A Scalable Multicast Architecture for One-to-many Telepresentations

Jim Gemmell
Microsoft Research
jgemmell@microsoft.com

Eve Schooler
California Institute of Technology
schooler@cs.caltech.edu

Roger Kermode
MIT Media Lab
woja@media.mit.edu

Abstract

We have developed a scalable reliable multicast architecture for delivering one-to-many telepresentations. In contrast to audio and video, which are often transmitted unreliably, other media, such as slides, images and animations require reliability. Our approach transmits the data in two layers. One layer is for session-persistent data, with reliability achieved by FEC alone, using the Fcast protocol. The other layer is for dynamic data, with reliability achieved using the ECSRM protocol, which combines FEC with NACK suppression. Our approach is scalable to large heterogeneous receiver sets, and supports late-joining receivers. We have implemented our approach in a multicast version of PowerPoint, a graphical slide presentation tool.

1 Introduction

A telepresentation is a presentation in which the presenter and/or some of the audience members, are not physically and temporally co-located but are *telepresent*, distributed in different locations and/or are participating at different times [11]. We believe that telepresentations have begun to revolutionize education, conferences, training, etc. by reducing associated costs and making the material available to much larger audiences. In this paper, we describe a scalable architecture for live multicast telepresentations, consisting of audio, video and presentation graphics. Presentation graphics may include data objects that are text, graphics, images, animations and special effects.

IP multicast is an excellent means of transmitting data to multiple destinations. However, it provides an unreliable datagram service, where there are no delivery guarantees. This is not an issue for some telepresentation media. For example, demonstrations of real-time audio and video transmissions regularly take place on the Multicast Backbone (MBone), a multicast-capable portion of the Internet [9]. Occasional packet loss is acceptable, both visually and audibly. If a lost packet were retransmitted, it often would arrive too late for the receiver to process the missing data. Consequently, packet audio and video typically are carried by a real-time transport protocol, such as RTP, which concerns itself more with timely delivery rather than reliable delivery. Although there are reliable protocols that exist for real-time media [16], at the present time the mainstream approach for robustness is to add forward error correction to the data stream [4]. We assume in this paper that solutions exist to transmit audio and video in telepresentations, and focus our attention on the presentation graphics.

In contrast to individual video frames, some presentation graphics may be displayed for a considerable length of time. For example, a slide with text bullet points may be displayed for several minutes. Therefore, reliable transmission is required. Furthermore, there is sufficient time to perform retransmissions. Our goal is to provide a reliable multicast solution for presentation graphics that is scalable to large audiences. As we expect the largest audiences to involve users connected via slow modems, accommodating low bandwidth connections is also an important goal.

Individual objects transmitted as part of the telepresentation graphics may persist for different timeframes. For example, a background image may be used throughout the entire presentation, while a particular graphic may only be used for a short time in the middle of the presentation. We want to ensure that resources are not wasted on objects that will no longer be used. To this end, we support three kinds of data persistence: no persistence, sliding-window persistence, and session-persistence. Our solution uses two transport protocols built on top of IP multicast to achieve these goals: Erasure-Correcting Scalable Reliable Multicast (ECSRM) and FEC-multicasting (Fcast). ECSRM is used for sliding-window persistence, while Fcast is used for session-persistent data. Non-persistent data is sent unreliably.

While ECSRM [11] and Fcast [25] are useful in many applications, in this paper we are particularly concerned with their applicability to one-to-many telepresentations. We have designed a Multicast PowerPoint prototype that utilizes ECSRM and Fcast. PowerPoint is a presentation graphics application built on a slide-show metaphor. For

live telepresentations, we have used Multicast PowerPoint in conjunction with packet audio and video transmitted via RTP. We hope to release Multicast PowerPoint on the Internet in the near future.

In the remainder of the paper, we discuss persistence issues and our implementation of Multicast PowerPoint in more detail. We present the underlying collaboration model, architectural assumptions, and the idea of data persistence. We review the properties of IP multicast, and outline the difficulties faced in building scalable reliability on top of it. We provide an overview of FEC in practice, then elaborate on how it can be integrated with multicast to handle both static and dynamic data in telepresentations. We describe the Fcast and ECSRM protocols and highlight the complementary nature of their tasks within large telepresentations. Finally, we discuss open issues for future work.

2 The problem of persistence

In a typical reliable unicast protocol, such as TCP, a data packet is cached at the sender until the receiver sends an acknowledgment (ACK) of the packet's receipt. Once acknowledged, the packet may be flushed from the cache. In a scalable reliable multicast, this simple scheme is often infeasible. If the receiver set is unknown or new receivers are allowed to join in mid-session, then it is impossible for the sender to determine when a packet may be flushed [12]. An obvious solution is to cache everything for the duration of the session. However, all session data would be treated as if it were equally useful throughout the session, even though some data may become "stale" later in the session. Maintaining cache space and utilizing transmission bandwidth on such data would be wasteful. Therefore, the problem is one of identifying the persistence of data: when is it created and for how long does it remains valid?

