RMX: Reliable Multicast for Heterogeneous Networks

Yatin Chawathe, Steven McCanne, Eric A. Brewer
Computer Science Division, University of California at Berkeley
{yatin,mccanne,brewer}@cs.berkeley.edu

Abstract— **Although IP Multicast is an effective network primitive for best-effort, large-scale, multi-point communication, many multicast applications such as shared whiteboards, multi-player games and software distribution require reliable data delivery. Building services like reliable sequenced delivery on top of IP Multicast has proven to be a hard problem. The enormous extent of network and end-system heterogeneity in multi-point communication exacerbates the design of scalable end-to-end reliable multicast protocols. In this paper, we propose a radical departure from the traditional end-to-end model for reliable multicast and instead propose a hybrid approach that leverages the successes of unicast reliability protocols such as TCP while retaining the efficiency of IP multicast for multi-point data delivery. Our approach splits a large heterogeneous reliable multicast session into a number of multicast *data groups* of co-located homogeneous participants. A collection of application-aware agents—Reliable Multicast proXies (RMXs)—organizes these data groups into a spanning tree using an overlay network of TCP connections. Sources transmit data to their local group, and the RMX in that group forwards the data towards the rest of the data groups. RMXs use detailed knowledge of application semantics to adapt to the effects of heterogeneity in the environment. To demonstrate the efficacy of our architecture, we have built a prototype implementation that can be customized for different kinds of applications.**

I. Introduction

The IP Multicast service model [1] extends the traditional best-effort Internet datagram delivery service for efficient multi-point packet delivery. Like IP unicast, the multicast service model is not reliable—packets may be dropped at any point along the distribution tree—and it defers services like reliable, sequenced delivery to higher layers. Although some applications such as real-time audio and video broadcasting [2], [3], [4] are tolerant to losses and thus well-suited to IP multicast, a number of applications that could benefit from efficient multicasting require reliable delivery. These include multi-player games, shared electronic whiteboards [5], software distribution and webcasting [6]. To support such applications, reliable multicast protocols such as RMTP [7], SRM [8] and PGM [9] build reliability on top of this best-effort service.

Although reliable multicast applications stand to reap enormous performance benefits from the underlying multicast service, they are fundamentally challenged by the heterogeneity that exists across the Internet. In the multicast domain, a communication source is potentially confronted with a wide range of path characteristics to each receiver, for example, different delays, link rates, packet losses, and competing congestion on the paths to the different receivers. In this scenario, if the source sends its data stream at the most constrained bit rate among all paths to all receivers, then many high-bandwidth receivers experience performance below the network's capability, whereas if the source sends at the maximum possible bit-rate, then low-bandwidth paths become congested and receivers behind these congested links suffer. This multiplicity of data paths and the possibility of multiple congestion points along independent sections of the paths imposes great difficulty on the design of an end-to-end scheme for reliable multicast.

Traditionally researchers have appealed to the end-to-end design principle [10] for building protocols to deal with reliability, congestion control, and rate adaptation. However, unlike reliable unicast protocols like TCP, reliable multicast protocols must deal with scale and heterogeneity. In spite of almost a decade of research on reliable multicast, no viable solution that solves these problems completely has emerged. Moreover, IP multicast deployment over the wide-area has yet to be comprehensively realized. Most Internet Service Providers have not deployed multicast largely due to management difficulties such as the absence of a robust inter-domain multicast routing protocol and lack of control over senders and receivers of data in a multicast session. In addition, although schemes have been built to provide TCP-friendly congestion control in the multicast realm [11], [12], these approaches have limitations. They work only with single-source sessions, and none satisfactorily accommodates bandwidth heterogeneity across the multicast distribution tree.

In this paper, we propose a hybrid approach to reliable multi-point communication that leverages well-understood and robust reliable *unicast* transport protocols and couples them with the multicast service model for efficient multi-point data delivery. Rather than assume the existence of a global multicast "dial-tone", we view IP multicast as an efficient protocol building block that need not be available everywhere. Our architecture is grounded in a hybrid communication model that partitions the heterogeneous multicast receiver set into a number of small homogeneous *data groups*, and uses robust unicast communication protocols across data groups. The model relies in part on end-to-end loss recovery mechanisms and in part on intelligent and application-aware adaptation carried out within the network. A network of application-aware agents—or Reliable Multicast proXies (RMXs)—uses detailed knowledge of application semantics to adapt to the heterogeneity constraints. This architecture allows local high bandwidth receivers to continue receiving data at a high rate from the source, while data sent to remote receivers is congestion-controlled by the RMXs.

The architecture relies on three key concepts. First, in order to localize the hard multicast problems of scalable loss recovery, congestion control and bandwidth allocation, we partition the large wide-area heterogeneous session into many smaller and simpler homogeneous sub-sessions. This *divide-and-conquer* approach effectively decouples each sub-session from the vagaries associated with the rest of the session partic-

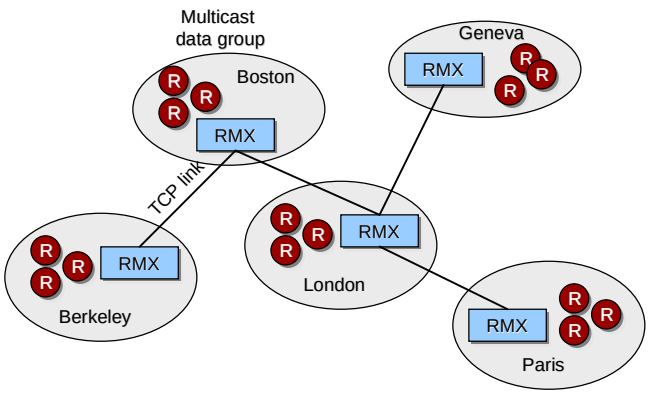

Fig. 1. **The RMX Architecture**.

ipants. Second, as data flows through an RMX, the RMX uses application-level knowledge to dynamically alter the content of the data or to adapt the rate and ordering of data objects. The RMX allows for the notion of *semantic reliability* as opposed to data reliability, that is, reliability of information rather than that of the representation of the information. Thus, by relaxing the semantics of reliability, we lift the constraint that all receivers advance uniformly with a sender's data stream; each receiver defines its own level of reliability and decides how and to what degree individual data objects might be transformed and compressed. Finally, to support these semantics, we leverage the *Application Level Framing* (ALF) [13] protocol architecture which says that application performance can be substantially enhanced by reflecting the application's semantics into the design of its network protocol. This approach to protocol design is a boon to protocol performance since the application is optimized for the network and vice versa.

