitively, the algorithm converges since the load balancing algorithm always tries to equalize the residual bandwidth of each server by adding more transit traffic to servers with available capacities while offloading smaller servers, and the dimensioning algorithm also reduces the excess capacities of big servers and add them to those whose bandwidth are less abundant.

![Figure 6: Performance of MDDBST and BDRBST on Load Balancing](image)

Figure 6 shows the performance of the two routing algorithms. The fact that BDRBST outperforms MDDBST significantly, suggests that it is worthwhile to endure a little higher delay in order to achieve a higher system utilization. Additionally, the different bandwidth allocation strategies also indicate that the closer the dimensioning process tied with the routing algorithm, the better the performance. When dimensioned with two round assignment, the performance of BDRBST algorithm approaches within 10% of the lower bound.

Furthermore, we have also simulated the routing behaviors under traffic variances. As the actual carried traffic load in the network may differ from our projected traffic load, i.e. traffic load we used to dimension the network, a good routing algorithm should adapt to such variances and achieve similar performance as before. We model the traffic variance as Gaussian noise added to each metro area's population, thus changing servers' probabilities of participating a multicast session and traffic intensity across the network.

Figure 7 illustrates the average results of multiple runs for all four differently dimensioned network. For the desired network operation point of $10^{-4}$ rejection ratio, a 50% noise results about 15% performance drop at most. The relative performance across four network configurations is preserved as traffic noise increases.
Figure 7: Comparison of Traffic Handling for Various Dimensioning Method

3.5 Transport Layer in AMcast

The role of the transport layer is to perform common functions and maintain common properties of data flows for distinct applications while alleviating individual applications from performing these tasks repeatedly. Additionally, the transport layer maintains flow fairness in the network and forces applications to be "good citizens" of the network world. These lead to two important roles for the transport layer: error control and flow control.

3.5.1 Error Control Issues

Error control concerns data reliability, consistency and sometimes data ordering. Packet errors can occur for various reasons including: link errors, buffer overflow at routers or OSs, etc, while packet re-ordering can result from multiple flow paths. Typically, these errors are handled by request and retransmission schemes, for example TCP for unicast connections; or by data encoding schemes, such as forward error correction. From Table 1, we can see that applications can be loosely divided into three categories according to their reliability requirements: a) total reliability with ordering, for example, a file transfer, where the receiving application can simply write all received bytes into a new file without worrying about missing words or re-ordered words or sentences; b) timed reliability with no-ordering, this typically implies that the application has a certain amount of buffer space and can make use of any data that is recovered within the buffer depletion time, but data recovered outside this time frame is rendered useless. In this case, ordering is
typically maintained by the application itself using timestamps and various resynchronization techniques for encoded bit streams; c) no reliability, where applications use specific coding and error recovery mechanisms. Therefore, a multicast error control module has to supply functions for the first two categories of applications.

The difference between error control in multicast vs. unicast is that data can be lost on all or only some receiver paths. It is therefore quite inefficient if each of these losses is retransmitted by the sender alone, as the aggregation of these losses may account for a significant portion of the whole data flow. Additionally, it is meaningless to retransmit data to those who have already received the same packets. Instead, we should distinguish the loss branch from others and recover data from its nearest neighbors. To summarize, if error control in unicast is to ask the question "which packet is lost", the equivalent question asked by a multicast error control scheme is "who lost which packets and who has those packets".

Fortunately, answering this question in AMcast is not as hard as in native IP multicast. As AMcast servers form an additional layer between networks and end users, they can leverage the existing point-to-point reliable transport protocols to prevent packet losses on each "virtual link", or provide temporary buffers at intermediate hops for fast recovery on a link-by-link basis. The buffer size at each hop is a configurable parameter depending on resource availability as well as the time frame to render application data. If total reliability is required, a data source should buffer every packet that is not acknowledged by some receivers, which could be done at the data entrance to the AMcast network.

3.5.2 Flow Control Issues

Flow control in general is a technique to achieve efficient and fair use of the network by preventing users from sending an excessive number of packets that may exceed network capacity or exceed its fair share of bandwidth. At the network layer or below, flow control techniques include buffer management schemes, packet pacing and dropping policies, or explicit rate control schemes. However, if flow control is not adequately done at the network layer, such as in today's Internet, it is the end-systems' responsibility to protect the network from congestion and congestion collapse.