A popular approach in the reliable multicast research community uses Application Level Framing (ALF) [7]. With ALF, the application is responsible for recovering lost data. Data is broken down for transmission into Application Data Units (ADUs), which are self-identifying so that they may be interpreted even if received out of order. In this manner the application could recognize what ADUs have become stale, and could avoid wasting cache space or communications bandwidth on them. However, this supposes that the persistence of each ADU in the application namespace is communicated to receivers. With large, dynamic namespaces, the amount of bandwidth required to communicate this information could be prohibitive. Providing arbitrary persistence in an ADU namespace remains an open research problem, and it is not clear that it is generally useful; we believe that many applications can work well with the persistence scheme we describe below.

Instead of allowing an arbitrary level of persistence for every data object, with negligible overhead we support three levels of persistence based on the specific characteristics of our application:

- no persistence (i.e., data that may be unreliably transmitted because its usefulness is transient),
- sliding-window persistence (i.e., dynamic, but critical data that is temporarily cached), and
- session persistence (i.e., data that is used throughout the entire presentation or session).

Sliding-window persistence is achieved by assigning each packet a sequence number, and allowing the application to explicitly increase the lowest sequence number that is to remain in the cache. Therefore, a sliding window allows variations on the Least Recently Used (LRU) strategy. Our protocols have no knowledge of ADUs, but only of packet sequence numbers. However, the sliding window is most useful when it is advanced according to ADU boundaries, e.g., a slide of presentation graphics. Our Multicast PowerPoint application uses the sliding window in this fashion.

In the case of Multicast PowerPoint, the session-persistent data is known before the telepresentation begins; no new session persistent data is generated dynamically during the session. Therefore, receivers will only need this data when they first join. We send the session-persistent data on a separate multicast channel so that transmitting it to late joiners will not impact the performance of the dynamic, sliding-window data. Additionally, the static nature of the session data allows us to utilize the Fcast protocol, as explained below.

3 Architecture of Multicast PowerPoint

One of the most successful demonstrations of scalable reliable multicast to date has been the MBone whiteboard tool, wb, which uses the SRM reliable multicast framework [10]. SRM stands for the Scalable Reliable Multicast protocol, which is constructed on top of IP multicast. Because some confusion may result between SRM and scalable reliable multicast in general (i.e., reliable multicast that is scalable) we will only refer to SRM by its acronym, and will use "scalable reliable multicast" in the general sense. We will use both wb and SRM as a point of departure in this paper.

While wb is designed to allow anyone in a session to write on the whiteboard, Multicast PowerPoint is designed for one sender presenting to an extremely large audience. This distinction is intentional; there is an inherent limit to the number of users who can draw concurrently on a whiteboard or who can present material simultaneously in a telepresentation. Even if the number of senders could technically be scaled, practically and socially the number of senders is not scalable. Imagine thousands of people scribbling on a whiteboard at once! As the audience scales you cannot allow an "open floor"

in which any audience member addresses the group or presenter at any time.

To avoid chaos at a practical and social level there must be a scalable floor control mechanism that admits a limited number of senders, whether to present, write on a white board, or give feedback to the presenter. There has been some work in this area, for example, the UC Berkeley question board [17]. Having a scalable floor control mechanism allows us to trivially extend our single-sender scheme to share the session bandwidth among a dynamic, but limited number of senders. We consider this issue beyond the scope of this paper. For now, we focus on a one-to-many model for data flow, from presenter to audience, and assume that floor-control methods such as these will co-exist with the scalable transport provided by our software.

As a stand-alone application, PowerPoint is a slide preparation and presentation tool. PowerPoint slides may include text, graphics, images, etc. These elements may be animated or combined with special effects. For example, one slide may dissolve into another or text may move across a slide when the mouse is clicked.

A Multicast PowerPoint telepresentation has four components: (1) the slide master, which is the background template used by all slides and which can include images, text, default colors, etc., (2) the individual slides, (3) annotations made on the slides, and (4) control information, indicating when to change slides or to perform an animation or effect. These four kinds of information are mapped into the three persistence levels discussed above.

The slide master is persistent for the whole session, as it is needed to render any slide.

Control information is sent as non-persistent data that is piggybacked on each data packet. When no data is being sent, control information is sent in a *heartbeat* message every half second. The control information indicates the current slide of the presentation and the step or animation point within that slide. Therefore, with the receipt of the most recent control information, old control information becomes irrelevant. Thus unreliable transmission of control messages is acceptable, not only because each control message is aged and expired quite rapidly, but also because new packets constantly update the most current control information.

Every effort is made to transmit the currently-viewed slide and its annotations in a timely fashion. In addition, Multicast PowerPoint pre-sends the next anticipated slide while the current slide is being displayed. Thus, when a control message is received that advances the slide show, the new slide can be rendered immediately without delay. Also, pre-sending the next slide allows more time to recover lost packets – an important consideration with ECSRM, as we shall see below. When the presenter moves past a slide, we do not want to use up network and other resources to complete the reliable transfer of the slide or annotations, as they are no longer displayed. Therefore, sliding-window persistence is used such that the window only contains the current slides, its annotations, and the next anticipated slide. Note that if the next slide is not the one anticipated, then the pre-send can be aborted (if it has not yet completed) and the correct slide can begin to be sent immediately.

