NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Early Experience with an Internet Broadcast System Based on Overlay Multicast

Yang-hua Chu   Aditya Ganjam   T. S. Eugene Ng
Sanjay G. Rao   Kunwadee Sripanidkulchai   Jibin Zhan
Hui Zhang

December 2003
CMU-CS-03-214$_3$

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was sponsored by DARPA under contract number F30602-99-1-0518, and by NSF under grant numbers Career Award NCR-9624979 ANI-9730105, ITR Award ANI-0085920, and ANI-9814929. Additional support was provided by Sloan Research Fellowship and Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, or the U.S. government.

Keywords: overlay networks, Internet evaluation, peer-to-peer, multimedia broadcast

Abstract

In this paper, we report on experience in building and deploying an operational Internet broadcast system based on Overlay Multicast. In over a year, the system has been providing a cost-effective alternative for Internet broadcast, used by over 3600 users spread across multiple continents in home, academic and commercial environments. Technical conferences and special interest groups are the early adopters. Our experience confirms that Overlay Multicast can be easily deployed and can provide reasonably good application performance. The experience has led us to identify first-order issues that are guiding our future efforts and are of importance to any Overlay Multicast protocol or system. Our key contributions are (i) enabling a real Overlay Multicast application and strengthening the case for overlays as a viable architecture for enabling group communication applications on the Internet, (ii) the details in engineering and operating a fully functional streaming system, addressing a wide range of real-world issues that are not typically considered in protocol design studies, and (iii) the data, analysis methodology, and experience that we are able to report given our unique standpoint.

1 Introduction

The vision of enabling live video broadcast as a common Internet utility in a manner that any publisher can broadcast content to any set of receivers has been driving the research agenda in the networking community for over a decade. The high cost of bandwidth required for server-based solutions or content delivery networks, and the sparse deployment of IP Multicast are two main factors that have limited broadcasting to only a subset of Internet content publishers such as large news organizations. There remains a need for cost-effective technology for low-budget content publishers such as broadcasters of seminars, workshops and special interest groups.

Recent work in Overlay Multicast [14, 9, 19, 7, 21, 30, 39, 22, 34, 25, 41, 10, 5] has made the case that overlay networks are a promising architecture to enable quick deployment of multicast functionality on the Internet. In such an architecture, application end-points self-organize into an overlay structure and data is distributed along the links of the overlay. The responsibilities and cost of providing bandwidth is shared amongst the application end-points, reducing the burden at the content publisher. The ability for users to receive content that they would otherwise not have access to provides a natural incentive for them to contribute resources to the system.

Most of the existing work, including our own earlier work [9, 8], focus on issues related to "protocol design," and evaluate their potential using simulation or university-based Internet test-beds. We believe that an equally important and complementary style of research can be conducted using an "application-centric" approach. This approach involves the wide-spread operational use of an application by real users, and letting the experience gained direct the research process. The more content publishers and receivers rely on the application, the stronger the case for Overlay Multicast, validating its relevance as a research question. In addition, the unique experience obtained in the process leads to important insight that can motivate future research in the area.

In adopting the "application-centric" approach, our primary consideration was to provide a useful and deployable tool to the general public, and reach operational status as quickly as possible. Therefore, we identify and address a wide range of issues, some of which are not typically considered in protocol design studies, but affect the successful deployment of Overlay Multicast. Our system copes with dynamics in user participation, adapts to application performance and Internet dynamics, supports users that have a wide range of network bandwidth and supports users behind network address translators (NATs) and firewalls. We have built supporting mechanisms such as logging receiver performance, monitoring of system components, and recovering from component failures. In engineering our system, we have adopted simple or natural solutions, with the provision that the design decisions could be revisited in the light of future experience. This approach has accelerated the deployment of the system, and, consequently has led to faster feedback from real deployment.

The challenges involved in obtaining operational experience we report in this paper must not be underestimated. First, we have invested significant effort in convincing content publishers and event organizers that it is worth their while to experiment with the new technology. Second, while we have made earnest efforts to get our system deployed, the participation of viewers in our broadcasts depends on a range of factors not under our control, including

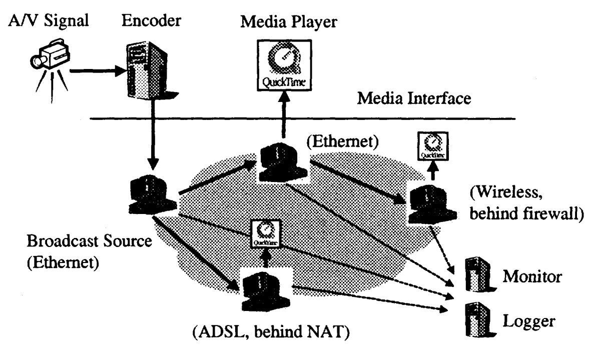

Figure 1: Broadcast system overview.

the content we have access to. Third, unlike conventional research experiments, we have frequently had to work under the pressure to succeed in even our earliest broadcast attempts. Failures would significantly deter event organizers and limit future adoption of our system. One consequence is that it is critical to adopt robust, stable and well-tested code – a performance refinement that may seem trivial to incorporate may take months to actually be deployed.

In over a year, we have been building an operational broadcast system based on Overlay Multicast and deploying it among more than 3600 real users in real Internet environments for over 20 events. We view the design and deployment effort as an ongoing process, and report on the experience accumulated so far. Overall, our experience confirms that Overlay Multicast is easy to deploy and can provide reasonably good application performance. In addition, we believe that our unique set of data, analysis methodology, and experience are useful to the research community.

The rest of this paper is organized as follows. In § 2, we present an overview of the system. § 3, 4, and 5 presents the deployment experience, analysis methodology, and performance analysis of our system. § 6 presents key design lessons learned from the experience that are guiding the future research directions.

2 System Overview

Figure 1 gives a high-level overview of our broadcast system. The encoder takes the multimedia signal from the camera, converts into audio and video streams, and sends to the broadcast source. The broadcast source and receivers run an overlay multicast protocol to disseminate the streams along the overlay. Each receiver gets the broadcast stream, and forwards to the media player running on the same machine. In addition, the participating hosts send performance statistics to the monitor and log server for both on-line and post-mortem analyses.

The detailed software architecture at the source and the receiver is depicted in Figure 2. Tracing the data flow, the broadcast source encodes the media signal into audio and multiple video packet streams (a), marks the packets with priority bits (b), and sends them to the overlay modules (shaded blocks). Multiple streams and prioritization are discussed in § 2.2. The overlay modules replicate packets to all of its children (c). Packets are translated from

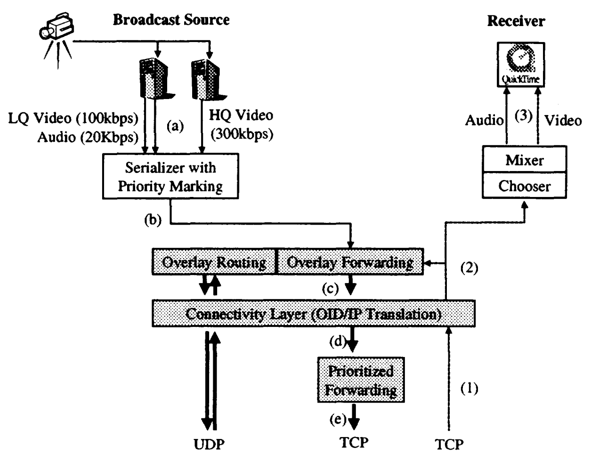

Figure 2: Block diagram of the software architecture for the broadcast source (left) and the receiver (right). Shaded blocks are shared by all hosts. Arrows indicate data flow.

Overlay ID (OID) to IP addresses (*d*), and forwarded to each child using prioritization semantics (*e*). Once a child receives packets, it translates IP addresses back to OIDs (1), selects the best video stream, adjusts the RTP/RTCP headers (2), and forwards to the media player (3). The use of OID is described in § 2.4. The child also sends each data packet to the overlay module which forwards the data to its descendants. The rest of this section describes each of these blocks in detail.

2.1 Overlay Protocol

We provide a sketch of the overlay protocol below as a basis for the rest of the discussion. Because our application is single-source, the protocol builds and maintains an overlay tree in a distributed fashion. The tree is optimized primarily for bandwidth, and secondarily for delay. Each node also maintains a degree bound of the maximum number of children to accept.

Group Management: New hosts join the broadcast by contacting the source and retrieving a random list of hosts that are currently in the group. It then selects one of these members as its parent using the parent selection algorithm. Each member maintains a partial list of members, including the hosts on the path from the source and a random set of members, which can help if all members on the path are saturated. To learn about members, we use a gossip protocol adapted from [32]. Each host A periodically (every 2 seconds) picks one member (say B) at random, and sends B a subset of group members (8 members) that A knows, along with the last timestamp it has heard for each member. When B receives a membership message, it updates its list of known members. Finally, members are deleted if its state has not been refreshed in a period (5 minutes).

Handling Group Membership Dynamics: Dealing with graceful member leave is fairly straight-forward: hosts continue forwarding data for a short period (5 seconds), while its children look for new parents using the parent selection method described below. This serves to minimize disruptions to the overlay. Hosts also send periodic control packets to their children to indicate live-ness.

Performance-Aware Adaptation: We consider three dynamic network metrics: available bandwidth, latency and loss. There are two main components to this adaptation process: (i) detecting poor performance from the current parent, or identifying that a host must switch parents, and (ii) choosing a new parent, which is discussed in the *parent selection* algorithm.