One key issue for multicast flow control is to deal with heterogeneous receiver capacities. Since the degree of congestion varies, sometimes wildly, from one receiver path to another, there is no single rate that a data source can easily identify and adjust its transmission rate to. A multicast application, therefore, has to make one of the following two decisions, depending on its own data reliability and timeliness requirements. Streaming based applications can choose to simply drop packets right above the congestion branch, and if data is encoded as layers, dropping can be done more intelligently to maximize performance. This also means that each intermediate hop can transmit at different rates on its path to downstream neighbors according to individual path characteristics, resulting in different performance experiences at receivers. On the other hand, for applications in which data reliability takes precedence over timeliness and which do not have the luxury of data encoding, a data source can choose to transmit at the rate that conforms to the most congested receiver path to avoid losses. This results in performance degradation for all receivers, although maybe only one of them is really congested. To avoid overwhelmingly slow receivers, we can set a minimum flow rate to drop out a receiver whose goodput is below the minimum value and use out-of-band methods to transmit data to this receiver.

Apart from receiver heterogeneity, there are several other issues in multicast flow control:

Congestion Detection and Feedback Without network layer support, an end system data source does not have any perception of where and when congestion has happened. Thus, it requires receivers to report their quality of reception directly to the data source or to relay reports through their upstream neighbors. These reports happen periodically because of the dynamics of network conditions. A receiver can infer congestion from its packet losses. However, the question arises of how many losses would indicate congestion or how long a period should a receiver calculates its loss rate over? The frequency of reports and rate adjustment decides the responsiveness of a flow control scheme. A delay in loss report or a prolonged period for loss rate calculation, postpones sources' reactions to network congestion and may cause network instability. Conversely, too frequent reports not only waste bandwidth but also result in sources' over-reaction to congestion, leading to poor performance.
Congestion Aggregation In the case of source adaptation, when receivers report congestion from their individual experiences, the data source should aggregate these signals to separate out independent congestions, since adaptation to the summation of these signals results in excessive rate drops and poor performance. The aggregation could be done by selecting the worst receiver, the one with the smallest available window or lowest reception rate. This requires the source to keep separate states for each paths. Alternatively, aggregation can be time-based, but it is rather difficult to select the right time interval since each receiver has different round-trip time from the source, which changes dynamically due to variations in network queuing delays.

Rate Adaptation Rate adaptation concerns how fast a source should decrease its data rate, or how many layers an upstream node should drop in time of congestion; and conversely, how fast a source should increase rate or how many layers an upstream node should add when congestion no longer exists. The adaptation strategy impacts the fair use of bandwidth for competitive flows. Previous work suggests two types of fairness, one is max-min fairness [3], where fairness is defined as the fair share of bandwidth on the bottleneck link; the other is proportional fairness [17], where the fair share of bandwidth is inversely proportional to a flow’s response time. In addition, a rate adaptation strategy should avoid rate oscillation and reach the steady state as early as possible.

Although error control and flow control are two separate issues, their design spaces are often correlated and their policies can be conflicting. For example, it is inefficient if the flow control policy drops packets aggressively for an application that requires data reliability. Therefore, the design of the transport functions must be within a specific application context. We discuss below AMcast transport support for two specific types of applications: one is many-to-many reliable applications, the other is conferencing applications.

3.5.3 Many-to-many Reliable Applications

Applications in this category include file transfers, data replication and synchronization services, etc. These applications are typically non-time critical and a data source often by default stores all the data and can retransmit any missing data upon request. We propose to use hop-by-hop TCP with coarse grained periodic Acks to achieve data reliability and TCP-compatible flow control. Figure 8 illustrates this option for a session instance.