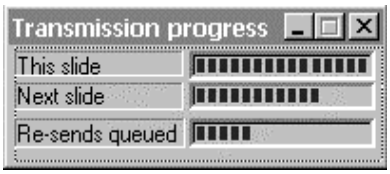

Figure 1. Transmission progress in Multicast PowerPoint.

The presenter in Multicast PowerPoint is given graphical feedback indicating how much of the current slide has been transmitted, how much of the next slide has been transmitted, and how many re-sends are currently queued to be resent (Figure 1). By giving the presenter an indication of what progress has been made by the protocol, the presentation may be paced accordingly.

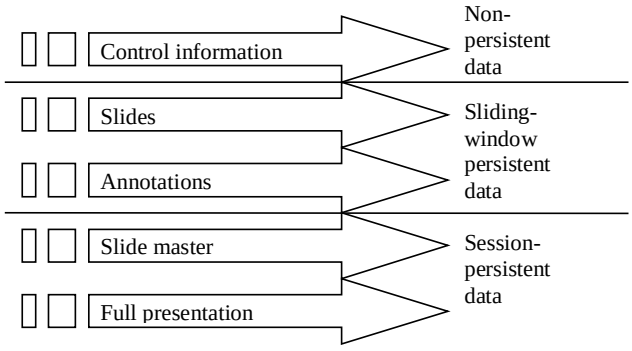

Figure 2. Logical channels of a telepresentation.

In addition to the timely delivery of the currently-viewed slide and its annotations, there may be a further goal to obtain a copy of the full presentation (the slide master plus the original slide set). We handle this via what is essentially a second session dedicated to transferring the entire presentation. However, it may be useful to logically couple this session with the live telepresentation for the purpose of cache-stuffing. That is, some receivers with poor connections may desire to tune in early and stuff their cache with as much of the presentation as possible, reducing how much they must rely on reception during the live multicast. Likewise, late joiners may wish to fill in missed slides, but opt to do so in background. The full presentation is considered session-persistent data.

Thus, we have five logical channels: control, slides, annotations, slide master, and the full presentation (Figure 2). It is possible to assign each logical channel to a different multicast address. However, to conserve multicast addresses it is also possible to combine some

logical channels on a single multicast address. For example, in our prototype, the control information, slides, and annotations are sent together.

Receivers may opt to receive all the logical channels at once, but are likely to join and leave logical channels as necessary and as a function of bandwidth availability. For instance, in Figure 3 a receiver first tunes in for the slide master. Once it is received, the receiver drops out of that transmission (which is devoted solely to repeated transmission of the slide master template) and tunes in to the slides, annotations, and control messages. After the live telepresentation completes, the receiver tunes in to the full presentation (which is dedicated to the repeated transmission of the slide master as well as original slide set) and picks up any pieces it has missed. Figure 4 depicts another scenario where the receiver tunes in to the full presentation prior to the live telepresentation. Then when the telepresentation starts, it only needs to receive control information and annotations.

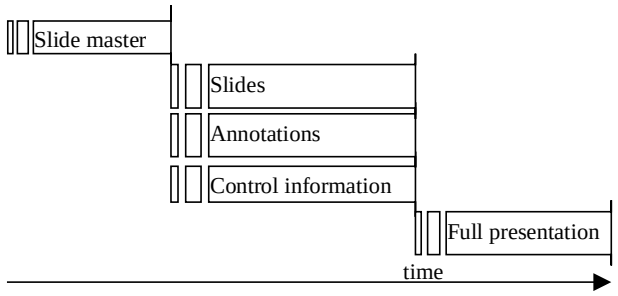

Figure 3. A receiver channel membership scenario

We assume that there exists an outside mechanism to share *session descriptions* between the sender and receiver. The session description might be located on a Web page, or conveyed via E-mail or other out-of-band methods. The session description indicates what multicast addresses are being used, when they are being used, and what kind of media is being carried over them. It also carries other information, such as the associated port number(s), data rate(s), TTL (a time-to-live or "scope" that defines how far each multicast packet can travel), type of FEC encoding, and a high-level description of the session.

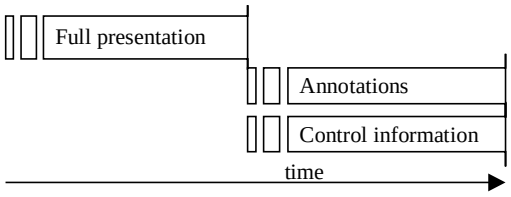

Figure 4. Another receiver channel membership scenario

Note that the data rates associated with each multicast address are likely to be different and to be tailored to the relative importance of the data in the telepresentation. For our particular application, we assume that each multicast address is rate-limited, and that receivers who cannot keep up with that rate do not join the address.