Each host maintains the application-level throughput it is receiving in a recent time window. If its performance is significantly below the source rate (less than 90% in our implementation), then it enters the probe phase to select a new parent. While our initial implementation did not consider loss rate as a metric, we found it necessary to deal with variable-bit-rate streams, as dips in the source rate would cause receivers to falsely assume a dip in performance and react unnecessarily. Thus, our solution avoids parent changes if no packet losses are observed despite the bandwidth performance being poor.

One of the parameters that we have found important is the *detection time* parameter, which indicates how long a host must stay with a poor performing parent before it switches to another parent. Our initial implementation employed a constant detection time of 5 seconds. However our experience reveals the need for the protocol to adaptively tune this timer because: (a) many hosts are not capable of receiving the full source rate, (b) even hosts that normally perform well may experience intermittent local network congestion, resulting in poor performance for any choice of parent, (c) there can be few good and available parent choices in the system. Changing parents under these environments may not be fruitful. We have implemented a simple heuristic for dynamically adjusting the detection time, involving an increase if several parent changes have been made recently, and a decrease if it has been a long time since the last parent change.

Parent Selection: When a host (say A) joins the broadcast, or needs to make a parent change, it probes a random subset of hosts it knows (30 in our implementation). The probing is biased toward members that have not been probed or have low delay. Each host B that responds to the probe provides information about: (i) the performance (application throughput in the recent 5 seconds, and delay) it is receiving; (ii) whether it is degree-saturated or not; and (iii) whether it is a descendant of A to prevent routing loops. The probe also enables A to determine the round-trip time to B. A waits for responses for 1 second, then eliminates those members that are saturated, or who are its descendant. It then evaluates the performance (throughput and delay) of the remaining hosts if it were to choose them as parents. If A does not have bandwidth estimates to potential parents, it picks one based on delay. Otherwise, it computes the expected application throughput as the minimum of the throughput B is currently seeing and the available bandwidth of the path between B and A. History of past performance is maintained so if A has previously chosen B as parent, then it has an estimate of the bandwidth of the overlay link $B - A$. A then evaluates how much improvement it could make if it were to choose B.

A switches to the parent B either if the estimated application throughput is high enough for A to receive a higher quality stream (see the multi-quality streaming discussion in § 2.3) or if B maintains the same bandwidth level as A's current parent, but improves delay. This heuristic attempts to reduce resource usage by making hosts move closer to one another.

Degree Bound Estimation: In order to assess the amount of upstream bandwidth resources each host can contribute to the overlay, we ask the user to choose whether or not it has at least a 10 Mbps up-link to the Internet. If so, we assign such hosts a degree bound of

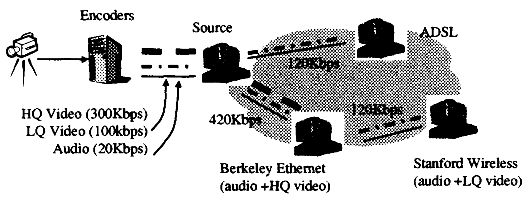

Figure 3: Single overlay approach to host heterogeneity.

6, to support up that many number of children. Otherwise, we assign a degree bound of 0 so that the host does not support any children. We have been experimenting with heuristics that can automatically detect the access bandwidth of the host, but this turns out not to be straightforward. We discuss this further in § 6.

2.2 Support for Receiver Heterogeneity

Internet hosts are highly heterogeneous in their receiving bandwidth, thus a *single-rate video coding* scheme is not the most appropriate. Various streaming systems have proposed using scalable coding techniques such layered coding or multiple description coding (MDC) in their design [37, 25, 5], however these technologies are not yet available in commercial media players. To strike a balance between the goals of rapid prototyping and heterogeneous receiver support, in our system, the source encodes the video at multiple bit-rates in parallel and broadcasts them simultaneously, along with the audio stream, through the overlay as shown in Figure 3. We run a unicast congestion control on the data path between every parent and child, and a *prioritized packet forwarding* scheme is used to exploit the available bandwidth. That is, audio is prioritized over video streams, and lower quality video is prioritized over higher quality video. The system dynamically selects the best video stream based on loss rate to display to the user. Thus, audio is highly protected. When a receiver does not have sufficient bandwidth to view the high quality video stream, or when there are transient dips in available bandwidth due to congestions or poor parent choices, as long as the lower quality video stream is received, a legible image can still be displayed. We note that while this design involves some overhead, it can be seamlessly integrated with layered codecs if available.

Much of the deployment experience reported in this paper uses TCP as the congestion control protocol. We implement priority forwarding by having parents in the overlay tree maintain a fixed size per-child priority buffer. Packets are sent in strict priority and in FIFO order within each priority class. If the priority buffer is full, packets are dropped in strict priority and in FIFO order (drop head). The priority buffer feeds the TCP socket, and we use *non-blocking write* for flow control. Note that once packets are queued in kernel TCP buffers, we can no longer control the prioritization. While we were aware of this limitation with using TCP, we were reluctant to employ untested UDP congestion control protocols in actual large scale deployment. Our subsequent experience has revealed that while the choice of TCP has only a minor hit on the performance of the prioritization heuristics, a more first-order issue is that it limits connectivity in the presence of NATs and firewalls. Faced with this, our recent broadcasts have begun using TFRC [13], a UDP-based congestion control protocol.

Child	Parent		
	Public	NAT	Firewall
UDP Transport			
Public	√	√	√
NAT	√	?⋆	?
Firewall	√	?	?⋆
TCP Transport			
Public	√	√	√
NAT	√	⋆	×
Firewall	√	×	⋆

Table 1: Connectivity Matrix. √ means connectivity is always possible. ? means connectivity is possible for some cases of NAT/firewall and ⋆ means connectivity is only possible if the hosts are in the same private network.

To prevent frequent quality switches that could annoy a user, we adopted a damping heuristic. Here, we aggressively switch to lower quality when high quality video has consistent loss for 10 seconds, and conservatively switch to higher quality when no loss is observed in the higher quality video stream for at least 50 seconds. Dynamically switching video qualities required us to implement an RTCP mixer[15]. When video qualities are switched, the mixer ensures the outgoing video stream to QuickTime is (i) masked as one contiguous stream; and (ii) time synchronized with the audio stream. One limitation in our current implementation is that if a host is displaying a low quality stream, the parent still forwards some data from the high quality stream. We are currently refining the implementation by adding heuristics to have the child unsubscribe from the higher quality stream, and periodically conduct experiments to see when network condition has improved so that it can start receiving the high quality stream.

2.3 Interface to Media Components

We use *QuickTime* [29] as the media player in our system because it is widely available and runs on multiple popular platforms. We use *Sorenson 3* [38] and MPEG4, both of which are supported by QuickTime, as video codecs. To support receiver heterogeneity, the source encodes the video at two target bit-rates (100 kbps and 300 kbps), and the audio at 20 kbps. We empirically determine the suitable encoding rates by experimenting with various encodings of conference talks. We find that a frame size of 640x480 is necessary to read the words on the slides. A minimal rate of 100 kbps yields watchable, 5 frames per second video motion. A rate of 300 kbps produces good video quality with 15 frames per second. To hide from the media player the fact that the overlay parent changes over time, we direct the player to a fixed *localhost:port* URL which points to the overlay proxy running at the same host. The overlay proxy handles all topology changes and sends data packets to the player as if it were a unicast broadcast server.

2.4 NATs and Firewalls

Our initial prototype did not include support for NATs and firewalls, and we were motivated to address this as we consistently needed to turn down over 20 – 30% of viewers in our early broadcasts for the lack of such support. NATs and firewalls impose fundamental restrictions on pair-wise connectivity of hosts on the overlay. In most cases, it is not possible for NATs and firewalls to communicate directly with one another. However, there are specific exceptions, depending on the transport protocol (UDP or TCP), and the exact behavior of the NAT/firewall. Adopting the classification from STUN [16], *Full Cone NATs* can receive incoming packets to a port from any arbitrary host once it sends a packet on that port to any destination. Many hosts can address a host behind a full cone NAT using the same port number. In contrast, *Symmetric NATs* allow incoming packets only from the host that it has previously sent a packet to. Different hosts address a host behind a symmetric NAT using different port numbers. Table 1 characterizes these restrictions for the different transport protocols, where columns represent parents and rows represent children. For example, communication is not possible between two NATed hosts using TCP unless they happen to be in the same private network. In addition, "?" denotes that communication is possible using UDP between two NATed hosts if one of them is behind a *Full Cone NAT*. The *firewalls* which we refer to in Table 1 allow UDP packets to traverse in either direction. The system does not support firewalls that block UDP.

The primary goals in supporting NATs and firewalls are: (i) enable connectivity, a generic problem shared by many applications wishing to support these hosts and (ii) address protocol-specific enhancements to become "NAT/firewall-aware" to improve efficiency and performance.

2.4.1 Enable Connectivity

Use Overlay Identifier for Unique Naming: In the overlay protocol, each host needs to have a distinct and unique identifier. The straightforward use of public and private IP address and port does not serve this purpose because of symmetric NATs. To resolve this, we assign a unique overlay identifier(OID) to each host and decouple it from its IP address, separating overlay naming from addressing. When a host A joins the group, it is assigned an OID by the source. The source creates a binding that maps the OID of A to its public and private addresses and ports. This binding is distributed as part of the group membership management protocol.