![Transport Function for Supporting Reliable Applications](image)

Figure 8: Transport Function for Supporting Reliable Applications

In this design, there is a single TCP connection covering each hop and packets from different upstream nodes are merged into this single TCP stream. A node processor reads packets from the input queues in a round-robin fashion, and temporarily stores them in a buffer before sending them to downstream neighbors. There is one buffer for each input queue. The send() function is implemented as non-blocking in this case. When a downstream neighbor is congested, as node C in figure 8, its output queue starts to fill up. There is a bitmap associated with each buffer, indicating over which connections the buffered packet has not been successfully sent. Node D will not read any more packets from A’s input queue unless the bitmap indicates A’s buffered packet has been sent (or queued) to all the downstream neighbors. Therefore, when C’s output queue fills up, it creates back pressure at A’s input queue and causes the TCP connection to advertise a smaller receiver window size. Consequently, A slows down its sending rate. In the mean time, if B is also an
active sender, the scheme will have the same effect to slow down B’s sending rate. Contrarily, it will not slow down C's sending pace unless one of the connections from D to A or D to B is congested. On the other hand, we do not explicitly place any fairness criterion on how A, B and D should share the connection bandwidth to C, because streams from A and B include streams from a number of other nodes further upstream whose size can vary depending on the number of active senders at the moment. For multiple sessions sharing a single congested downstream link, the effect of statistical multiplexing allows bandwidth to be distributed evenly among sessions.

In order to achieve total reliability, a data source must be able to retransmit any missing packets requested by any receivers. Link-by-link TCP by itself is not enough to achieve end-to-end data reliability since losses could happen during node or network failures which force a tree branch to be reconstructed. In order to detect such losses, a receiver need to remember the last highest sequence number from each data source and re-synchronize the data streams every time an old connection is torn down and a new connection is set up. Instead of the TCP sequence numbers, an application source or the AMcast server representing the source need to provide additional data identifiers for the purpose of re-synchronization. Furthermore, we can use coarse grained Acks to inform the sources of all the packets that are received correctly by all receivers. More discussions on this can be found in section 4.3, where we implemented a similar multicast Ack scheme and data mapping scheme in ALMI.

3.5.4 Video Conferencing Applications

As audio/video stream based applications become more and more common in the Internet, transport layer support for these applications is crucial to their success. Currently, most of these applications’ data are encoded, quite often in a format that supports different bandwidth transmissions, such as the MPEG format [19] and the SureStream™ [7] format from Real Networks. Although these data coding scheme are quite coarse grained from a flow control point of view, it provides a better way than FIFO for an upstream node to decide which packets to drop for each of the heterogeneous downstream receivers, i.e. drop packets from the coding layer that contains the highest frequency information. In addition, if a node discovers a session’s incoming stream rate is higher than the highest output rate for this stream, it sends a feedback packet to its upstream neighbor indicating the desired reception rate. Conversely, a node can also ask its upstream node to speed up the data rate. This provides explicit rate control in an end-to-end manner but still allows heterogeneous data rate on each tree path. When there are multiple senders, the upstream node also needs to decide whose data to discard during congestion. Assuming for a single application, all senders will use the same encoding scheme, then the node can discard the packets from the sender whose data frequency is the highest.

When there are multiple video streams competing against each other, a node must maintain their individual rates by deploying queueing mechanisms at the output interface. Since each stream’s rate may be determined by downstream feedback, it is not desirable to enforce equal bandwidth sharing in this case. Rather, we need to decide dynamically the weights of individual queues apportion to the corresponding stream’s rate.

The error control for this type of applications can be time-based request or forward error corrections. If a request-retransmission scheme is used, an upstream node needs to have buffer space large enough to cover the round-trip time to the furthest away next hop receiver plus the stream buffer depletion time. Additionally, since the flow control policy may drop different packets for each receiver, the sequence number for the same data object may differ for each of them. Therefore, the receivers need to have knowledge of the encoding scheme and explicitly identify a data object in their retransmission request.

3.5.5 Issues in Inter-flow Fairness

We have so far discussed flow control for bandwidth sharing for the same type of data streams. For different types of data streams, e.g. when a reliable data stream is competing with a video stream, they also need to share the bandwidth in some reasonable manner. The issue is that an equal share of bandwidth for both streams may not be desirable, for example, the quality of a video applications is highly dependent on its receiving network throughput while file transfers can more easily tolerate reductions on throughput.
One option is to provide minimum rate guarantees for both streams and allow a conferencing stream to obtain extra bandwidth if all the TCP streams have met their minimum rates. Such rate guarantees can be met by deploying fair queuing schemes [2, 26, 27] at the output interface. However, one difficulty is that TCP always oscillates and transmits in bursts. Additional delays in Acks may have adverse impact on its performance. It requires further research to examine the impact of fair queuing schemes on TCP.