In order to pre-fetch the full presentation, a session description might indicate an earlier start time for the multicast address carrying the full presentation channel than for the multicast address(es) devoted to the live telepresentation. The full presentation multicast address may also have a completion time further in the future than the other channels, for example to accommodate late joiners who are only able to fill in missing slides after the official completion time of the live telepresentation.

In our current Multicast PowerPoint prototype, we identify each slide (ADU) simply by a number, which is compact enough to include in each packet. However, in the future we may split up the collection of slides into smaller units and refer to them by URL. In such a case, it would be necessary to communicate a cross-reference between the URL and an integer identifier used in its place during transmission. This information may be carried in the session description, or transmitted in-band.

4 Scalable reliable multicast

A number of efforts have been undertaken to provide reliability on top of IP multicast [3][6][8][10][14][21][26] [29][30]. However, some of these approaches do not scale. In designing a reliable multicast scheme that scales to arbitrarily large receiver sets, there are typically two problems. First, if the sender must keep state information for each receiver, the state can become too large to store or manage, resulting in *state explosion*. Second, there is the danger of reply messages coming back to the sender causing *message implosion*, i.e., overwhelming the sender or the network links to the sender. These reply messages may be ACKs that a packet has been successfully received or negative acknowledgments (NACKs) that indicate a packet has not been received.

State explosion can be avoided by simply keeping no state at the sender and making the receiver responsible for detecting loss. Such schemes are referred to as *receiver-reliable*, and some reliable multicast protocols adopt this approach. In order to deal with implosion, a number of techniques have been devised:

- **No Back Channel** [2][23][24]**:** In this receiver-reliable approach, redundant data is sent for loss repair. No messages are sent back in the direction from receiver to sender. In the simplest case the data is simply looped as a *data carousel* or *broadcast disk*. A more effective approach uses forward error correction (FEC) to encode packets.
- **Local Repair** [10]: NACKs and repairs are not sent to the whole group, but are kept within a restricted area. Any member of the group may perform a re-send. This method keeps losses in one topological region from impacting the rest of the group.

- **Hierarchy** [14][21][30]: Hierarchical approaches organize the receiver set into a tree, with the sender at the root and the degree of the tree limited. Each inner node is only responsible for reliable transmission to its children, which limits state explosion and message implosion and accomplishes local repairs.
- **Suppression** [22][10]**:** This receiver-reliable scheme uses delay to avoid implosions. All NACKs are multicast. When a receiver detects a lost packet, it delays ("suppresses") the NACK for a random amount of time, in hopes of receiving a NACK for the same packet from some other host. Whether it has sent or suppressed the NACK, a receiver then resets its timer for that packet and repeats the process until the packet is received.
- **Polling and Key Matching** [13][5]: All nodes generate a random key of sufficient bits so that uniqueness is extremely likely. The sender sends a polling message, which includes a key and a value to indicate the number of bits that must match between the sender's key and a receiver's key. When there is a match with the given number of bits, a receiver is allowed to request a re-transmission. The sender therefore is able to throttle the amount of traffic coming from receivers, and to obtain a random sampling of feedback.

All of these approaches have drawbacks. Data carousels can only be used when the data is static and long lived and when the receiver is willing to wait an entire loop to obtain missed data. Multicast receivers will have differing loss rates, so any FEC targeted at a particular rate is bound to be too much for some and too little for others. Hierarchies raise the question of tree management. A static tree has problems with setup, inflexibility, and maintenance. A dynamic tree may become unstable when the membership is unstable (consider viewers who "channel-surf"), and may select unsuitable interior nodes (a slow processor behind a slow modem). Local repairs are only marginally helpful when losses are near the sender. Both hierarchical and local repair approaches need to map to network topology, but have only the very crude use of TTL at their disposal. Suppression essentially trades off delay for scalability. At a certain point the delay becomes so long as to render it useless. Polling also trades delay for scalability.

Naturally, hybrids are possible. Local repairs are often combined with hierarchy. Polling may be used as a means to tune delay in a suppression scheme. Multicast PowerPoint supports scalable reliable multicast via a hybrid of FEC with other techniques. For dynamic session data, it uses ECSRM, which combines FEC with suppression. For session-persistent data, it relies on Fcast, combining FEC with a data carousel.

5 FEC: (n,k) linear block encoding

Most of the FEC literature deals with error correction, that is, the ability to detect and repair with both erasures (losses) and bit-level corruption. However, in the case of IP multicast, lower network layers will detect corrupted packets and discard them. Therefore, an IP multicast application need not be concerned with corruption; it can focus on erasure correction only.

The form of erasure correction (EC) utilized in the Multicast PowerPoint prototype is known as (*n*,*k*) linear block code. *k* source packets are encoded into *n*>*k* packets, such that any *k* of the encoded packets can be used to reconstruct the original *k* packets (Figure 5). For example, parity can be used to implement (*k+1*, *k*) encoding. (*n*,*k*) encoding and decoding algorithms have been developed that are efficient enough to be used by general purpose personal computers [1][24][15]. For example, Rizzo's public domain codec has been shown to operate at rates of over 11 MB/s on a 133 MHz Pentium and at rates over 350 KB/s on a 25 MHz 386 [24].