In previous work [14], we presented a "stub-extension" solution where RMXs provided access to reliable multicast sessions for clients that were either impoverished (hand-held PDAs, etc.) or handicapped due to poor network connectivity. The RMX acted as a stub for the client and participated in a global multicast session feeding data from the global session to the client. In this paper, we generalize this approach to a wide-area distributed framework for tackling wide-area heterogeneity through a collection of RMXs arranged as an application-level overlay network. Our design has been strongly influenced by our experiences implementing the prototype RMX described in [14]. We have implemented the wide-area framework as application-customizable RMXs and built custom modules to support real applications.

In the remainder of this paper, we develop this overlay network architecture. Section II presents an overview of the RMX architecture. Section III presents details of data communication using this architecture. We describe a transport-independent naming scheme for data objects in a session, and present a data recovery scheme in the face of losses. In Section IV, we show how our architecture makes heavy use of ALF at all levels. Next, we describe how real applications interact with our architecture, and present some evaluation of the framework. Finally, we summarize other research related to our approach, present some future work and our conclusions.

II. The RMX Architecture

In the unicast world, TCP is the dominant protocol by which applications achieve reliable communication. TCP works well across the wide area because it adaptively accommodates a large range of network rates, delays and losses. TCP employs a feedback loop between the sender and the receiver that detects and reacts to congestion. Our architecture leverages this robust and congestion-friendly behavior of TCP to assist in wide-area multicast. Figure 1 illustrates the RMX architecture. Rather than rely on a single global multicast communication channel, we partition the logical channel into a number of smaller *data groups*. This divide-and-conquer approach results in a topological clustering of receivers into homogeneous clusters thereby reducing the wide-area heterogeneity problem into a simpler problem of rate adaptation within homogeneous clouds and point-to-point rate adaptation across clouds.

A participant in the RMX session (a source or a receiver) joins the session by subscribing to its local data group. Each data group is an independent, locally-scoped multicast channel containing a representative RMX that is strategically placed in the network to service its data group. The RMXs spread across the data groups organize themselves into a spanning tree via unicast interconnections. An RMX communicates with its peer RMXs across the wide-area using a reliable unicast protocol such as TCP and relies on simple local-area multicast congestion control schemes within each data group. Since each data group consists of roughly homogeneous clients, multicast in such a homogeneous environment becomes tractable.

This model of partitioning a multicast session into a number of smaller data groups interconnected by unicast links is based on a framework that we call *scattercast* [15]. Scattercast is a delivery-based data forwarding service for multi-point communication where data is delivered to multiple receivers through a combination of unicast and multicast forwarding. The RMX architecture builds on the scattercast model by integrating application-specific intelligence and semantics into the forwarding service.

Each RMX participates in the multicast channel associated with its data group. In addition, it listens for data from any of the unicast connections to its neighboring RMXs in the spanning tree. Thus, RMXs form an application-level overlay network listening for data on all of their "links" (unicast connections to other RMXs or the multicast data group) and forwarding incoming data to other RMXs using scattercast. In the event of a packet loss, the receivers and RMXs cooperatively initiate an end-to-end recovery protocol that attempts to recover losses from local receivers before propagating the loss report towards the source. In order to identify data objects or missing data fragments across the overlay RMX network, we need a common framework to name individual data objects in a man-

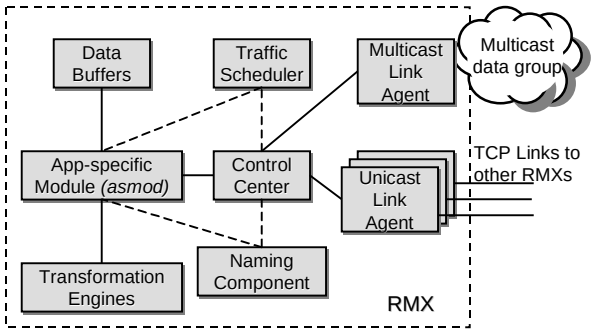

Fig. 2. **The internals of an RMX**.

ner that is independent of the underlying unicast or multicast transport protocols. This argues for an application-level naming scheme that is embedded in the RMX transport. In Section III, we describe in detail the data communication model that is used to deliver data across the RMX network, recover missing data, and name data consistently across the entire framework.

Figure 2 shows the individual components of an RMX. Each link into the RMX has an associated *link agent* that implements the specific details of the (unicast or multicast) reliability protocol used on that link. The *control center* is the coordination and dispatch unit that shuttles data coming in from any link through the rest of the system. Each RMX has a customizable *Application-Specific MODule* or *asmod* that implements the application-specific pieces of the RMX. The *asmod* is a dynamic code library that is loaded into the RMX at run-time. The control center passes incoming data to the *asmod* which, in turn, decides how the data is forwarded to the other links. The *asmod* relies on specialized *transformation engines* to potentially convert the data into a different representation before forwarding it. A cache of data buffers stores data temporarily before it is forwarded.

To realize an RMX-controlled multicast session, RMX agents must be placed in the network at strategic locations and organized into an overlay network. Co-located receivers with similar network characteristics should organize themselves into data groups and elect an appropriate location within the group for their representative RMX. Placing RMXs strategically to form an overlay network is a hard problem. It requires some scheme for determining and exploiting the structure and topology of the underlying multicast distribution tree. For the purpose of this work, we assume that such a protocol exists to cluster receivers into data groups and to build the RMX overlay network across data groups. Our implementation uses manual configuration of receivers into independent data groups that are bridged by a manually constructed spanning tree of RMXs. In Section VII, we discuss some techniques that we are currently investigating for dynamically forming data groups and constructing the overlay network of RMXs.

III. The Data Communication Model

Our split unicast-multicast communication model requires special mechanisms to ensure that data propagates properly throughout the entire RMX session. This section details how communication is achieved across the RMX network. Although the architecture does not require any specific RM protocol within data groups or unicast protocol across RMXs, we have chosen Scalable Reliable Multicast (SRM) [8] and TCP as the underlying transport protocols.

A. Data Delivery

Data delivery over the RMX network uses scattercast forwarding. A source that generates data distributes it to its local data group using SRM. The local group members including the representative RMX use SRM's built-in data recovery mechanisms to recover any lost data. The RMX forwards data from the local group towards the participants in the other groups. Whenever the RMX receives data on a link, it forwards the data to all of its remaining links including its local data group using spanning-tree flooding [16]. As the data flows from RMX to RMX it eventually propagates to the entire session.