Learn, Maintain, and Translate Bindings: There are two ways for a host B to learn bindings for host A. First, it can learn the binding as part of the group membership operations. Second, it may receive packets directly from A. Bindings learned by the second method are prioritized because they are the only ones that can be used to talk to a host behind a symmetric NAT. Each host B maintains the OID and associated binding for every other member A that it knows. The OID is translated into the appropriate binding when B wishes to send a packet to A. In some cases A and B may be behind the same private network, but have different public IP addresses. This is common in the case of large corporations that use multiple NAT gateways. We use a simple heuristic to match the prefixes in the public IP address. This matching expires if B does not receive packets from A after a short while.

Set up TCP Parent-Child Connection for Data: We use bi-directional connection initiation, by which both parent and child attempt to open a connection to the other. If one is a public and the other is NAT/firewall, then only one of the connections will be successful.

If both are public, then both connections will be successful and we arbitrarily close the connection initiated by the host with higher IP address.

2.4.2 Making the Protocol Aware of NATs and Firewalls

The protocol works correctly with the connectivity service, without needing to make any changes. However, being aware of connectivity constraints can improve protocol efficiency and performance. We have identified 2 changes to the protocol to make it explicitly aware of connectivity constraints.
Group Management and Probing: To increase the efficiency of control messages, we enhance the group management protocol to explicitly avoid control messages between pairs of hosts that cannot communicate (e.g., NAT-NAT). Similarly, for probing, we do not allow NATs/firewalls to probe other NATs/firewalls.
Self-Organization: If the overlay protocol is aware of the NAT and firewall hosts in the system, it can support more of them by explicitly structuring the tree. For example, an efficient structure is one in which public hosts use NAT or firewall hosts as parents to the extent possible. In contrast, a structure in which a public host is a parent of another public host is inefficient because it reduces the potential parent resources for NAT hosts. However, it was not clear whether the increased complexity of such mechanisms would lead to significant benefit, so we did not optimize for it. We discuss this further in § 6.

3 Deployment Status

3.1 System Status

To make the broadcast system easily and widely accessible, and attract as many participants as possible, we have taken effort to support multiple OS (Linux, Windows, MAC) and player platforms (QuickTime, Real Player) and develop user-friendly interfaces for both publishers and viewers. With the subscriber Web interface, any receiver can tune in to a broadcast by a single click on a web-link.

The broadcast system is also designed for ease of deployment. We learned from our first broadcast event that having 5 graduate students spend 2 days to manually set up a broadcast was a barrier for deployment. Our publishing toolkit [12] has evolved since then into a user-friendly web based portal for broadcasting and viewing content. This portal allows content publishers to setup machines, machine profiles (such as which machines should be the source, log servers, and encoders), and events. With this information configured, the broadcast can be launched directly from the web. With no prior experience using the system and minimal support from us, most content publishers spend a couple hours to set up and run a broadcast. A monitoring system has been built to provide content publishers with online information about individual participating hosts, the current overlay tree, the bandwidth on each overlay link, and the current group membership. In addition, the system can recover from simple failures such as automatically re-starting the log server when it crashes.

As a research vehicle, the broadcast system has a built-in logging infrastructure that enables us to collect performance logs from all hosts participating in the broadcast for post-

Event	Duration (hours)	Unique Hosts/ Waypoints	Peak Size/ Waypoints
SIGCOMM 2002	25	338/16	83/16
SIGCOMM 2003	72	705/61	101/61
DISC 2003	16	30/10	20/10
SOSP 2003	24	401/10	56/10
Slashdot	24	1609/29	160/19
Distinguished Lectures Series (8 distinct events)	9	358/139	80/59
Sporting Event	24	85/22	44/22
Commencement (3 distinct events)	5	21/3	8/3
Special Interest	14	43/3	14/3
Meeting	5	15/2	10/2

Table 2: Summary of major broadcasts using the system. The first 4 events are names of technical conferences.

mortem analysis. The logs are sent on-line to a log server during the session. The data rate is bounded at 20 kbps to avoid interfering with the overlay traffic.

3.2 Deployment Experience

Over the last year, the system has been used by 4 content publishers and ourselves to broadcast more than 20 real events, the majority of which are conferences and lectures, accumulating 220 operational hours. In all, the system has been used by over 3600 participants. We summarize some of our key experience with regard to how successful we were in attracting publishers and viewers to use the system, the extent of our deployment, and some of the factors that affected our deployment.

Attracting content publishers: One of the key challenges we face is finding content. It has been difficult to access popular content such as movies and entertainment, as they are not freely available and often have copyright limitations. However, we have been more successful at attracting owners of technical content, such as conferences, workshops and lectures. Typically event organizers have expressed considerable interest in the use of our system. However given the wariness toward adopting new technology, convincing an event organizer to use the system involves significant time and ground-work. The key element of our success has been finding enthusiastic champions among conference organizers who could convince their more skeptical colleagues that it is worth their while to try the new technology even when they are already overwhelmed by all the other tasks that organizing a conference involves. We have also learned that the video production process is important, both in terms of cutting costs given that conferences operate with low-budgets, and in terms of dealing with poor Internet connectivity from the conference sites to the outside world.

Viewer Participation: Table 2 lists the major broadcasts, duration, number of unique participants, and the peak group size. The broadcast events attracted from 15 to 1600 unique participants throughout the duration and peaked at about 10 to 160 simultaneous participants. Most of the audience tuned in because they were interested in the content, but could not attend the events in person. The Slashdot broadcast is different in that wanting to explore a larger scale and wider audience, we asked readers of Slashdot [36], a Web-based discussion forum, to experiment with our system. While some of the audience tuned in for

	SIGCOMM 2002 broadcast 8/2002 9am-5pm (total 141 hosts)				
Region	North America (101)	Europe (20)	Oceania (1)	Asia (12)	Unknown (7)
Background	Home (26)	University (87)	Industry (5)	Government (9)	Unknown (14)
Connectivity	Cable Modem (12)	10+ Mbps (91)	DSL (14)	T1 (2)	Unknown (22)
	Slashdot broadcast 12/2002 2pm-10:30pm (total 1316 hosts)				
Region	North America (967)	Europe (185)	Oceania (48)	Asia (8)	Unknown (108)
Background	Home (825)	University (127)	Industry (85)	Government (80)	Unknown (199)
Connectivity	Cable Modem (490)	10+ Mbps (258)	DSL (389)	T1 (46)	Unknown (133)
NAT	NAT (908)	Public (316)	Firewall (92)		

Table 3: Host distributions for two broadcast events, excluding waypoints, shown only for a portion of the broadcast.

the content, others tuned in because they were curious about the system.

While our deployment has been successful at attracting thousands of users, the peak group sizes in our broadcasts have been relatively low with the largest broadcast having a peak size of about 160. One possible explanation for this is that the technical content in these broadcasts fundamentally does not draw large peak group sizes. Another possibility is that users do not have sufficient interest in tuning in to live events, and prefer to view video archives. Our ongoing efforts to draw larger audience sizes include contacting non-technical organizations, and incorporating interactive features such as questions from the audience to the speaker.

We wish to emphasize that our limited operational experience with larger group sizes has been constrained by the lack of appropriate content, rather than due to specific known limitations of our system. We have had encouraging results evaluating our system in Emulab [42] using 1020 virtual nodes, multiplexed over 68 physical nodes, as well as simulation environments with over 10,000 nodes. Our hope is to use the workloads and traces of environment dynamics, resources and diversity from our broadcasts to design more realistic simulations and emulations in the future.

Diversity of Deployment: The diversity of hosts that took part in two of the large broadcasts (SIGCOMM 2002 and Slashdot), excluding waypoints, can be seen from Table 3. The deployment has reached a wide portion of the Internet - users across multiple continents, in home, academic and commercial environments, and behind various access technologies. We believe this demonstrates some of the enormous deployment potential of overlay multicast architectures - in contrast, the usage of the MBone [4] was primarily restricted to researchers in academic institutions.

Decoupling development version from deployment version: One of the challenges associated with operational deployment is the need for robust, well-tested and stable code. Bugs can not only affect the performance of a broadcast, but can also significantly lower our credibility with event organizers championing our cause. This requires us to adopt extensive testing procedures using Emulab [42], Planetlab [28], and Dummynet [33] before code is marked ready for deployment. Further, in actual deployment, we typically use an older version of our system (several months) compared to our development version. One consequence of this is that even though certain design enhancements may seem trivial to incorporate, it may take several months before being used in actual broadcasts.