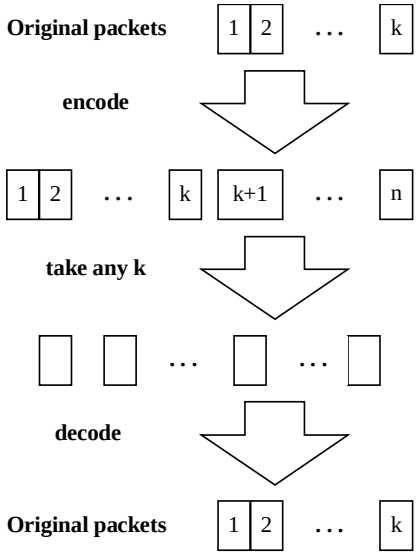

Figure 5. (n,k) encoding and decoding: k original packets are reconstructed from any k encoded packets.

In practice, *k* cannot be arbitrarily large. For example, typical *k* values are 16 and 32, and *n* is generally less than 255 [24]. The basic EC unit is a *block* and a typical block size is 1024 bytes. We use the terms block and packet interchangeably, because the transmission payload is one and the same as an EC block. Each packet in the session must be assigned into an EC group of *k* packets, which may then be encoded into *n* packets. Each packet is identified by an *index* specifying which of the *n* encoded packets it is, as well as a *group* identifier associating it with an EC group.

Some encoding schemes, including the one we have chosen for our prototype, simplify matters by making the first *k* of the *n* encoded blocks be copies of the original blocks. Therefore, if no packets are lost, a receiver does not incur any processing overhead decoding the *k* packets. Another nice property of FEC encoding is that encoded packets are approximately the same size as original packets. The only overhead introduced in the packet header is the need to indicate the index and group (40 bits in our prototype).

6 Persistent session data

For the persistent session data, we employ Fcast [25]. The Fcast protocol combines (n,k) encoding with a data carousel; its use of (n,k) encoding reduces the time taken for a receiver to receive a missing packet, as compared to a plain data carousel. In the Multicast PowerPoint context, Fcast is used for the logical channel containing the slide master, as well as the channel meant for the full presentation. That is, the channels containing only static session-persistent data. When a receiver joins such a channel, Fcast is able to deliver the data to it in near-optimal time, as we shall see below. Because it uses no back-channel, it is as scalable as IP Multicast.

In most regards, the Fcast protocol is a straightforward adaptation of the algorithms presented in [23][27]. Fcast has network and codec performance essentially like the one-layer case in [23]. However, there are two notable aspects about its implementation: its ability to multiplex any number of static objects (e.g., files or slides) onto a single channel, and its treatment of memory versus disk storage.

6.1 Multiplexing static objects

The transmission order of the blocks from an EC group is important. The receiver must obtain *k* distinct blocks out of the *n* blocks to reconstruct the original *k* blocks. Any duplicates that are received must be discarded. Therefore, we do not want to repeat transmission of any block until all the others have been sent. Furthermore, the ordering among the groups is important. If all *n* packets for a group were to be sent at once, this could create a long delay for a receiver waiting for a packet from some other group. Therefore a packet is sent from each group in turn.

Because *k* and *n* are values constrained by software complexity, an object of size *N* blocks is partitioned into *G=N/k* groups. Thus, the typical transmission order for an *N* block object, with *G* groups might be as suggested by [23] and shown in Figure 6: block 0 from each group, block 1 from each group, … block n-1 from each group. When the last packet of the last group is sent, the next transmission cycle begins again with block 0. Note that the darkly shaded area represents the original object of size *N=Gk* and the transmission order is such that original blocks (indices < *k*) are sent before any encoded blocks.

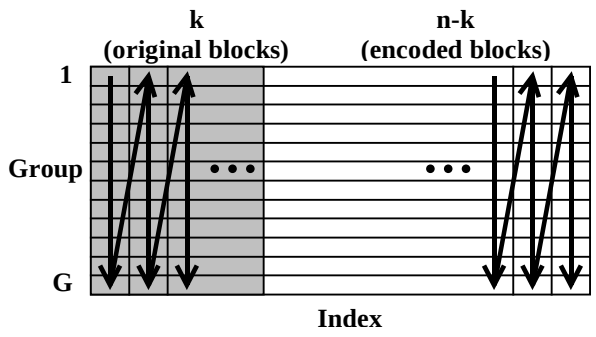

Figure 6. Transmission order: Packet 1 of each group is sent, then packet 2, etc.

Suppose a receiver gets to the point where it only needs one packet from a certain group to have a complete transmission. Then, in the worst case, it may require the receipt of *G* additional packets. In other words, the receiver may have to receive a packet from each of the other groups before getting a packet from the desired group. Suppose now that somehow losses were correlated so that *k* packets were received from all groups but one, and no packets from that one group. Then at least *kG* more packets would need to be received. However, such correlation is unlikely unless some periodic failure occurs in the network. Even so, correlation may be avoided by a number of techniques, including randomly perturbing the group order in each cycle. An alternate approach (that we do not use) is to take advantage of correlation and to split FEC onto different channels [15]. For more detailed analysis, we refer the reader to [23] and simply point out that without correlation the number of unnecessary packets received is quite low. Furthermore, while a packet may be unnecessary to one given receiver it is likely to be useful to some other receiver within the session.