Since all participants in a single data group have essentially similar network characteristics, bandwidth management and congestion control are straightforward. Over the wide-area, however, data flows via TCP connections which have built-in congestion- and flow-control. Given the heterogeneity over the wide area, different TCP links in the RMX network are likely to have different bit-rates and delays. Thus data may flow into an RMX faster than it can be forwarded. The RMX addresses this by using a combination of the following mechanisms:

- If we assume infinite buffer-space in the RMX, the RMX can buffer incoming data while it tries to forward it along the low-bandwidth connections.
- With finite buffering, the RMX can drop incoming data as its buffers fill up and rely upon end-hosts to recover the dropped data.
- It can adopt an end-to-end flow control scheme whereby it informs the RMX along the upward stream towards the source to slow down its sending rate. Such a feedback mechanism can eventually bring the source transmission down to a rate that is suitable for the RMXs to handle.
- The RMX can transform incoming data and forward lower bit-rate versions of the data.

Our notions of semantic reliability and ALF enable application-specific adaptation and transformation of the data that arrives into an RMX before it is forwarded on to the other links. For example, if an incoming data object is a high-resolution image, the RMX may transform it to a lower-resolution representation for faster delivery across low-bandwidth links. Alternatively, it may transform the image into a progressive representation (such as progressive JPEG) and send each scan of the progressive image as an independent layer. Depending on the capabilities of the receiving client devices or applications, the RMX may also convert the data to a different format that is compatible with the clients. Clients,

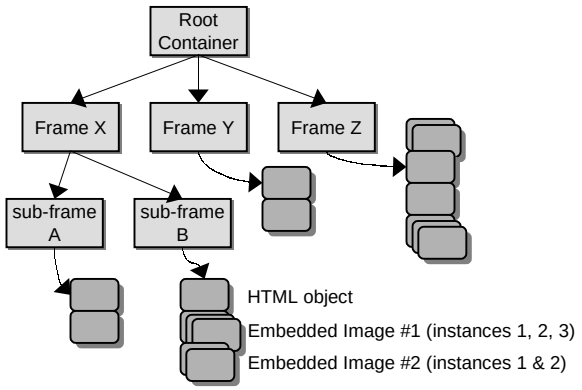

Fig. 3. **A TINS hierarchy:** A web document consisting of multiple document frames.

in turn, can request the original high resolution data from the source or local RMXs if they so desire.

B. End-to-end Reliability

In addition to delivering data, we need an end-to-end data recovery scheme that allows applications to request missing data or to request a different (more refined) representation of a data object. Each transport-level hop from the source to the receiver in the RMX framework is reliable (using either SRM or TCP). If we assume infinite buffering capabilities at RMXs, then there are no drops within the RMXs themselves and all data eventually propagates to all receivers. However, if an RMX has finite buffer space, and the incoming data rate exceeds the rate at which the RMX can forward outgoing data objects, then the RMX must drop data. Even if RMXs do not drop any data, receivers may still wish to recover a different representation of an object from the one that the local RMX forwarded. This requires an end-to-end recovery mechanism that allows receivers to recover missing data from the source or across the RMX network.

When a receiver requests a piece of data, the request is multicast to the receiver's data group using SRM. If the data cannot be recovered from the local group, then the group's RMX forwards the recovery request upward along the RMX hierarchy towards the source of the data. Intermediate RMXs first attempt to recover the data from their local data group, and if that fails, forward the request toward the source.

To determine the direction toward the source, RMXs use a scheme similar to that used by *learning bridges* to discover the route toward hosts on a LAN [16]. Each RMX maintains a table that is used to determine the path toward sources from that RMX. When data arrives on an incoming link from a source, the RMX makes an entry in the table mapping that source to the incoming link along with a timestamp indicating the time when the entry was last updated. Every time new data arrives on that link from the same source, the timestamp is updated. Periodically, a process sweeps through the table purging old entries. This ensures that recovery proceeds in the right direction even in the face of dynamic reconfiguration of sources and RMXs.

To limit the scope of retransmitted data, we use a scheme based on PGM (PraGmatic Multicast) [9] and BCFS (Bread-Crumb Forwarding Service) [17]. When an RMX forwards a recovery request towards the source, it records the link on which the retransmitted data must be returned. If another recovery request for the same data arrives, the RMX suppresses the new request and records the new link for that request in its previously saved state. When retransmitted data arrives, the RMX forwards it only along those links that had requested that data. The per-link state is "soft" and is eventually timed out to ensure correct operation in the event of loss of the retransmitted data.

C. Data Naming

As data flows through the RMX overlay network, it traverses a number of links some of which run an instance of TCP while others are multicast links running SRM. Each link associates its own transport-layer name with the data. For example, TCP uses byte-level sequence numbers while SRM uses container IDs and sequence numbers [18]. In order to manipulate data across the entire overlay network, we require a naming protocol that is consistent across the entire network. Thus, instead of relying on transport-layer identifiers to name data, we allow applications to define their own names and preserve these names as data flows across the RMX network.

The problem of data naming is further complicated by the fact that RMXs are allowed to transform data before forwarding. End-participants must be able to distinguish between various representations of a data object and request specific representations if they so desire. Our solution is to use a well-defined naming structure for data that exposes the notion of multiple representations of the same data object to the application. We call this *TINS* or Transport-Independent Naming System.

We derive our naming scheme from SNAP (Scalable Naming and Announcement Protocol) [18], a hierarchical naming scheme for data within an SRM session. SNAP organizes application data into ADUs (Application Data Units). The entire namespace for ADUs is organized hierarchically into a set of containers. Applications assign ADUs to specific containers. Each container is assigned a SNAP container ID by the protocol. Within a container, objects are assigned a unique sequence number. Thus, any object in an SRM namespace has a name defined by a $\{$*container ID, sequence number*$\}$ pair. Although such a hierarchical namespace may not be well-suited for certain applications, it is general enough for most common application data.

Rather than allow the transport protocol to assign its own identifiers to the namespace (such as byte sequence numbers in TCP or container IDs in SRM), TINS extends SNAP to explicitly allow applications to assign their own names to containers. These application-level names are preserved in the transport protocol. Figure 3 shows an example of a TINS namespace for disseminating a web document that consists of multiple document frames. Each frame in the document is represented in the

namespace as a container. A frame may contain other frames or a sequence of data objects that represent the content of the frame. These objects may be HTML pages, embedded images, Java applets, etc. Another example of a namespace is one for disseminating news reports: the various sections of the news, such as headlines, business news, sports, etc., are grouped into separate containers, and news items within each section are disseminated as ADUs within the corresponding container.