3.6 System Architecture of AMcast Servers

We have so far discussed issues in the network part of AMcast, including issues in network design, routing and transport support. To explore the performance and scalability of this system, we also propose a system architecture that implements the actual data forwarding, multicast routing control and transport functionality. Part of the system can be integrated with the Crossbow [10] plug-in platform for its on-the-fly kernel function maneuverability and speeding up kernel data processing path.

3.6.1 System Software Architecture

![AMcast Server Software Architecture](image)

Figure 9: AMcast Server Software Architecture

Figure 9 depicts the system data paths of three possible combinations of error and flow control mechanisms. From left to right, they correspond to hop-by-hop TCP with total reliability; layered data encoding with no or timed reliability; and rate-based flow control with reliability. Vertically, data paths are divided into four layers. The topmost layer is mostly for session control purposes such as functions performed during session initialization or member addition, and setups of kernel plugins; the second layer performs data duplication and forwarding tasks according to routing table set by AMcast routing daemon; the third layer leverages the existing OS network layer and crossbow framework to implement the appropriate transmission mechanisms set by session controls; the bottom layer includes the conventional unicast IP routing and forwarding functions. There are also two system wide modules for session management and routing purposes. The function of AMcast routd is to update other servers’ states and construct multicast trees for session requests based on its current knowledge of the network state. The session management module deals with end user subscriptions. It listens to user's session join and leave request and directs session flow to the current subscribers.
3.6.2 Server Cluster Architecture

Figure 10 illustrates the design of an AMcast server cluster. The clusters are connected to the network by a switch with load balancing and packet classification capabilities. The switch control (SC) sets up interconnections among servers and the output paths from each server to all other interfaces. Both unicast and multicast routing functions reside at the switch control. The SC also collects load information from each server periodically and is able to distribute a new session request to the least loaded server. The SC communicates with the servers to setup their AMcast forwarding tables for each session. The input port processor (IPP) implements packet classification functions on a line card, where the lookup table is constructed by the SC.

![AMcast Forwarding Table of Server IV]

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Flow ID</th>
<th>Neighbors</th>
<th>Delegator</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>3</td>
<td>A B C</td>
<td>II</td>
</tr>
<tr>
<td>5000</td>
<td>2</td>
<td>D E</td>
<td>III</td>
</tr>
<tr>
<td>5000</td>
<td>1</td>
<td>F G</td>
<td>self</td>
</tr>
<tr>
<td>5001</td>
<td>3</td>
<td>A C G</td>
<td>self</td>
</tr>
</tbody>
</table>

![Control Path Delegation]

When a session size grows larger, a single server may be unable to handle all the data processing that is needed. Therefore, the SC spawns other servers to help. The splitting of the processing power among servers can be done on the basis of splitting the neighbor set, i.e., each server handles the data path for a subset of the neighbors within a session. As shown in Figure 10, the original session neighbors include only A and B, and the session is only handled by server II. When neighbors D, E, F, G also join the session, the SC selects server III and IV, on the least-load basis, to join the session and handle the connections to the new neighbors. The incoming session packets are multicast from the input port to all three servers, while the outgoing packets are unicast by each server to their individual downstream neighbors. The issue here is to coordinate the TCP or UDP control information among the servers. One way to achieve this is by delegating the server which handles a particular neighbor to send back control information to that neighbor. For example, server II will send TCP Acks to neighbor A, B and C; and server III to neighbor D and E. When a server, other than server III, receives Acks from D, it simply discards them. The same function can be applied to applications that use a request-retransmission scheme for error control as well, where retransmissions are treated as control information and are only handled by the delegated server.

4 End-system Only Multicast Service

To facilitate faster application design and deployment, and support multicast sessions that only involve a small number of users, we have designed and implemented ALMI [22], an application-level group communication middleware, which builds multicast trees directly among end users' systems. In this section, we describe the communication channels provided by ALMI and its related protocol operations for both ALMI group controller and group members. We also describe operations related to multicast tree generation and criteria of tree updates and stability issues. One of the advantages gained in ALMI is its value-added application specific components, which we will also describe in detail.
4.1 Overview of ALMI Communication Architecture

An ALMI session consists of a session controller and multiple session members. The session controller is a program instance, located at a place that is easily accessible by all members. It may be collocated 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 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 bidirectional unicast 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 members for reasons we will explain later in this section; it does not indicate direction of data flow.