Fcast can be used to convey multiple objects. This is useful when we combine multiple objects onto a single channel, as in the case of the channel for the full presentation, consisting of each slide including the master. Each object is split into groups of size *k*. Every packet header then identifies its contents by object ID, group number, and index. The packet transmission order for multiplexed objects is as follows. The packet ordering begins with block 0 of the first group of the first object. The sender slices the object along block indices, then steps through index *i* for all groups within all objects before sending blocks with index *i+1*. When block *n* of the last group of the last object is sent, the transmission cycles. Figure 7 shows an example for three objects, where the first object contains *G* groups, the second object *G'* groups, and the third object *G''* groups.

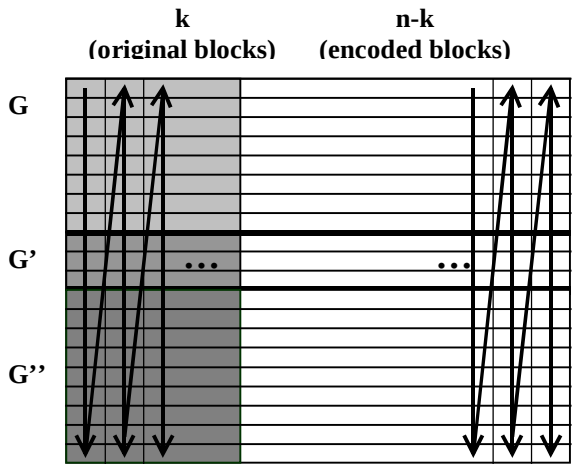

Figure 7. Transmission order for multiple objects

The multiplexing capability of Fcast allows for flexible coupling and de-coupling of logical channels. This may prove to be especially useful when the Multicast PowerPoint prototype begins supporting live Web links, which would need to be multicast along with other live telepresentation data. Each link might be considered its own static slide, which very likely will point to still other links that should be treated as static slides. Fcast will be able to multiplex the content of several links onto a single multicast channel. Presently, the coherent indexing scheme allows a receiver to avoid processing redundant information, by comparing the indices of an incoming packet (object, group, index) with those received across the aggregate of the channels. For instance, a late joiner may only have had time to subscribe to the slide master channel, followed by the live telepresentation channel(s). Afterwards, when the receiver chooses to tune into the full presentation, it will be able to detect and ignore already-received packets.

6.2 Memory versus disk storage

As shown in Figure 7, Fcast sends both original and encoded blocks. To construct an encoded block, *k* original blocks must be processed in memory. A number of caching options are possible:

- Cache original blocks in memory to avoid repeated disk access
- Cache encoded blocks in memory to avoid repeated computation
- Cache encoded blocks on disk to avoid repeated computation and disk access of originals

Computation is unlikely to be the bottleneck. A typical CPU (133 MHz Pentium) can encode at 88 Mb/s, which is faster than the typical LAN (10 Mb/s Ethernet), and certainly faster than typical Internet access. Additionally, it is common for there to be many more encoded blocks than originals, e.g., k=32, n=255. Therefore, in-memory caching of encoded blocks is unlikely to be worthwhile.

Network connection	Transmission Rate	Required Disk Rate, k=32
Modem	56 Kb/s	1.75 Mb/s
ISDN	128 Kb/s	4 Mb/s
xDSL/Cable Modem	1 Mb/s	32 Mb/s
Ethernet	2 Mb/s	64 Mb/s
Fast Ethernet	4 Mb/s	128 Mb/s
Fast Ethernet	10 Mb/s	320 Mb/s

Table 1. Required disk throughput rate (sustained) for sender assuming no caching.

However, disk access could become a bottleneck, as *k* original blocks are required to create each encoded block. Storing only originals on disk, with no caching of any kind, requires that the disk have a sustained rate that is *k* times faster than the network transmission speed. Table 1 gives examples of disk rates required for various transmission speeds. Note that performance in an Ethernet begins to degrade at about 30% utilization, so we consider 4 and 10 Mb/s to be in the domain of a Fast (100 Mb/s) Ethernet.

We assume that a typical disk can sustain > 0.5 MB/s, so supporting modem and ISDN connections should not be a problem. However, for higher speed connections the disk may well be the bottleneck. For such scenarios, either encoded blocks could be cached on disk (requiring extra disk space), or the original blocks should be cached in memory. We are primarily concerned with large Internet audiences with low speed modems, so no caching is necessary. To simultaneously support both high and low rate receivers, a layered approach may be adopted (see remarks in the Conclusion).