Each object generated by a source has an application-specific TINS name. We call this the *primary name* of the object. In addition to the primary name, a source or any RMX may attach a secondary or *instance name* to a specific representation of an object. Each different representation of the object must have a unique instance name. For example, suppose a source generates a high-resolution image object. If an RMX transforms this image to a lower resolution, the RMX assigns a new instance name to the transformed representation while at the same time preserving the original primary name generated by the source.

Within each data group, the TINS namespace is propagated using SRM's built-in delivery and recovery mechanisms. Across the RMX overlay network, the namespace is propagated using scattercast forwarding and recovery as described in Sections III-A and III-B. Whenever a source constructs a new container, a message is transmitted to the rest of the session announcing the existence of the new container. In order to allow receivers to detect and repair losses of container announcements or ADUs in the namespace hierarchy, the source periodically transmits a "signature" of the current state of the root of the namespace. The signature of a container is defined recursively as an MD5-hash function of the sequence number of the last ADU transmitted within that container and the signatures of all its children containers. A receiver can compare the signature announcement with its own state to detect losses. Losses of namespace announcements are repaired using the same mechanisms that are used to repair data. Details of the namespace announcement and recovery protocol are described in [18].

If a receiver is missing a certain data object, it can issue a recovery request for a specific instance of the object by specifying the instance name in its recovery request, or issue a request only for the primary name in which case any instance can satisfy that request. Thus an application that receives a low-resolution or transformed representation of an object can request for the original highest-quality instance with an explicit request.

IV. Application-defined Semantic Reliability

Our initial prototype of the RMX described in [14] was tailored to a specific application: a shared electronic whiteboard for PDAs. Although it integrated application knowledge in its transformation and forwarding mechanisms, it did so in an ad hoc manner and it was relatively difficult to generalize the prototype to other applications. Instead, an RMX should use a common well-defined interface to the application that encompasses flexible application policies. The application-specific module (*asmod*) in the RMX provides a mechanism to embed such policies. The *asmod*s specialize the RMX operation in three ways:

- data reliability,
- transmission scheduling, and
- dynamic data transformation.

We discuss each of these in detail.

A. Application-customizable Data Reliability

Traditional transport protocols such as TCP fail to capture the structure that an application may need to intelligently control and adapt to the communication path. For example, if the transport library detects a loss of sequence numbers 4567 through 6012 and queries the application as to whether it should recover the missing data, the application has no idea what those sequence numbers correspond to in its own data space. Similarly, it is impossible to specify flexible ordering constraints for data using a flat sequence space. To overcome this, our hierarchical naming scheme, TINS, allows applications to incorporate a well-defined and semantically rich naming structure in the transport protocol. This makes it easy for receiving applications and RMXs to control protocol behavior.

In TINS, containers within a namespace are the unit of selective reliability: an application can choose to apply different reliability and ordering semantics to different containers. If an application does not care about certain sub-spaces of the data namespace, then the *asmod* may decide never to issue recovery requests for missing data in that sub-space. For example, a web page application may be configured to ignore images; hence if an image object is missing and needs recovery, the application or the *asmod*s can instruct the protocol library to ignore the missing object and avoid unnecessary retransmissions. This structured naming scheme thus permits applications to specify different kinds of loss recovery and ordering policies via the hierarchy of containers and objects.

B. Application-customizable Transmission Scheduling

In addition to customizing data recovery in an application-specific manner, the RMX provides hooks to the *asmod*s to schedule transmission or forwarding of data depending on the application's and receivers' requirements. Applications may prioritize data objects based on receiver interest or capabilities.

As with application-driven reliability, we use a hierarchical data organization to allow the RMX to schedule data for transmission. The traffic scheduler in the RMX constructs a hierarchy of *traffic classes* for bandwidth allocation. The *asmod* assigns each traffic class a certain percentage of the available bandwidth. When the *asmod* has a data object to transmit, it assigns the object to a specific traffic class. The scheduler honors the bandwidth restrictions for each traffic class while scheduling the data for transmission over the network. The control traffic such as namespace dissemination and recovery requests are assigned their own independent traffic class to ensure that they do not conflict with application data. As an example of a traffic class hierarchy, a web-page dissemination application may choose to create two top-level traffic classes, one for HTML and one for embedded images, and give a higher percentage of

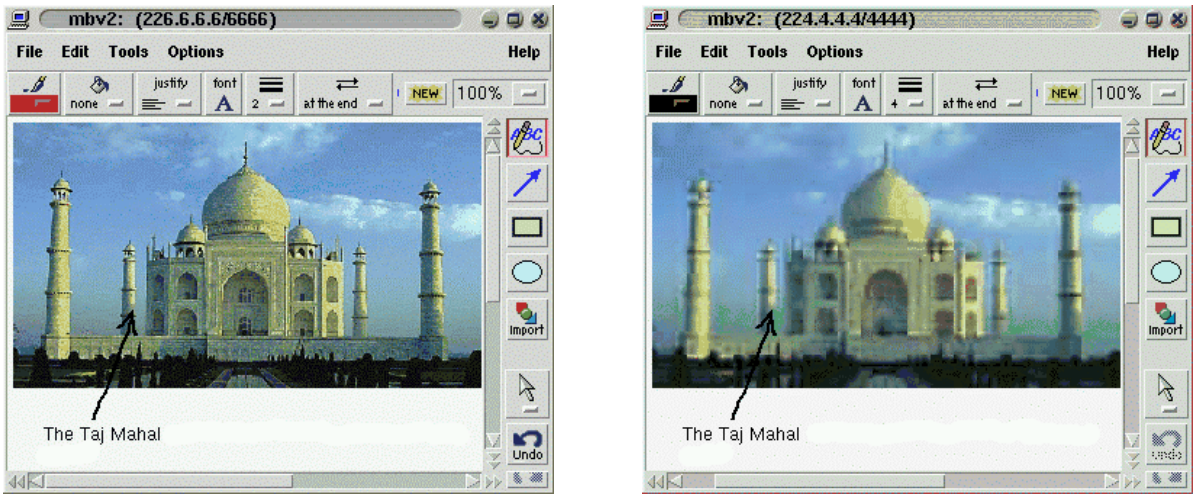

Fig. 4. **The mediaboard application:** The app on the left is the source, and the one on the right is a low-bandwidth receiver.

the bandwidth to the HTML class. Moreover, if we use progressive image formats, one may envisage splitting the image traffic class into one for the first scan of the progressive images and another for the refinement scans.