Use of Waypoints: Right from the early stages of our work on Overlay Multicast, we have been debating the architectural model for deploying Overlay Multicast. On the one

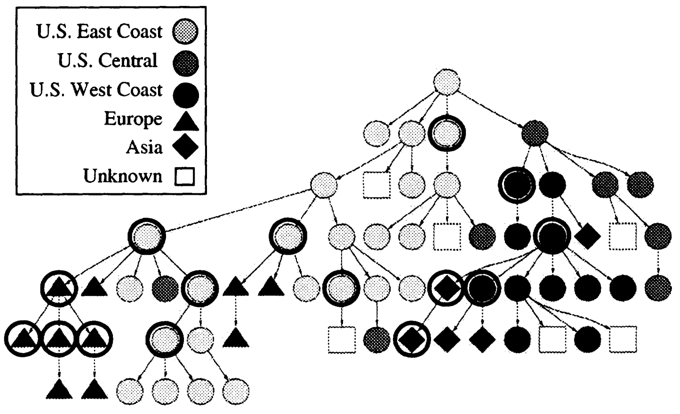

Figure 4: Snapshot of the overlay tree during Conference 1. Participants, marked by geographical regions, were fairly clustered. Waypoints, marked by outer circles, took on many positions throughout the tree.

hand, we have been excited by the deployment potential of *purely application end-point architectures* that do not involve any infrastructure support and rely entirely on hosts taking part in the broadcast. On the other hand, we have been concerned about the feasibility of these architectures, given that they depend on the ability of participating hosts to support other children. When it came to actual deployment, we were not in a position to to risk the success of a real event (and consequently our credibility and the content provider's credibility) by betting on such an architecture. Thus, in addition to real participants, we employed PlanetLab [28] machines, which we call waypoints, to also join the broadcast (also listed in Table 2). From the perspective of the system, waypoints are the same as normal participating hosts and run the same protocol – the only purpose they served was increasing the amount of resources in the system. To see this, consider Figure 4, which plots a snapshot of the overlay during the *Conference* broadcast. The shape and color of each node represents the geographical location of the host as indicated by the legend. Nodes with a dark outer circle represent waypoints. There are two points to note. First, the tree achieves reasonable clustering, and nodes around the same geographical location are clustered together. Second, we see that waypoints are scattered around at interior nodes in the overlay, and may have used normal hosts as parents. Thus they behave like any other user, rather than statically provisioned infrastructure nodes. While our use of waypoints so far has prevented direct conclusions about purely application end-point architectures, we can arrive at important implications for these architectures leading to reduced use of waypoints in subsequent broadcasts, as we have done in § 6.

4 Analysis Methodology

We conduct off-line analysis on the performance logs collected from hosts participating in the broadcasts. Our evaluation and analysis focus on the following questions:
• How well does the system perform in terms of giving good performance to the user?
• What kind of environments do we see in practice? How does the environment affect system

performance? Are there quantitative indices we can use to capture environment information?
• Using trace-based simulations on the data, can we ask "what-if" questions and analyze design alternatives that could have led to better performance?

The data that we use for the analysis is obtained from performance logs collected from hosts participating in the broadcast. We have instrumented our system with measurement code that logs application throughput sampled at 1 second intervals, and application loss rate sampled at 5 second intervals. Note that the sample period is longer for loss rates because we found from experience that it is difficult to get robust loss measurements for shorter sampling periods.

We define an *entity* as a unique user identified by its $< publicIP, privateIP >$ pair. An entity may join the broadcast many times, perhaps to tune in to distinct portions of the broadcast, and have many *incarnations*. The following sections, report analysis on incarnations unless otherwise stated.

Some of the analysis requires logs to be time synchronized. During the broadcast, whenever a host sends a message to the source as part of normal protocol operations (for example, gossip or probe message), the difference in local offsets is calculated and printed as part of the log. In the offline analysis, the global time for an event is reconstructed by adding this offset. We have found that the inaccuracy of not considering clock skew is negligible.

In this section, we provide an overview of our analysis methodology. We present results from broadcasts in § 5. Finally, in § 6, we quantitatively analyze the performance benefits that may accrue from key design modifications motivated by our experience.

4.1 User Performance Metrics

We evaluate the performance that individual users observe by measuring their average and transient network-level performance. In addition, user-level feedback is also presented to provide a more complete picture of the user experience.
•**Average performance** is measured as the mean application-level throughput received at each incarnation. This provides a sense of the overall session performance.
•**Transient performance** is measured using the application-level losses that users experience. Using the sampled loss rate from the performance logs, we mark a sample as being a loss if its value is larger than 5% for each media stream, which in our experience is noticeable to human perception. We use three inter-related, but complementary metrics: (i) fraction of session for which the incarnation sees loss; (ii) mean interrupt duration; and (iii) interrupt frequency.

Fraction of session for which the incarnation sees loss is computed as follows. If an incarnation participates for 600 seconds, it would have about 120 loss samples. If 12 of those samples are marked as being a loss, then the incarnation sees loss for 10% of its session.

We define an interrupt to be a period of consecutive loss samples. Interrupt duration is computed as the amount of time that loss samples are consecutively marked as losses. The interrupt durations are then averaged across all interrupts that an incarnation experiences. Note that this metric is sensitive to the sampling period.

Interrupt frequency is computed as the number of distinct interrupts over the incarnation's session duration, and reflects the dynamicity of the environment. A distinct interrupt

is determined to be a consecutive period for which the loss samples are marked as a loss. This metric is biased by incarnations that have short session durations. For example, if an incarnation stays for 1 minute, and experiences 2 distinct 5-second interrupts, the interrupt frequency would be once every 30 seconds.
•**User Feedback** complements the network-level metrics described above. We encouraged users to fill in a feedback form and rate their satisfaction level for various quality metrics such as ease of setup, overall audio and video quality, frequency of stalls, and duration of stalls. The results are, however, subjective and should be considered in conjunction with the more objective network-level metrics.
•**Additional Metrics** to capture the quality of the overlay have also been analyzed. For example, we have looked at the efficiency of the overlay based on resource usage [9], and overlay stability based on the rate of parent changes. Due to space limitations, we do not present these results.

4.2 Environmental Factors

A self-organizing protocol needs to deal with events such as an ancestor leaving, or congestion on upstream overlay links by making parent changes. Two key factors that affect performance then are: (i) the dynamicity of the environment; and (ii) the quality of resources (parents) available in the environment. The more dynamic an environment, the more frequently a host is triggered to react; the poorer the resources, the longer it could potentially take to discover a good parent.

4.2.1 Dynamics

The two key aspects of dynamics are: (i) group dynamics; and (ii) dynamics in the network. We measure group dynamics using mean interarrival time and session duration. We note however that the membership dynamics and overlay performance may not follow a strict cause and effect relationship. For example, users that see poor performance may leave, thus creating more dynamics in the system.

Our measurements are not conducive to summarizing network dynamics in terms of frequency and duration because of several reasons. First, we have measurements only for the subset of overlay links chosen and used by the protocol for data transfer. Second, the measurements could be biased by the protocol's behavior. For example, the observation of congestion duration may be shorter than in reality because the protocol attempts to move away from congestion and stops sampling that path. Instead, we characterize network dynamics by looking at the causes and location as described in § 4.3.

4.2.2 Environment Resources

Two key factors capture the quality of resources in an environment: (i) outgoing bandwidth of hosts, which directly bounds the number of children hosts can take; and (ii) the presence of NATs and firewalls which places connectivity restrictions on parent-child relationships. In this section, we introduce a metric called the *Quality Index* to capture the outgoing bandwidth of hosts, and then extend it to consider NATs and firewalls.

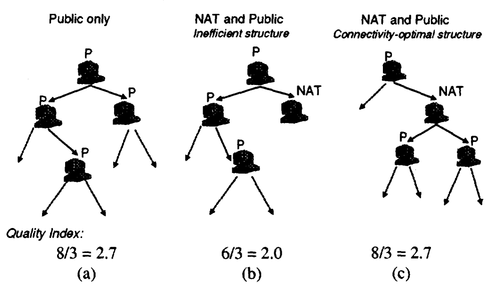

Figure 5: Example of quality index computation.

We define the *Quality Index* as the ratio of the number of receivers that the members in the group could *potentially sustain* to the number of receivers in the group for a particular source rate. By number of hosts that can be potentially sustained, we mean the sum of the existing hosts in the system and the number of free slots in the system. For example, consider Figure 5(a), where each host has enough outgoing bandwidth to sustain 2 children. The number of free slots is 5, and the *Quality Index* is $(5+3)/3 = 8/3$. Further, for a given set of hosts and out-going bandwidth, the *Quality Index* is the same for any overlay tree constructed using these hosts. A *Quality Index* of 1 indicates that the system is saturated, and a ratio less than 1 indicates that not all the participating hosts in the broadcast can receive the full source rate. As the *Quality Index* gets higher, the environment becomes less constrained and it becomes more feasible to construct a good overlay tree.

We have extended the definition of *Quality Index* to incorporate the connectivity constraints of NATs and firewalls, by only considering free slots available for NAT hosts. For example, in Figure 5(b), the number of slots available for NAT hosts is 3, and the *Quality Index* is 6/3. However, we note that the *Quality Index* not only depends on the set of hosts, but also becomes sensitive to the structure of the overlay for that set of hosts. Thus, while Figure 5(c) has the same set of hosts as Figure 5(b), we find the number of free slots for NATs is 5 and the *Quality Index* is 8/3.

We observe that the optimal structure in terms of accommodating NATs is one where public hosts preferentially choose NATs as parents. Based on this observation, the *optimal* Quality Index for a set of hosts involving NATs and firewalls is difined as S/N, where $S = S_{public} + Min(S_{nat}, N_{public})$. Here, S_{public} and S_{nat} are the maximum number of children that can be supported by the public and NAT hosts, N_{public} is the number of receivers that are public hosts and N is the total number of receivers. Figure 5(c) is an optimal structure for the set of hosts, and it can be verified that the formula confirms to the result stated above.

We wish to close with two practical issues that must be borne in mind with the *Quality Index* . First, it captures only the availability of resources in the environment, but does not account for factors such as performance of Internet paths. Also, the *Quality Index* is computed assuming global knowledge, but in practice, a distributed protocol may not be able to use the resources as optimally as it could have.

4.3 Loss Diagnosis

When evaluating a self-organizing protocol, we need to distinguish between losses that could possibly be fixed by appropriate self-organization techniques from the losses that are fundamental to the system (i.e. those caused by access link capacity limitations, trans-oceanic bottleneck link congestions and local congestions). Further, we are interested in identifying the location of losses in the overlay tree, and attribute causes to the loss. We now summarize steps in our loss diagnosis methodology below:

• *Identifying Root-Events:* If a host sees bad performance, then all of its descendants downstream see bad performance. Our first step filters out losses at descendants, and isolates a set of "root-events". If a host sees losses at a particular time, we determine whether its parent saw losses in a 5 second window around that time. This correlation relies on the time synchronization mechanism that we described earlier in the section.