The Fcast receiver has a more complicated task. Blocks may or may not arrive in the order sent, portions of the data stream may be missing, and redundant blocks will need to be ignored. Because the receiver is designed to reconstruct the file(s) regardless of the sender's block transmission order, the receiver does not care to what extent the block receipt is out of order, or if there are gaps in the sender's data stream. As each block is received, the receiver tests:

- Does the block belong to the Fcast session?
- Has the block not been received yet?
- Is the block for a file that is still incomplete?
- Is the block for a group that is still incomplete (a group is complete when *k* distinct blocks are received)?

If a block does not pass these tests, it is ignored. Otherwise, it is written immediately to disk. It is not stored in memory because its neighboring blocks are not sent contiguously, and even if they were, they might not arrive that way or at all. The receiver keeps track of how many blocks have been received so far for each group and what the block index values are. The index values are

needed by the FEC decode routine. When the new block is written to disk, it is placed in its rightful group within the file (i.e., the group beginning at location *k*blocksize*group*). But, it is placed in the *next available block position within the group*, which may not be its final location within the file. Once the receiver receives *k* blocks for a group, the entire group of blocks is read back into memory, the FEC decode operation is performed on them if necessary, and the decoded group of blocks is written back out to disk (beginning at the same location) with all blocks placed in their proper place. Consequently, the Fcast disk storage requirements are equal to the file size of the transmitted file(s).

In order to write each block to disk as it is received, disk throughput at the receiver is required to be at least that of the transmission rate. At ISDN rates, this should not be a problem, but at higher rates supporting such random writes may be difficult. Also, decoding requires *k* blocks must be read, then decoded, and then the *k* decoded blocks must be written to disk. If this is to be done while blocks are being received disk throughput must be about *2k* times faster than the receive rate. Often this may not be the case, so decoding should be done at a lower priority than writing received blocks. If no time is available for decoding, it should be deferred until after all necessary blocks have been received.

7 Dynamic session data

For the dynamic session data, we employ ECSRM [11]. The ECSRM protocol combines NACK suppression with erasure correction. In Multicast PowerPoint, dynamic session data consists of the current slide, its annotations, and the anticipated next slide. Fcast is inappropriate because a data carousel of dynamically changing data is extremely awkward to manage, and because Fcast is targeted towards file transfers, where ECSRM works in-memory to support small objects like annotations. Furthermore, Fcast relies on the data being static so that it can order transmission across groups.

In the remainder of this section, we present ECSRM and related work.

7.1 ECSRM

Our implementation of ECSRM allows a window of persistence to be dynamically set. Each packet is assigned a sequence number. A low water mark is then updated during the session, indicating the lowest sequence number that may be requested for retransmission. Each packet's header updates the low water mark.

ECSRM uses suppression of NACKs to avoid implosion. NACKs are scheduled when a gap is detected in the sequence numbers of the received packets. If a host has a NACK scheduled and receives the same NACK from another host before the timer expires, it will suppress its NACK. That is, the NACK is not sent, but the timer is reset as if it had been sent. The original delay before sending a NACK is set as a random value in the range [MinD, MaxD], where MinD and MaxD are tunable parameters. Subsequent timer delays are increased using exponential random back-off. That is, the *i*'th time a receiver sets its timer for the receipt of a particular packet, it will set the timer in the range 2^{i-1}[MinD, MaxD].

One of our design goals is to work well with slow modems, so rate control is a concern. Unlike SRM, described below, we only allow the sender to issue repairs. If repairs were issued from other receivers, then the application would risk generating uncontrolled spikes in bandwidth usage, which might cause problems for modem connections. Consequently, ECSRM performs rate control on all traffic from the sender, on original traffic as well as retransmissions. NACKs may still create spikes in bandwidth usage, but NACKs are small and suppression is employed, so a small slice of bandwidth reserved for NACKs should suffice to solve this problem.

ECSRM does not respond to a NACK by re-sending a lost packet, as would other suppression schemes. Instead ECSRM responds by sending an erasure-correcting packet. As with Fcast, packets are assigned to EC groups of size *k*, where *k* is a parameter set by the application for the session. However, because ECSRM deals with dynamic data, it cannot transmit one packet from each group in turn. Instead, packets are assigned to the most recent group as they are generated. When the current group is filled with *k* original packets, a new group is started. An (*n*,*k*) erasure correcting code is then applied to generate as many as *n* packets for each group, as needed only, where the first *k* are the original packets. The *k* original packets may be sent immediately without waiting for the group to fill, or performing any encoding. If a receiver obtains all *k* packets from a group they may be immediately used without any decoding.

Receivers keep track of the number of lost packets per EC group. NACK messages specify the number of packets lost from a particular EC group. In order to compactly represent burst errors, NACK ranges are sent of the form {*1stGroup, 1stCount, LastGroup, LastCount*}. This indicates that *1stCount* packets were lost from group *1stGroup, LastCount* packets were lost from group *LastGroup,* and all packets were lost from any group *g,* such that *1stGroup* < *g* < *LastGroup.* The basic idea behind suppression is that a receiver suppresses its NACK if it hears a NACK from another receiver that specifies the same EC group and a lost packet count at least as high. However, in practice suppression is based on ranges, so a range is suppressed only when another range is heard that is a superset of the original range.