We can use any suitable traffic scheduler in the RMX such as Hierarchical Round Robin (HRR) [19], Start-time Fair Queuing (SFQ) [20], Hierarchical Fair Service Curve (HFSC) [21], or Class-Based Queuing (CBQ) [22]. Our current implementation relies on the HFSC scheduler.

C. Application-customizable Transformation

Our third mechanism for injecting application semantics into the RMX is the use of specialization code for performing application- and data-specific transformation operations to convert data objects on the fly inside the RMX. These transformations depend heavily on the individual applications under consideration.

The *asmod* decides what kinds of transformations are to be applied to data before it is forwarded. It may do so in a statically determined manner (e.g. convert all images to progressive format), or rely on feedback from the traffic scheduler or the link agents to determine the extent of transformation required. For example, if a link agent can notify the RMX whenever it detects congestion in the network, the *asmod* can use this information to aggressively transform high-resolution data to lower resolutions. Similarly, *asmod*s can adjust their transformation granularity based on TCP throughput measurements across RMXs.

Although the TCP layer does not provide sufficient feedback to the application, one can envisage using a more ALF-like unicast transport protocol based on the *Congestion Manager* work at M.I.T. [23]. Such a transport protocol can be better integrated into the RMX: it allows the *asmod* to dynamically determine the available bandwidth constraints and can notify the *asmod* whenever it detects congestion in the network. The *asmod* can in turn use this information to determine the amount of transformation to apply to incoming data.

V. RMX Applications

To illustrate the viability of the RMX architecture, we now look at concrete examples of how RMXs are specialized for different applications.

A. Shared Electronic Whiteboard

The original shared electronic whiteboard application, *wb* [5], was developed at the Lawrence Berkeley Laboratory. Based on experiences with *wb*, researchers at Berkeley have built a new whiteboard tool, *mediaboard* [24], that uses the Berkeley SRM library *libsrm* [25]. This application allows a diverse set of media to be created and displayed interactively by a group of users sharing a multicast session. A mediaboard session consists of a shared presentation space that is divided into a number of canvas pages. It supports traditional whiteboard data types such as line drawings and text, and adds support for other media such as images and postscript files.

Incorporating this application into our framework involved two tasks: making the application aware of the RMX naming protocol TINS, and implementing an *asmod* that understood mediaboard semantics. The mediaboard *asmod* was customized in the following ways:

- The TINS namespace for the mediaboard consists of a root container and many second-level containers, one for each page in the session. Operations on a particular page are objects within the corresponding container. When objects are created by the source, they are assigned a null instance name. If an RMX transforms an object, it assigns the new representation a different instance name that describes the transformation that was applied.
- The *asmod* converts all high-resolution images in the session to progressive JPEG representations. Each scan of the progressive JPEG is identified as a separate instance of the original data object. This allows the *asmod* to selectively send only a certain number of scans depending on available bandwidth and to pri-

oritize the base scans over the refinement scans. The receiving application accumulates successive scans and displays them as they arrive from the network. Each scan is assigned an instance name that defines the scan as part of a progressive representation and includes the scan number of this scan as well as the total number of scans that make up the image.

- The *asmod* prioritizes low-bandwidth operations such as lines, geometric shapes, move, copy, delete, etc. over high-bandwidth data like images. In addition it incorporates receiver feedback while allocating bandwidth. Receivers periodically send reports to their local RMX informing it of the page that they are currently viewing. The *asmod* at the RMX aggregates these reports and propagates them to upstream *asmod*s. The *asmod*s use these reports to assign bandwidths to pages that are proportional to the number of receivers that are looking at that page. This form of receiver-driven bandwidth adaptation is based on consensus-driven schemes described in [26].
- The *asmod* relies on the traffic scheduler to perform traffic prioritization and bandwidth allocation. Each page on the mediaboard has a corresponding traffic class. The page traffic class is sub-divided into two classes, one for the low-bandwidth data and the other for high-bandwidth objects. The high-bandwidth traffic class generally tends to contain image data which the *asmod* transforms into progressive scans; hence, this class is further sub-divided into classes for the base scans and for the refinement scans. Each page traffic class is allocated a bandwidth that is proportional to the number of receivers that are looking at that page. Thus, in the common case, with all local receivers viewing the page with current activity, most of the bandwidth is allocated to that page.

Figure 4 shows an example of two mediaboard applications: the source and a receiver behind a low bandwidth link. The source generates a high resolution image on the mediaboard which is translated into multiple progressive scans at the RMX. The scans trickle down to the receiver over the low bandwidth connection. In the figure, the receiver has accumulated only the first two scans of the original image. Over time, the rest of the scans eventually arrive at the receiver and it can reconstruct the full-quality image. With the RMX, the receiver is able to render an approximate version of the image as soon as the first scan arrives; the approximation is refined as more scans reach the receiver.

B. Push-based applications

Our second example application is a push-based information dissemination application called *InfoCast* [27]. This application periodically transmits updates of a live data feed, such as news headlines, stock quotes, weather updates, etc. to a group of receivers. The InfoCast source constructs a namespace hierarchy that groups related information into containers. For example, a stock quotes service creates a container for each symbol on the stock exchange; the latest stock quotes for each symbol are transmitted as data objects in the corresponding container. Similarly, a news service creates containers for each news section (headlines, world news, business, sports, etc.).

Researchers in our project have built a stand-alone version of the InfoCast application. We are currently in the process of integrating the application into the RMX framework. Like the mediaboard, InfoCast uses rich content such as images and HTML. Hence, transformations similar to those used for the mediaboard application can be used in the InfoCast *asmod*. The *asmod* converts images to low resolution versions or to progressive representations. In addition, since InfoCast data is periodic, new data replaces old data. For example, in a newscast scenario, later news obsoletes older pieces of news, thus obviating the need to recover any missing old news. The *asmod* uses this feature of the data to its advantage and suppresses recovery for any missing data that might have been superseded by newer data.

In addition, receivers indicate their interest in certain categories of data. For example, in a weather-cast application, receivers in Boston may only care about weather information in Boston and its surroundings, while receivers in San Francisco would be more interested in weather for the San Francisco Bay region. The *asmod*s use this feedback from receivers to filter out unnecessary data objects.

VI. EVALUATION

As part of our design process, we implemented a prototype of the RMX framework using the MASH toolkit [28] as our development platform. This toolkit is a Tcl/C++-based programming framework for multimedia networking applications developed by the MASH research project at UC Berkeley. Our early efforts at building the prototype did not include a well-defined transport-independent naming scheme. Our experiences with that prototype led to the evolution of TINS. With a structured naming protocol that exposed application structure to the RMXs and RMX data transformations to the applications, the design of the applications and *asmod*s was greatly simplified. Our current implementation assumes infinite buffering at the RMX and provides for application-specific transformations and data scheduling. We have built an *asmod* for the mediaboard application, and are currently building one for InfoCast.