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. Session members also monitor 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 round trip response delay. These delay measurements are then reported to the controller and serve as the costs used to calculate a Minimum Spanning Tree.

![Figure 11: ALMI Packet Header Format](image)

ALMI uses a common packet format to carry both data and control packets, shown in Figure 11. The content of this packet header is rather straightforward. 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, e.g. registration messages from hosts to the controller, connection request and acknowledgment between parent and child, distribution tree messages, etc. The Tree Incarnation field is to prevent loops and partitions of the multicast tree, and we explain it later in this section.

Fault Tolerance

ALMI takes a 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.

Loop Prevention and Data Reliability

In order to prevent loops and partitions in the multicast tree, all ALMI packets bear a Tree Incarnation field. 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 same 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 a tree version number that is 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.

4.2 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 (MST) 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 round trip delay, measured at the ALMI layer, as the performance metric. The choice of latency as metric is due to its importance to most applications and that it 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 a low delay, low bandwidth link to carry its traffic.

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 mesh and have \( O(n^2) \) message exchanges to compute the best multicast tree. However, as group size grows, it becomes unscaleable to have a 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 a 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 the 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. As described above, an ALMI multicast tree is a degree-bounded spanning tree. Since most end hosts tend to be on access links 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 networks, 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 and re-initiation of the data connection. Moreover, data packet may be lost or duplicated during a tree transition, 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 tree switching are user-configurable parameters.

4.3 Design of Application Specific Components 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. 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.

End to End Data Reliability

A TCP-equivalent reliable transport protocol for multicast communication has been the subject of active research in recent years [20]. 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 aggregates requests and retransmits data without the need for router support or knowledge of session topology. 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 “no data” response is sent back to the requester, i.e., the head of the loss subtree. The requester 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, the requester at the root of the loss subtree forwards them to downstream receivers. 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 requests all the way up to the source can sometimes 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 has much less processing delay.

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.
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 the 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 buffered packets with sequence numbers 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.

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 3-tuple containing <file name, offset, length>. Other more sophisticated mechanisms such as hierarchical data naming schemes [9,23] can be incorporated as well, to achieve better flexibility and efficiency.

Other Components

There are many other functions that could be incorporated into ALMI, such as delay constraints for real-time sessions, access control for private multicast sessions, etc. In ALMI, an application delay bounds can be achieved by constraining the diameter of the minimum spanning 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.

4.4 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 maintenance, 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 [30], 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.

![Random Graph (500 nodes, p = 0.01)](image)

**Figure 12: Cost Comparison of ALMI MST and Shortest Path Tree in Random Graph**

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 12 and 13 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 12 consist of 500 nodes with an average node degree of 5, and transit-stub graphs in Figure 13 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 [30]. 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 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 12 and 13. 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 12 and 13, 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 [12] 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 [29], 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.
4.5 Experimental Evaluation of ALMI

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.

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 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 14: Evaluation of ALMI MST in WAN Test](image)

Figure 14 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 [14, 21, 24] 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 Research Plan

We plan to investigate the feasibility of AMcast using both simulation and implementation. The focus of the simulation study will be on the algorithmic aspects of the network design process and routing algorithms. Current results show that BDRBST with a two round dimensioning strategy achieves the best performance, but there is still plenty of room to improve. Additionally, we need to look at different network topologies and traffic mixes and observe performance under these different conditions. On the implementation side, we plan to investigate the scalability issue of the system as well as the performance for some of the proposed transport layer functions. Specifically, we are seeking collaboration in implementation of multicast flow control with layered data encoding. And it is also possible to test the performance of the error control mechanism through a prototype system. For flow control of bulk data, since this will involve larger scale network topologies and flow mixes in order to examine the responsiveness and fairness of the protocol, we plan to use the network simulator (Ns2) [16] for evaluation purposes.

The development of ALMI has been conducted through the past year. Currently, integration with applications is underway. The project of FGP key server synchronization [18] is the first application to fully function using ALMI as the underlying communication channel. Other toy applications include an ALMI remote distribution service which has been used for performance tests in earlier work. We plan to continue the refinement as well as the development of the middleware package for other types of applications.

![Time line of Research Plan](image)

Figure 15: Time line of Research Plan

References