• *Identifying Network Events:* Next, we classify the losses between the host and its parent based on cause. In our system, there are potentially two primary causes: (i) parent leave or death, and (ii) network problems (congestion or poor bandwidth) between the parent and child. There could be other miscellaneous causes such as host with slow processors and implementation bugs. Parent leave or death events are straightforward to identify from the logs. Hosts with slow processors are detected by abnormal gaps in time-stamps of operations that log messages at periodic intervals. Implementation bugs are revealed by abnormal patterns we detect during manual verification and analysis of logs. Thus, after a detailed elimination process and exhaustive manual verification, we classify the remaining losses that we are not able to attribute to any known cause as due to network problems.

• *Classifying constrained hosts:* Network losses can occur at several locations: (i) local to the child where a parent change is not needed; or (ii) local to the parent, or on the link between parent and child. As a first step, we identify hosts that see persistent losses near it. Hosts in this category include those that never see the full source rate throughout the session, or hosts that burst up to the full source rate for very short periods, but are not able to sustain the bandwidth for the entire duration. We identify these hosts using the following heuristic. If a host has seen losses for over 80% of the session, all of which are "root losses", and has tried at least 5 distinct parents during the session, then we decide the host is bandwidth constrained. Inherent here is the assumption that the protocol is doing a reasonable job in parent selection. This heuristic works well in environments with higher Quality Index. Finally, we manually verify these hosts and look for other evidence they are constrained (for example, location across a trans-oceanic link, names indicating they are behind wireless links etc.).

• *Classifying congestion losses:* The remaining losses correspond to hosts that usually see good performance but see transient periods of bad performance. If its siblings experience loss at around the same time, it is evidence that the loss is near the parent and not near a child; if a child has made several parent changes during an extended loss period, it is evidence that the loss is near the child. For the events that we are unable to classify, we label them as having "unknown location".

Event	Duration (hours)	Incarnations Excluding Waypoints	Mean Session Interarrival Time (sec)	Incarnation Session Duration (minutes)		Entity Session Duration (minutes)		% Eligible Parents	
				Mean	Median	Mean	Median	All	Public
SIGCOMM 2002	8	375	83	61	11	161	93	57%	57%
SIGCOMM 2003	9	102	334	29	2	71	16	46%	17%
Lecture 1	1	52	75	12	2	26	19	62%	33%
Lecture 2	2	72	120	31	13	50	53	44%	21%
Lecture 3	1	42	145	31	7	42	31	73%	43%
Slashdot	8	2178	17	18	3	11	7	19%	7%

Table 4: Summary of group membership dynamics and composition for the 6 larger broadcasts using the system.

5 Analysis Results

We present results from 6 of our larger broadcasts, 5 of which were conference/lecture-type broadcasts, and the other being *Slashdot*. For multi-day events, such as SIGCOMM 2002 and 2003, we analyzed logs from one day in the broadcast. For Slashdot, we present analysis results for the first 8 hours. In this section, we will present environment characterizations and performance results of the broadcasts. The analysis will indicate strong similarities in the environment for the conference/lecture-type broadcasts. However, they differ significantly from Slashdot. When we wish to illustrate a more detailed point, we use data from the *SIGCOMM 2002* and *Slashdot* broadcasts. The *SIGCOMM 2002* broadcast is one of the largest conference/lecture-type broadcasts, and is representative of these broadcasts in terms of application performance and resources.

5.1 Environment Dynamics

Table 4 lists the mean session interarrival time in seconds for the 6 broadcasts in the fourth column. For the five broadcasts of conferences and lectures, the mean interarrival time was a minute or more, whereas the interarrival time for Slashdot was just 17 seconds. Slashdot has the highest rate of group dynamics compared to all other broadcasts using our system. Note that the session interarrival times fit an exponential distribution.

Two different measures of session duration are listed in Table 4: individual incarnation duration and entity duration (cumulative over all incarnations) which captures the entity's entire attention span. For entity session duration, again, we find that all 5 real broadcasts of conferences and lectures have a mean of 26 minutes or more, and a median of 16 minutes or more. In the SIGCOMM 2002 broadcast, the median session duration was 1.5 hours which corresponds to one technical session in the conference. To contrast, the Slashdot audience has a very short attention span of 11 and 7 minutes for the mean and median. This indicates that the Slashdot audience may have been less interested in the content. The incarnation session duration also follows a similar trend with shorter durations. Note that SIGCOMM 2003 and Lecture 1 have very short median incarnation session durations. This is caused by 1 or 2 entities testing the system out, joining and leaving in less than a minute. Once we removed such entities, the median went up to 12 minutes or more, bringing it closer to the other 3 conferences and lectures.

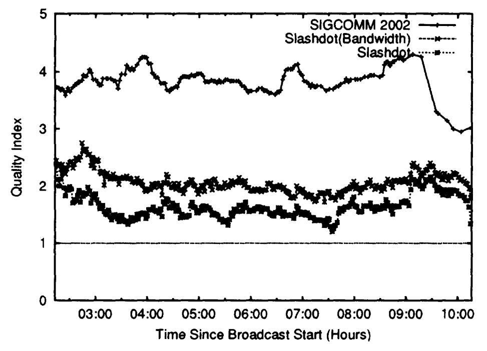

Figure 6: Quality Index as a function of time for (a) SIGCOMM 2002, (b) Slashdot with bandwidth constraint, (c) Slashdot with bandwidth and connectivity constraints.

5.2 Environment Resources

We look at the percentage of incarnations in the system that were eligible as parents, the last 2 columns in Table 4. The 5 conference and lecture broadcasts have the same trend, with 44% or more incarnations that can serve as parents. On the other hand, only 19% of incarnations could be parents in Slashdot. Further, when we consider the fraction of *public* hosts that could be parents, we find this ranges from 17 − 57% for the conference-style broadcasts, but is just 7% for the Slashdot broadcast. This indicates that there were much less available resources in the system in the Slashdot broadcast. Note that we did not have NAT/firewall support in the SIGCOMM 2002 broadcast.

Figure 6 depicts the quality index of the system as a function of time of the broadcast. The top and the lowest curves represent the *Quality Index* for the *SIGCOMM 2002* and *Slashdot* broadcasts, and are consistent with the definition in § 4.2.2. We note that the lowest curve corresponds to the actual overlay tree that was constructed during the broadcast. The middle curve, *Slashdot (Bandwidth)* considers a hypothetical scenario without connectivity constraints (that is, all NAT/firewall hosts are treated as public hosts). The SIGCOMM 2002 broadcast has a quality index of 4, potentially enough to support 4 times the number of members. In contrast, the *Slashdot (Bandwidth)* has a quality index of 2, and *Slashdot* has a quality index that is barely over 1. Thus, not only was the distribution of out-going bandwidth less favorable in the *Slashdot* broadcast, but also the presence of connectivity constraints made it a much harsher environment.

5.3 Performance Results

The previous analysis indicates that 5 of our broadcasts have similar resource distributions and dynamics patterns, but the Slashdot environment was more diverse and more dynamic. This section evaluates how the system performs.

Figure 7 plots the cumulative distribution of mean session bandwidth, normalized to the source rate for the 6 broadcasts. Five of the broadcasts are seeing good performance with more than 90% of hosts getting more than 90% of the full source rate in the SIGCOMM 2002, Lecture 2, and Lecture 3 broadcasts, and more than 80% of hosts getting more than 90% of the full source rate in the SIGCOMM 2003 and Lecture 1 broadcasts. In the Slashdot

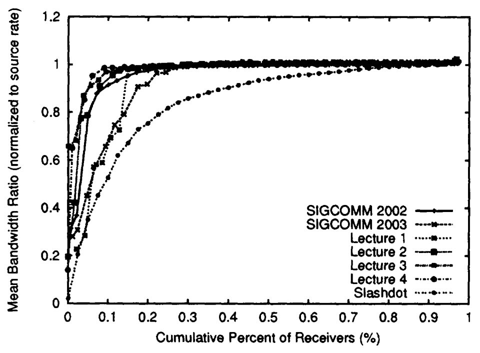

Figure 7: Cumulative distribution of mean session bandwidth (normalized to the source rate) for the 6 larger broadcasts.

	Setup ease	Audio Quality	Video Quality
SIGCOMM 2002	95%	92%	81%
Slashdot	96%	71%	66%

Table 5: Summary of user feedback for two broadcast events. Each number indicates the percentage of users who are satisfied in the given category.

broadcast, fewer hosts, 60%, are getting the same performance of 90% of the full source rate.

To better understand the transient performance, and performance of different stream qualities, we zoom in on the *SIGCOMM 2002* , which we will refer to as *Conference* , and *Slashdot* broadcasts. Figure 8 depicts the cumulative distribution of the fraction of time all incarnations saw more than 5% packet losses in all three streams in Slashdot and the Conference broadcast, for incarnations that stay for at least 1 minute. For the Conference broadcast, the performance is good. Over 60% of the hosts see no loss in audio and low quality video, and over 40% of the hosts see no loss in high quality video. Further, over 90% of the hosts see loss for less than 5% of the session in the audio and low quality streams, and over 80% of the hosts see loss for less than 5% of the session in the high quality stream. We will further analyze the performance of the hosts that are seeing the worst performance in § 5.4 and demonstrate that these are mostly hosts that are fundamentally constrained by their access bandwidth. For the Slashdot broadcast on the other hand, the low quality video and audio streams see reasonable performance, but the performance of the high quality stream is much less satisfactory. Over 70% of the users see loss for less than 10% of the session in low quality video, but only 50% of users see loss for less than 10% of the session for high quality video. Note that the audio and low quality streams are seeing better performance than the high quality because of the use of the priority buffer described in § 2.2. For sessions with a high loss rate of high quality video, the low quality one was actually displayed to the user.