We have performed some preliminary evaluation of our framework using the VINT network simulator *ns* [29]. The goal of these simulations is not to perform exhaustive measurements of the system, but simply to demonstrate that the architecture is feasible and is able to handle Internet heterogeneity. A more thorough analysis and measurement of the architecture will require greater experience with a mature system and a wider range of applications, and will be the focus of future research.

Our experimental topology is depicted in Figure 5. It consists of two local-area networks, each with three nodes, separated by a low-bandwidth wide-area link. To measure the effectiveness of our framework, we conducted two sets of experiments: one in which the source and all receivers participated in a single global SRM session, and another in which the participants on each side of the wide area link formed their own data groups and RMXs at nodes 4 and 5 interconnected the data groups via

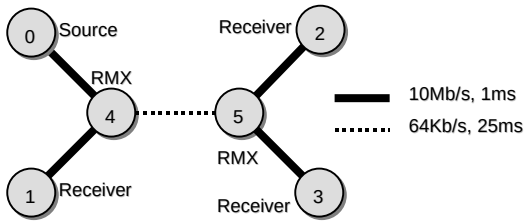

Fig. 5. **The simulation topology**.

a TCP connection. To be fair, the results from these experiments are biased against SRM since our SRM implementation does not incorporate any local recovery or wide-area congestion control schemes. Yet, they demonstrate the graceful adaptiveness of our architecture to network heterogeneity.

To simulate data from the mediaboard application, we attached a constant bit-rate data stream to the source; this stream modeled a steady stream of 20KB images sent from the source. In the RMX experiment, the RMX at node 4 transformed all images to progressive JPEG representation. To model this transformation, we ran a separate experiment and determined the average time for transforming a 20KB image through our progressive JPEG transformation engine. On an Intel Pentium 300MHz machine, the engine took approximately 130ms to perform the transformation. We added a delay corresponding to this transformation time to the RMX at node 4.

Figure 6 shows the losses that occur in the SRM sessions in both sets of experiments at different data rates. As long as the source's data rate was low enough not to overflow the bottleneck link, the global SRM session demonstrated no loss. But as we increased the data rate beyond the bottleneck capacity, the loss rate rapidly increased. On the other hand, with the RMX architecture, each SRM session was isolated in the local area high-bandwidth data groups, and so observed no loss.

Figure 7 depicts the arrival times of image ADUs at each receiver. In the global SRM case, as long as the sending data rate does not exceed the bottleneck bandwidth, all receivers progress evenly, receiving data at a steady pace. However, as soon as we reach the bottleneck bandwidth, the network drops packets and receivers 2 and 3 issue recovery requests to recover missing image data thus increasing the arrival times for those images. Increasing the sending rate beyond the bottleneck capacity only ends up flooding the network resulting in increased packet loss and recovery traffic.

As expected, with RMXs, even when the sender's rate is increased to 5Mb/s, the RMX at node 4 shields the low-capacity link from the high-rate data group. Thus receiver 1 can take full advantage of its high connectivity to the source and receive data that full speed while the RMX reduces the rate before sending it over the bottleneck link. Moreover, the RMX converts the images to progressive JPEGs. The base scan of the progressive representation is only a fraction of the original image size. Since each scan is sent as an independent instance of the image, the receivers can render the base scan as soon as they receive it.

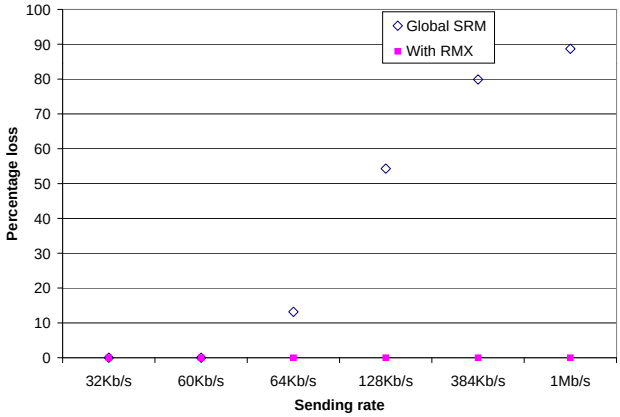

Fig. 6. **Comparison of SRM losses using global SRM versus the RMX architecture**.

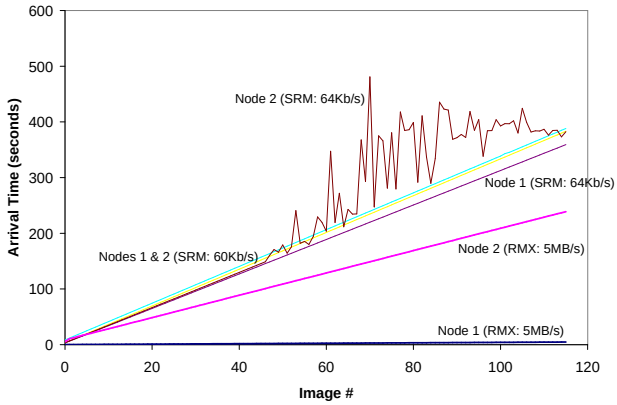

Fig. 7. **Comparison of arrival rate of data using global SRM versus the RMX architecture**.

The receivers behind the bottleneck link proceed at the fastest speed allowed by the TCP congestion control mechanisms. Although RMXs introduce additional delay due to transformation, this is offset by the gains in performance from robust congestion control and faster downloads due to transformation.

VII. Future Work

Our current implementation of the RMX architecture ignores some critical issues that must be addressed for the architecture to be widely deployable. We assume static placement of RMXs in the network and static configuration of receivers into data groups. Placing RMXs strategically in an overlay network and grouping co-located receivers into topologically sensitive data groups is a hard problem. We are currently working on a promising approach for dynamically grouping receivers and building a dynamic and efficient distribution tree across RMXs. Our approach is based on a simulated annealing process that attempts to gradually approach an optimal distribution tree using periodic random measurements. Other researchers have pro-

posed similar approaches to this problem in [30] and [31].

In addition to group formation, the RMX architecture needs to be robust to network failure and must be able to adapt to changing network conditions and faults. RMXs should automatically recover from network node failures. The Active Service framework [32] is a promising direction for solving some of these issues.

As described in Section IV-C, the RMX architecture can benefit from the Congestion Manager [23] framework for adapting to network congestion. Feedback from the CM can be used to drive the RMXs adaptation policies, to determine the aggressiveness of data transformation, etc. This is another avenue of research that we plan on exploring in the future.

VIII. Related Work

The notion of proxies as intermediaries between clients and servers has been around for a while. Numerous proxy mechanisms have been proposed for HTTP. Proxies have been used as caching and pre-fetching agents [33] to hide the latency of fetching data from across the network. [34] presents a generic framework for Internet proxy services. In the context of multicast, [35] is a proxy framework for real-time audio/video data.

Content layering is another scheme that has been widely used to tackle heterogeneity in multicast environments [36], [37], [38]. Layering typically involves encoding the source data into multiple layers; the base layer provides an approximate representation of the original data, and each additional layer provides more information about the original data.

Rizzo et al. [39] describe a Reliable Multicast data Distribution Protocol (RMDP) that relies on Forward Error Correction techniques to adapt to client and network heterogeneity. Digital Fountain [40] is a scheme that encodes data in a form that it can be reconstructed from any subset of the encoding packets that is equal in length to the source data. In combination with a clever layering strategy, this scheme can service a heterogeneous set of receivers, where each participant receives the encoded fountain at whatever rate best suits its network capacity.

RMTP [7] is a tree-based reliable multicast protocol that organizes group members into a hierarchical tree structure and aggregates acknowledgments at midpoints in the network. Each branch in the tree has a designated receiver (DR) to receive acknowledgments from its children and aggregate them upwards to the sender. The RMX architecture is similar to RMTP in that it groups receivers around an RMX just as RMTP groups receivers around DRs. But, RMTP is entirely a transport-level protocol with no application intelligence. On the other hand, the RMX architecture explicitly involves the application at all levels of its transport decisions.

Yallcast [30] and EndSystem Multicast [31] are frameworks for multi-destination content distribution that build a "multicast" distribution tree using unicast connections between end-clients, without relying on any intelligence in the network infrastructure. In contrast, the RMX architecture explicitly uses infrastructure support in the form of application-aware proxies to construct the distribution tree as well as for dealing with heterogeneity in the content dissemination process.

A lot of work has been done recently to dynamically discover the multicast topology and build localized clusters of multicast receivers. Levine et al. [41] proposed the use of IGMP MTRACE packets to allow receivers to obtain their path to the source of a multicast group; receivers use the multicast path information to determine how to achieve local error recovery and effective congestion control. Self-organizing Transcoders (SOT) [42] are a scheme for dynamic adaptation of continuous-media applications to varying network conditions by allowing groups of co-located receivers that experience losses due to a bottleneck link to elect a representative transcoder for local repair. In [43], Ratnasamy et al. use a multicast-tree-inference algorithm to build a protocol building block—a distributed Group Formation Protocol (GFP)—that allows receivers to self-organize into a source-rooted hierarchy of disjoint multicast groups where the hierarchy is congruent with the multicast tree topology from the source of the session.

Other researchers have proposed the use of intelligent computing in the network to assist in the design of reliable multicast protocols. Active Reliable Multicast (ARM) [44] uses the concept of "active routers" that can perform customized computation on behalf of the end-points. They provide best-effort soft-state storage and perform data caching for local retransmission, NACK fusion/suppression, and partial multicasting for scoped retransmission.

IX. Summary

We have presented an architecture for reliable multicast that allows applications to dynamically adapt to the heterogeneity that is inherent in the wide-area Internet. Our solution relies on intelligent agents in the network (Reliable Multicast proXies or RMXs) that mitigate network heterogeneity by acting as intermediaries between the source and the consumer of the data and dynamically adapting the content and/or the rate of the data to best suit the clients' needs and interests. An overlay network of RMXs spread across the wide-area Internet isolates co-located multicast receivers into homogeneous data groups thereby localizing the hard multicast problems of scalable loss recovery, congestion control and bandwidth allocation. To extend reliability across the wide area, RMXs form a spanning tree of TCP connections, thereby reaping the benefits of TCP-friendly wide-area behavior while retaining the efficiency of multi-point communication in the local area. We have built a prototype of our architecture that allows RMXs to be customized for specific applications via *asmod*s (or Application-Specific MODules). Our example applications demonstrate the programmability of our architecture to suit different flavors of applications. The RMX architecture is a new direction for resolving heterogeneity in the context of reliable multicast by moving application intelligence into the network infrastructure, while at the same time maintaining compatibility with the existing IP multicast architecture.

ACKNOWLEDGMENTS

We would like to thank our colleagues at Berkeley and the anonymous reviewers for their useful feedback. This research was supported by DARPA contract N66001-96-C-8508, by the State of California under the MICRO program, and by grants from Fuji Xerox, IBM, Intel, Microsoft, and Philips.

REFERENCES

[1] Stephen E. Deering, *Multicast Routing in a Datagram Internetwork*, Ph.D. thesis, Stanford University, Dec. 1991.

[2] Van Jacobson and Steven McCanne, *Visual Audio Tool*, Lawrence Berkeley Laboratory, Software available at `ftp://ftp.ee.lbl.gov/conferencing/vat`.

[3] Steven McCanne and Van Jacobson, "*vic*: A Flexible Framework for Packet Video," in *Proceedings of ACM Multimedia '95*, San Francisco, CA, Nov. 1995, pp. 511–522.

[4] Mark Handley, *Session DiRectory*, University College London, Software available at `ftp://cs.ucl.ac.uk/mice/sdr/`.

[5] Steven McCanne, "A Distributed Whiteboard for Network Conferencing," Class report, UC Berkeley, May 1992.

[6] PointCast Inc., "PointCast Home Page," `http://www.pointcast.com/`.

[7] John C. Lin and Sanjoy Paul, "RMTP: A Reliable Multicast Transport Protocol," in *Proceedings of IEEE Infocom '96*, San Francisco, CA, Mar. 1996, pp. 1414–1424.

[8] Sally Floyd, Van Jacobson, C. Liu, Steven McCanne, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," in *Proceedings of ACM SIGCOMM '95*, Boston, MA, Aug. 1995, pp. 342–356.

[9] Tony Speakman, Dino Farinacci, Steven Lin, and Alex Tweedly, *Pragmatic General Multicast (PGM) Reliable Transport Protocol*, CISCO Systems, 1998, Internet Draft.

[10] J. Saltzer, D. Reed, and D. Clark, "End-to-end Arguments in System Design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 195–206, 1984.

[11] Lorenzo Vicisano, Luigi Rizzo, and Jon Crowcroft, "TCP-like Congestion Control for Layered Multicast Data Transfer," in *Proceedings of INFOCOM '98*, Mar. 1998.