Figure 9 depicts the cumulative distribution of the duration of interrupts seen by each incarnation. We find that the interrupt duration is almost identical for 5 curves: all 3 streams in Conference, and low quality video and audio in Slashdot. However, the high quality video in Slashdot sees a pronounced higher interrupt duration. More than 70% of hosts see a mean interrupt duration of less than 10 seconds, and 90% of hosts see a mean interrupt duration of

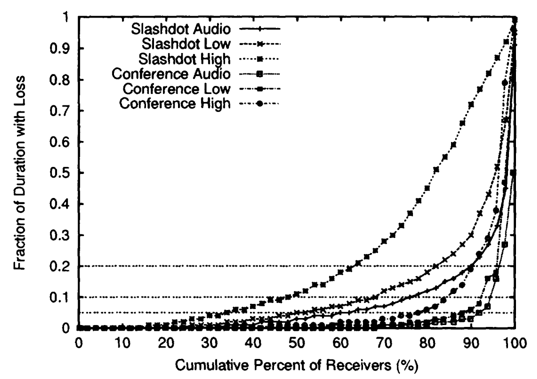

Figure 8: Cumulative distribution of fraction of session time with more than 5% packet loss of hosts in the two broadcasts.

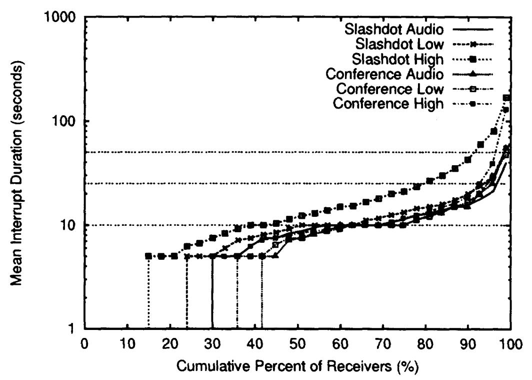

Figure 9: Cumulative distribution of mean interrupt duration.

less than 25 seconds for all 5 streams. However, 90% of hosts see a mean interrupt duration of less than 50 seconds for the Slashdot high quality stream.

We have also analyzed the cumulative distribution of the frequency of interrupts seen by each incarnation. We find that the interrupt frequency is higher for Slashdot, probably reflecting the more dynamic environment. For example, in the Conference broadcast over 80% of hosts see an interrupt less frequent than once in five minutes and 90% see an interrupt less frequent than once in two minutes. In Slashdot, 60% of hosts see an interrupt less frequent than once in five minutes and 80% see an interrupt less frequent than once in two minutes.

User Feedback: Table 5 summarizes statistics from a feedback form users were encouraged to fill when they left the broadcast. Approximately 18% of users responded and provided feedback. Most users were satisfied with the overall performance of the system, and more satisfied with the overall performance in the Conference broadcast, which is consistent with the network level metrics in Figures 7 and 8.

5.4 Loss Diagnosis

Figures 8 and 9 show that for the *Conference* broadcast, while most users saw good performance, there is a tail which indicates poor performance. To better understand the tail, we analyze the data using the loss diagnosis methodology presented in § 4.3. Figure 10 shows the breakdown of all loss samples across all hosts. We find that almost 51% of losses are not fixable by self-organization. 49% corresponded to hosts that were bandwidth constrained, while 2% of losses belonged to hosts that were normally good, but experienced network problems close to them for a prolonged period. 6% of losses corresponded to network events that were fixable by adaptation, while 18% of losses corresponded to network events that we were not able to classify. Manual cross-verification of the tail revealed about 30 incarnations that were marked as constrained hosts. This corresponded to about 17 distinct entities. Of these, 5 are in Asia, 1 in Europe, 3 behind wireless links, 1 behind a LAN that was known to have congestion issues, and 7 behind DSL links.

Finally, Figure 10 indicates that dynamics in the network is responsible for significantly

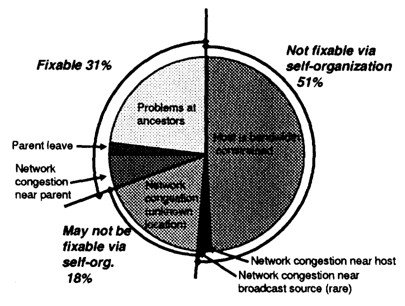

Figure 10: Loss diagnosis for Conference.

more losses than group dynamics. In some cases, even well-provisioned paths see prolonged periods of congestion. As an anecdotal example, we observed that a gigabit link between a U.S. academic institution and the high-speed Internet2 backbone that typically provides good consistent performance, had a congestion epoch that lasted up to 3 minutes. Both observations are consistent with other broadcasts including Slashdot.

6 Lessons Learned

Our experience over the last year, substantiated with data and analysis, has pointed us toward four key design lessons that are guiding future refinements of our system.

Our first lesson sheds light on the potential of *purely application end-point based* overlay multicast architectures that rely entirely on the hosts taking part in the broadcast. As discussed in § 3.2, our deployment used waypoints, additional hosts that help increase the resources in the system but were otherwise no different than normal clients. We analyze how important the resources provided by waypoints was to the success of our broadcasts.

Our next three lessons deal with techniques that can enable good performance in environments with low Quality Index, even in the absence of waypoints. The analysis for these lessons assume that the resources provided by waypoints is unavailable, and consequently a purely application end-point architecture.

Lesson 1: There is opportunity to reduce the dependence on waypoints and use them in an on-demand fashion.

In order to understand whether or not waypoints are necessary to the success of a broadcast, we look at Figure 11 which plots the *Quality Index* in the Conference and Slashdot broadcasts, with and without waypoints. The Conference broadcast had enough capacity to sustain all hosts even without waypoint support. Furthermore, most of the broadcasts, similar to the Conference broadcast, are sustainable using a purely application end-point architecture. In one of the lecture broadcasts, all the waypoint left simultaneously in the middle of the broadcast due to a configuration problem, and we found that the system was able to operate well without the waypoints.

On the other hand, we find that the connectivity constraints in the Slashdot broadcast resulted in a low *Quality Index* that occasionally dipped below 1 in Figure 11. This indicates

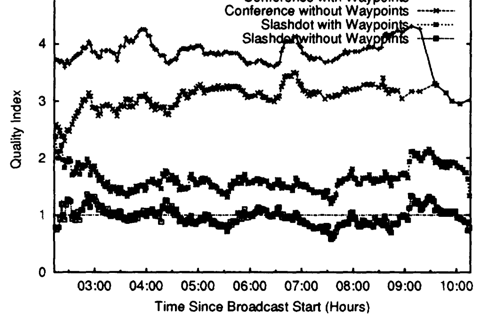

Figure 11: Quality Index as a function of time with and without waypoint support.

that it was not feasible to construct an overlay among all participating hosts that could sustain the source rate. Dealing with such environments can take on two complementary approaches (i) design techniques that can enable good performance in purely application end-point architecture, even in the absence of waypoints (which forms the thrust of the subsequent lessons in this section), or (ii) use a waypoint architecture, with the insight that waypoints may not be needed for the entire duration of the broadcast, and can be invoked on-demand. For ease of deployment, our objective is to explore both approaches and gradually decrease the dependence on waypoints, using them as a back-up mechanism, only when needed.

We note that in the long-term, waypoint architectures may constitute an interesting research area in their own right, being intermediate forms between pure application end-point architectures and statically provisioned infrastructure-centric solutions. The key aspect that distinguishes waypoints from statically provisioned nodes is that the system does not depend on these hosts, but leverages them to improve performance.

Lesson 2: Exploiting heterogeneity in node capabilities through differential treatment is critical to improve the performance of the system in environments with low Quality Index. Further, there is considerable benefit to coupling such mechanisms with application-specific knowledge.

If the Quality Index dips below 1, the system must reject some hosts or degrade application quality. In this section, we evaluate performance in terms of the fraction of hosts that are rejected, or see lower application quality. We consider three policies. In the *First-Come-First-Served (FCFS)* policy that is currently used in our system, any host that is looking for a new parent, but finds no unsaturated parent is rejected. In the *Contributor-Aware* policy, the system distinguishes between two categories of hosts: contributors (hosts that can support children), and free-riders (hosts that cannot support children). A contributor C that is looking for a new parent may preempt a free-rider (say F). C can either accommodate F as a child, or kick it out of the system if C is itself saturated. This policy is motivated by the observation that preferentially retaining contributors over free-riders can help increase overall system resources. Finally, we consider *Rate-Adaptation* where a parent reduces the video rate to existing free-riders in order to accommodate more free-riders. For example, a

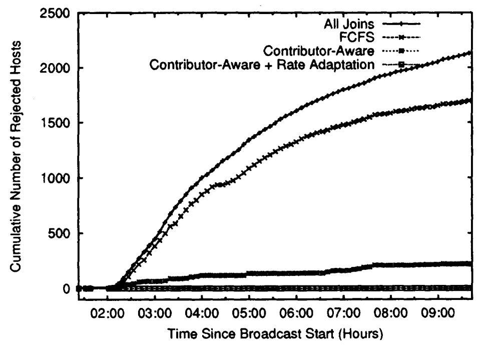

Figure 12: Number of rejected hosts under three different protocol scenarios in the simulated Slashdot environment.

parent can stop sending the high quality video (300 kbps) to one child, and in return, support three additional 100 kbps children. This policy is an example that not only differentially treats hosts based on their capabilities, but also exploits application knowledge.