[12] Brian Whetten and Jim Conlan, "A Rate-based Congestion Control Scheme for Reliable Multicast," GlobalCast Communications, Oct. 1998.

[13] D. D. Clark and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," in *Proceedings of ACM SIGCOMM '90*, Philadelphia, MA, Sept. 1990.

[14] Yatin Chawathe, Steve Fink, Steven McCanne, and Eric Brewer, "A Proxy Architecture for Reliable Multicast in Heterogeneous Environments," in *Proceedings of ACM Multimedia '98*, Bristol, U.K., Sept. 1998.

[15] Sylvia Ratnasamy, Yatin Chawathe, and Steven McCanne, "A Delivery-based Model for Multicast Protocols using ScatterCast," *Unpublished*, July 1999.

[16] Radia Perlman, *Interconnections: Bridges and Routers*, Addison Wesley, Reading, MA, 1992.

[17] Koichi Yano and Steven McCanne, "The Breadcrumb Forwarding Service and the Digital Fountain Rainbow: Toward a TCP-friendly Reliable Multicast," Unpublished, June 1999.

[18] Suchitra Raman and Steven McCanne, "Scalable Data Naming for Application Level Framing in Reliable Multicast," in *Proceedings of ACM Multimedia '98*, Bristol, U.K., Sept. 1998.

[19] C. R. Kalmanek, H. Kanakia, and S. Keshav, "Rate-controlled Servers for Very High-Speed Networks," in *Proceedings of the IEEE Conference on Global Communications*, Dec. 1990.

[20] Pawan Goyal, Harrick M. Vin, and Haichen Cheng, "Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," in *Proceedings of ACM SIGCOMM '96*, Stanford, CA, Aug. 1996.

[21] Ion Stoica, Hui Zhang, and T. S. Eugene Ng, "A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service," in *Proceedings ACM SIGCOMM '97*, Cannes, France, Sept. 1997.

[22] Sally Floyd and Van Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, Aug. 1995.

[23] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," in *Proceedings of SIGCOMM '99*, Cambridge, MA, Sept. 1999.

[24] Teck-Lee Tung, "Mediaboard: A Shared Whiteboard Application for the MBone," M.S. thesis, University of California, Berkeley, Dec. 1997.

[25] Suchitra Raman, Yatin Chawathe, and Steven McCanne, libsrm*: The Berkeley SRM toolkit*, University of California, Berkeley, Software available at `http://www-mash.cs.berkeley.edu/mash/software/srm2.0`.

[26] Elan Amir and Steven McCanne Randy Katz, "Receiver-driven Bandwidth Adaptation for Light-weight Sessions," in *Proceedings of ACM Multimedia '97*, Seattle, WA, Nov. 1997.

[27] Tina Wong, Thomas Henderson, Suchitra Raman, Adam Costello, and Randy Katz, "Policy-Based Tunable Reliable Multicast for Periodic Information Dissemination," in *Proceedings of WOSBIS*, Dallas, TX, Oct. 1998.

[28] Steven McCanne et al., "Toward a Common Infrastructure for Multimedia-Networking Middleware," in *Proceedings of the Seventh International Workshop on Network and OS Support for Digital Audio and Video*, St. Louis, Missouri, May 1997, Association for Computing Machinery.

[29] Steven McCanne and Sally Floyd, *The LBNL/UCB Network Simulator*, Lawrence Berkeley Laboratory, University of California, Berkeley, Software available at `http://www-nrg.ee.lbl.gov/ns/`.

[30] Paul Francis, "Yallcast: Extending the Internet Multicast Architecture," `http://www.yallcast.com/`.

[31] Hui Zhang, "A Case for EndSystem Multicast," June 1999, Presentation at the Internet End-to-end Research Meeting.

[32] Elan Amir, Steven McCanne, and Randy Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding," in *Proceedings of ACM SIGCOMM '98*, Vancouver, British Columbia, Canada, Sept. 1998.

[33] C. Mic Borman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz, "The Harvest Information Discovery and Access System," *Computer Networks and ISDN Systems*, vol. 28, pp. 119–125, 1995.

[34] Armando Fox, Steven Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier, "Cluster-based Scalable Network Services," in *Proceedings of SOSP '97*, St. Malo, France, Oct. 1997, pp. 78–91.

[35] Elan Amir, Steven McCanne, and H. Zhang, "An Application-level Video Gateway," in *Proceedings of ACM Multimedia '95*, San Francisco, CA, Nov. 1995, pp. 255–265.

[36] Nachum Shacham, "Multipoint Communication by Hierarchically Encoded Data," in *Proceedings of IEEE Infocom '92*, 1992, pp. 2107–2114.

[37] Navin Chaddha, Gerard A. Wall, and Brian Schmidt, "An End to End Software Only Scalable Video Delivery System," in *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video*, Durham, NH, Apr. 1995, Association for Computing Machinery.

[38] Steven McCanne, Van Jacobson, and Martin Vetterli, "Receiver-driven Layered Multicast," in *Proceedings of ACM SIGCOMM '96*, Stanford, CA, Aug. 1996, pp. 117–130.

[39] Luigi Rizzo and Lorenzo Vicisano, "A Reliable Multicast Data Dristribution Protocol Based on Software FEC Techniques," in *Proceedings of HPCS '97*, Greece, June 1997.

[40] John Byers, Michael Luby, Michael Mitzenmacher, and Ashu Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," in *Proceedings of ACM SIGCOMM '98*, Vancouver, British Columbia, Canada, Sept. 1998.

[41] B. N. Levine, S. Paul, and J.J. Garcia-Luna-Aceves, "Organizing Multicast Receivers Deterministically According to Packet-Loss Correlation," in *Proceedings of ACM Multimedia '98*, Bristol, U.K., Sept. 1998.

[42] Isidor Kouvelas, Vicky Hardman, and Jon Crowcroft, "Network Adaptive Continuous-Media Applications through Self Organised Transcoding," in *Proceedings of the Network and Operating Systems Support for Digital Audio and Video*, Cambridge, U.K., July 1998.

[43] Sylvia Ratnasamy and Steven McCanne, "Scaling End-to-end Multicast Transports with a Topologically-sensitive Group Formation Protocol," in *Proceedings of the 7th International Conference on Network Protocols*, Toronto, Canada, Oct. 1999.

[44] Li-Wei H. Lehman, Stephen J. Garland, and David L. Tennenhouse, "Active Reliable Multicast," in *Proceedings of INFOCOM '98*, Mar. 1998.