We evaluate the potential of these policies by conducting a trace-based simulation using the group membership dynamics pattern from the Slashdot broadcast. We retain the same constitution of contributors and free-riders, but remove the waypoints from the group. We simulate a single-tree protocol where each receiver greedily selects an unsaturated parent, and we assume global knowledge in parent selection. If there is no unsaturated parent in the system, then we take action corresponding to the policies described above. Figure 12 shows the performance of the policies. We see that throughout the event, 78% of hosts are rejected using the *FCFS* policy. *Contributor-Aware* policy can drastically reduce the number of rejections to 11%. However, some free-riders are rejected because there are times when the system is saturated. With the *Rate Adaptation* policy however, no free-rider is rejected. Instead, 28% of the hosts get degraded video quality for some portion of the session.

Our results demonstrate the theoretical potential of contributor-aware rejection and rate adaptation. A practical design has to deal with many issues, for example, robust ways of automatically identifying contributors (see next lesson), techniques to discover the saturation level of the system in a distributed fashion, and the trade-offs in terms of larger number of structure changes that preemption could incur. We are currently in the process of incorporating these policies in our design and evaluating their actual performance.

Lesson 3: It is imperative to design techniques that can automatically infer the capabilities of nodes. In particular, techniques are needed for inferring the outgoing access bandwidth of nodes

As the previous lesson indicates, it is important to design protocol techniques that differentially treat nodes based on their contributions. An issue then is determining the contribution level of a node to the system, and in particular, determinining the outgoing access bandwidth of a node. In our current system, the user is asked if his access bandwidth has a 10Mbps up-link to the Internet to help determine whether the host should have children (§ 2.1). This approach is susceptible to free-loaders[35], where a user declares that he has

	10+Mbps	Below 10Mbps	Total
User truthful	11.1%	60.8%	71.9%
User lied	5.4%	4.9%	10.3%
User inconsistent	4.3%	13.5%	17.8%
Total	20.8%	79.2%	100.0%

Table 6: Accuracy in determining access bandwidth based on user input in Slashdot.

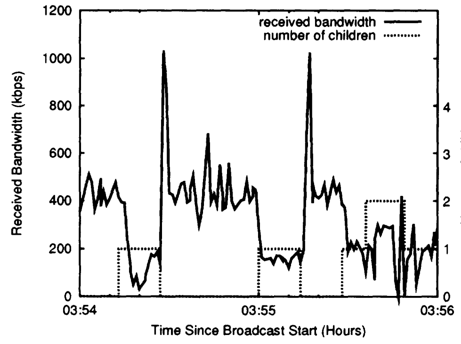

Figure 13: An example of a misconfigured DSL host taking children, causing poor performance to itself and its children.

less resources than he really does. However, an equally damaging problem in the context of Overlay Multicast is when a user declares he has more resources than he does. To see this, consider Figure 13 which depicts the performance of a DSL host that lied about having a 10Mbps up-link to the Internet, during the *Slashdot* broadcast. Whenever the host accepts a child, it affects not only the child's performance, but also its own performance. Further, a similar problem arises when a host can support less children (e.g. 4) than it claimed (e.g. 6). In a future design that prioritizes hosts that contribute more (Lesson 2), these effects can get further exacerbated.

To appreciate how reliable users were in selecting the correct access bandwidth in the Slashdot broadcast, consider Table 6. Each column represents a true access bandwidth, and each row represents a particular type of user behavior. "User Inconsistent" refers to users that had joined the group multiple times during the broadcast, and had selected both 10+Mbps option and lower than 10 Mbps option between consecutive joins, perhaps trying to figure out whether the choice yielded any difference in video quality. We determined the real access bandwidth using an off-line log analysis involving the following techniques: (i) DNS name, (ii) the TCP bandwidth of the upload log, (iii) online bottleneck bandwidth measurement, and (iv) Nettimer [20] from our university to target hosts. Since no single methodology is 100% accurate, we correlate results from all these techniques. We omit the details for lack of space.

From the table, we see that while 20.8% of hosts were behind $10Mbps$ links, only about half of them (11.1% of total) were truthful. Our trace-based simulation on the Slashdot log indicates that on average, this results in a 20% increase in *Quality Index* . Further, we find that while 79.2% of the users were behind links lower than $10Mbps$, about 4.9% chose the higher option or were being inconsistent (13.5%) about their connectivity.

We have been experimenting with techniques to automatically detect the outgoing access bandwidth of hosts. While access bandwidth measurement has been well studied in

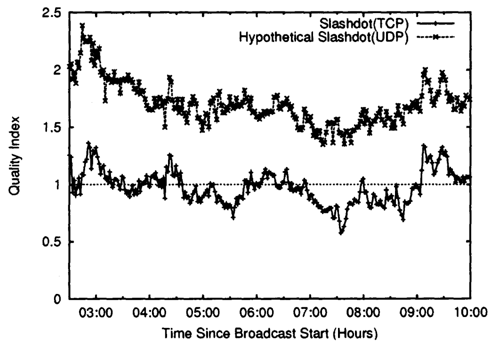

Figure 14: Quality Index comparison of two connectivity solutions for NAT/firewall: (i) Slashdot (TCP), (ii) Hypothetical Slashdot (UDP).

the literature [17, 11, 20], many of them are not applicable because the measurement code must run at user-level and with coarse application-level time-stamps. We have experimented using *traceroute* to find the local network topology, and *ping* of different sizes to estimate bottleneck bandwidth. We have also been experimenting with techniques that passively monitor the performance of parents and automatically learn their access bandwidth. These techniques show promise and we hope to deploy them in the future.

Lesson 4: Addressing the connectivity constraints posed by NATs and Firewalls may require using explicit NAT/firewall-aware heuristics in the protocol.

In light of our experience, NATs and firewalls can constitute an overwhelming fraction of a broadcast (for example, 50%-70% in *Slashdot*), and thus significantly lower the *Quality Index*. Clearly, using UDP as the transport protocol could improve the situation by increasing the amount of pair-wise connectivity, particularly connectivity between Full-Cone NATs. However, a less obvious improvement, which we briefly presented in § 2.4 is to make the self-organizing protocol explicitly aware of NAT/firewalls. In particular public hosts should preferentially choose NATs as parents, leaving more resources available for NATs/firewalls.

We now evaluate the potential of these two design improvements to help determine whether or not the additional complexity is worth the performance gains. Figure 14 shows the Quality Index for the system for the various design alternatives as a function of time, again omitting waypoint hosts. The lowest curve corresponds to the optimal quality index that can be achieved with a TCP-based protocol. The topmost curve corresponds to the optimal quality index with UDP and a NAT/firewall-aware self-organizing protocol. We see a significant increase of 74%. The combination of the two techniques above can significantly improve the *Quality Index*. Both techniques are being implemented in the latest version of our system and will soon be used for upcoming broadcasts.

7 Related Work

In this section, we discuss how our work relates to (i) other existing Internet broadcast systems and (ii) work in the Overlay Multicast community.

Broadcast Systems: The MBone [4] Project, and its associated applications such as vic [24], vat [18], and MASH [23] made a great effort to achieve ubiquitous Internet broadcasting However, the MBone could only touch a small fraction of Internet users (mostly networking researchers) due to the fundamental limitations of IP Multicast and dependence on the special MBone infrastructure. In contrast, our system has over a short time already reached a wide range of users, including home users behind a range of access technologies, and users behind NATs and firewalls.

Commercial entities, such as Akamai [2] and Real Broadcast Network [31], already provide Internet broadcasting as a charged service. They rely on dedicated, well-provision infrastructure nodes to replicate video streams. Such an approach has some fundamental advantages such as security and stable performance. However, these systems are viable only for larger-scale publishers, rather than the wide-range of low budget Internet broadcasting applications we seek to enable.

Recently, several peer-to-peer broadcast systems have been built by commercial entities [3, 6, 40] and non-profit organizations[26]. To our knowledge, many of these systems focus on audio applications which have lower bandwidth requirements. However, given the limited information on these systems, we are unable to do a detailed comparison.

Overlay Multicast: Since overlay multicast was first proposed four years ago many efforts [14, 9, 19, 7, 21, 30, 39, 22, 34, 25, 41, 10, 5] have advanced our knowledge on protocol construction by improving performance and scalability. Most this work has been *protocol-centric* , and has primarily involved evaluation in simulation, and Internet testbeds such as PlanetLab. In contrast, this paper adopts an *application-centric* approach, which leverages experience from actual deployment to guide the research. We address a wide range of issues such as support for heterogeneous receivers, and NATs and firewalls, which are not typically considered in protocol design studies. To our knowledge this paper is among the first reports on experience with a real application deployment based on overlay multicast involving real users watching live content. We believe our efforts complements ongoing research in overlay multicast, by validation through real deployment, and providing unique data, traces and insight that can guide future research.

The overlay protocol that we use is distributed, self-organizing and performance-aware. We use a distributed protocol, as opposed to a centralized protocol [27, 25], to minimize the overhead at the source. The self-organizing protocol constructs an overlay tree amongst participating hosts in a tree-first manner, similar to other protocols [19, 41, 14], motivated by the needs of single source applications. In contrast there are protocols that construct a richer mesh structure first and then construct a tree on top [9, 7], or construct DHT-based meshes using logical IDs and employ a routing algorithm to construct a tree in the second phase [22]. Such protocols are typically designed for multi-source or multi-group applications.

In our protocol, members maintain information about hosts that may be uncorrelated to the tree, in addition to path information, while in protocols like Overcast [19] and NICE [34], group membership state is tightly coupled to the existing tree structure: While Yoid [14] and Scribe [22] also maintain such information, the mechanisms they adopt are different. Our system uses a gossip protocol adapted from [32], while Yoid builds a separate random control structure called the mesh, and Scribe constructs a topology based on logical identifiers.

Overcast [19] and Narada [9] discuss adaptation to dynamic network metrics such as bandwidth. Our experience indicates that a practical deployment must consider several details such as dynamic tuning of network detection time to the resources available in the environment, consider hosts that cannot sustain the source rate, and consider VBR streams, and indicate the need for further research and understanding in this area.

Recent work such as CoopNet [25], and Splitstream [5] has demonstrated significant benefits by tightly coupling codec-specific knowledge and overlay design. In these works, the source uses a custom codec to encode the multimedia stream into many sub-streams using multiple description coding, and constructs an overlay tree to distribute each sub-stream. This approach not only increases overall resiliency of the system, but also enables support for heterogeneous hosts by having each receiver subscribe to as many layers as its capacity allows. While we believe this a great direction for future research, our design has been influenced by practical system constraints on an immediately deployable operational system, and our desire to interoperate with commercial media players and a wide range of popular codecs. We hope to leverage ideas from this approach as the research attains greater maturity, and when custom codecs become available.

NATs and Firewalls: Several efforts such as UPnP [1] and STUN [16] focus their efforts in enabling connectivity of NATs and firewalls. Our focus in this paper has been on the interplay between the application and NAT/firewall support. In particular, we have examined how the connectivity constraints imposed by NATs and firewalls can impact overlay performance, and on issues related to the integration of protocol design with NATs and firewalls. While Yoid [14] supports NATs and firewalls, it supports such hosts as children only, whereas we try to use NATs as parents when possible. We believe this is one of the first reports on experience with an overlay multicast system in the presence of NATs and firewalls.

8 Summary and Future Work

In this paper, we have reported on our operational experience with a broadcast system based on Overlay Multicast. To our knowledge this is among the first reports on experience with real application deployment based on Overlay Multicast, involving real users. Our experience has included several positives, and taught us important lessons both from an operational deployment stand-point, and from a design stand-point.

Our system is satisfying the needs of real content publishers and viewers, and demonstrating the potential of Overlay Multicast as a cost-effective alternative for enabling Internet broadcast. The system is easy to use for both publishers and viewers. We have successfully attracted over 3600 users from diverse Internet locations to use our system. However, we have had limited success in attracting larger scales of participation, primarily because of the difficulty in getting access to non-technical content. Our experience with several conference/lecture-type broadcasts indicate that our system provides good performance to users. In such environments, we consistently observe that over $80 - 90\%$ of the hosts see loss for less than 5% of their sessions. Further, hosts that perform poorly are typically bandwidth constrained hosts. Even in a more extreme environment with a low *Quality Index*, users see good performance in audio and low quality video.

Getting the system deployed has frequently required finding an enthusiastic champion of the technology to convince their colleagues to use it. This has raised the stakes to ensure the success of a broadcast, which could in turn trigger further interest in the use of the system. Consequently, we have needed to use stable and well-tested code in our deployment, rather than code that implements the latest performance enhancements. Another consequence has been our use of waypoints, additional hosts that help increase the resources in the system, but were otherwise no different than normal clients. The use of waypoints has been motivated by the need to balance between conflicting goals - on the one hand we want to understand the resource availability in purely application end-point architectures; on the other hand we need to have a series of successful broadcasts in the first place before such knowledge can be obtained.

Our subsequent analysis has investigated the potential of *purely application end-point architectures*, that do not rely on the use of waypoints. Our analysis both show the promise for such architectures, but also the need to incorporate additional key design elements. For most of our broadcasts, there is sufficient bandwidth resources to enable a solution purely within the application end-point framework. In broadcasts with lower Quality Index, techniques that exploit the heterogeneity in node capabilities through differential treatment and application-specific knowledge bear significant promise. Our broadcasts have also forced us to better appreciate the connectivity constraints posed by NATs and firewalls, and have led us to investigate explicit NAT/firewall-aware heuristics in the protocol. While our lessons have been derived in the context of our system, we believe they are of broader applicability to the community as a whole.

With the experience accumulated over the last year, we have set several milestones for the next 1 year horizon. Our milestones include:
• At a design level, we hope to incorporate some of the design refinements described above which can enable better performance in purely application end-point architectures. Our hope is to gradually minimize dependence on waypoints, through the use of on-demand waypoint invocation mechanisms.
• At an operational level, we hope to pursue wider and larger-scale deployment by attracting more publishers of both technical and non-technical content to the system, and convincing them to conduct their own broadcasts, incorporating interactivity features that might attract larger scales in synchronous applications, and encouraging other groups to run the broadcasts. Finally, while we have been conducting studies on the scalability of the system using emulations and simulations, we hope to gain deployment experience with larger peak group sizes.

Acknowledgements

We thank James Crugnale, Brian Goodman, Tian Lin, Nancy Miller, Jiin Joo Ong, Chris Palow, Vishal Soni, and Philip Yam for the help with the implementation and experimentation of the broadcast system. We also thank several event organizers who are early adopters of our system, and broadcast their events over the Internet.

References

[1] Understanding Universal Plug and Play. Microsoft White Paper.

[2] Akamai. http://www.akamai.com/.

[3] Allcast. http://www.allcast.com/.

[4] S. Casner and S. Deering. First IETF Internet audiocast. *ACM Computer Communication Review*, pages 92–97, 1992.

[5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Content Distribution in Cooperative Environments. In *Proceedings of SOSP*, 2003.

[6] Chaincast. http://www.chaincast.com/.

[7] Y. Chawathe. Scattercast: An architecture for Internet broadcast distribution as an infrastructure service. Fall 2000. Ph.D. thesis, U.C. Berkeley.

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

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

[10] J. Albrecht D. Kostic, A. Rodriguez and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of SOSP*, 2003.

[11] A. B. Downey. Using pathchar to estimate internet link characteristics. In *Measurement and Modeling of Computer Systems*, pages 222–223, 1999.

[12] End system multicast toolkit and portal. http://esm.cs.cmu.edu/.

[13] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of the ACM SIGCOMM*, August 2000.

[14] P. Francis. Yoid: Your Own Internet Distribution, http://www.aciri.org/yoid/. April 2000.

[15] R. Frederick H. Schulzrinne, S. Casner and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC-1889, January 1996.

[16] C. Huitema J. Rosenberg, J. Weinberger and R. Mahy. STUN - Simple Traversal of UDP Through Network Address Translators. IERF-Draft, December 2002.

[17] V. Jacobson. Pathchar - a tool to infer characteristics of internet paths. In *Presented at MSRI talk*, April 1997.

[18] V. Jacobson and S. McCanne. Visual Audio Tool (vat). In *Audio Tool (vat), Lawrence Berkley Laboratory*. Software on-line, ftp://ftp.ee.lbl.gov/conferencing/vat.

[19] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000.

[20] K. Lai and M Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, March 2001.

[21] J. Liebeherr and M. Nahas. Application-layer Multicast with Delaunay Triangulations. In *IEEE Globecom*, November 2001.

[22] A.M. Kermarrec M. Castro, P. Druschel and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications Vol. 20 No. 8*, Oct 2002.

[23] S. McCanne, E. Brewer, R. Katz, L. Rowe, E. Amir, Y. Chawathe, A. Coopersmith, K. Mayer-Patel, S. Raman, A. Schuett, D. Simpson, A. Swan, T. L. Tung, D. Wu, and B Smith. Toward a Common Infrastucture for Multimedia-Networking Middleware. In *Proceedings of NOSSDAV*, 1997.

[24] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. In *ACM Multimedia*, November 1995.

[25] V.N. Padmanabhan, H.J. Wang, and P.A Chou. Resilient Peer-to-peer Streaming. In *Proceedings of IEEE ICNP*, 2003.

[26] Peercast. http://www.peercast.org/.

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

[28] Planetlab. http://www.planet-lab.org/.

[29] Quicktime. http://www.apple.com/quicktime.

[30] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level Multicast using Content-Addressable Networks. In *Proceedings of NGC*, 2001.

[31] Real broadcast network. http://www.real.com/.

[32] R. Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. Technical Report TR98-1687, Cornell University Computer Science, 1998.

[33] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. In *ACM Computer Communication Review*, January 1997.

[34] B. Bhattacharjee S. Banerjee and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, August 2002.

[35] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, January 2002.

[36] Slashdot. http://www.slashdot.org/.

[37] S.McCanne, V.Jacobson, and M.Vetterli. Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM*, August 1996.

[38] Sorenson. http://www.sorenson.com/.

[39] S.Q.Zhuang, B.Y.Zhao, J.D.Kubiatowicz, and A.D.Joseph. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination, April 2001. Unpublished Report.

[40] Streamer. http://streamerp2p.com/.

[41] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *Proceedings of Fourth International Workshop on Networked Group Communication (NGC)*, October 2002.

[42] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI02*, pages 255–270, Boston, MA, December 2002.