NACKing based on group losses rather than individual loss provides a reduction in both NACK traffic and in responses to NACK traffic. NACK traffic is reduced because NACK suppression is based on the number of

losses per group rather than on individual packets. The traffic in response to NACKs is reduced because an erasure-correcting packet can replace any given lost packet from a group.

Whenever a NACK is sent, it incurs header overhead from IP, UDP and the SRM protocol, amounting to at least 40 bytes. With 32 bits used to represent a group and 8 bits a count, a NACK range uses 10 bytes. Therefore, a receiver makes more efficient use of bandwidth if it sends multiple NACK ranges at a time. At any time there may be a number of NACK ranges waiting for their timer to expire. When the first timer expires, additional NACKs are included ("piggy-backed") in the packet, even though their timers haven't expired. This affords further suppression and increased timeliness at low cost in terms of bandwidth.

When the sender receives a NACK range from which it can infer that *count* packets were lost from group *g*, *count* FEC packets are normally sent in response. However, some re-send suppression is needed to prevent duplicate NACKs from generating too many responses. With SRM, NACKs are ignored for a certain interval after a NACK is received. With ECSRM, each group retains a record of the time at which the last *k* packets were sent from the group (including EC packets). When a NACK for *count* packets is received, the last *count* time values are inspected. For each of them that is within the suppression threshold, *count* is decremented by one. For example, if a NACK for 3 packets was received, and 3 EC packets sent, and then a NACK for 5 packets is received within the threshold period, then the count would be suppressed down to 2, and only 2 EC packets would be sent. For each FEC group, a counter is kept of the last FEC packet sent. When all *n* packets have been sent, the counter wraps around and starts over with the first packet.

The above assumes that the group of packets being NACKed is full (contains *k* packets). However, that may not be the case. For example, suppose the sender starts a session, sends one packet, and then does not transmit any more data for a long time. Receivers who have lost the one packet may then detect the loss via the sender's heartbeat packets and NACK a loss of one packet from a group. In this example, the group is not yet full; it contains only one packet. In the case where a NACK is received and the EC group is not yet full, ECSRM will wait for a short time, in the hope that the group will fill. If it does not fill after that time, then original packets are resent. Naturally, re-sending originals should be avoided. To this end, the EC group size, *k*, is kept relatively small. ECSRM allows the value to be set as a parameter of the session, and the choice of *k* will depend on the application.

Our implementation of ECSRM allows packets of differing size to be sent in a single session. This implies that a decoded packet may not have the same length as the received packets used to produce it. In order to deal with this, we include a field in the header indicating the packet size. When encoding, the payload and the length are encoded, but the header is not, as the header must be able to be processed without any decoding (the header is needed to do the decoding). For Multicast PowerPoint, we attempt to keep all packets the same size. However, when fragmenting a slide, the last packet for the slide may be small. Such packets, along with annotation packets, are the only packets of a differing size.

ECSRM optionally allows EC packets to be sent when the channel is idle. When the session is initialized, a rate limit must be specified. When the full rate is not utilized, then extra EC packets may be sent, even though no NACKs have requested them. When the channel is idle, the sender cycles through the cached (sliding-window persistent) FEC groups and sends the next EC packet for each group. There are two benefits to this approach. First, it may replenish lost packets that have not yet been NACKed, further suppressing NACKs. Second, it allows receivers with no multicast back channel for NACKs to have losses corrected (e.g., UUNET multicasts to modem dial-up customers, but does not allow them to multicast), given enough idle channel time in the session. In the latter case, ECSRM operates very much like an Fcast with a dynamically changing carousel.

The scalability benefits from ECSRM are most apparent with a large receiver set experiencing low-level uncorrelated loss. Consider 1,000,000 members, and a random, independent, loss of 0.01%. The chance of all nodes receiving a given packet is $0.9999^{1,000,000} < 10^{-43}$. So it is a virtual certainty that each packet will be lost by some receiver. Without any erasure correction, each packet must be sent at least twice, cutting the effective bandwidth in half. Also, a NACK would need to be sent for each packet. Suppose, now, that ECSRM is used with a group size, *k*, of 7. A single EC packet can correct the loss of any single packet in the group, and correct the kind of low-rate loss described above. This means a loss of only 1/8 of effective bandwidth, compared to a loss of ½ of effective bandwidth without erasure correction. Furthermore, the NACK traffic would be reduced by a factor of 7, as a NACK for a single packet from each group will suppress all other NACKs for a lost packet in the group. In contrast, a NACK is needed for each packet without erasure correction. Of course, re-sends and NACKs may also be lost, but for the purpose of simple comparison we ignore that loss here.

The above example considers independent loss. With correlated loss, using erasure correction does not provide any benefit over just re-sending original packets. However, it does not use any more network resources (only CPU overhead for decode). Furthermore, for losses to be totally correlated implies that the loss occurs very near the sender. Such loss must reduce *any* scheme to retransmission of the same number of packets. At any rate, as the receiver set scales, the number of links and

sub-networks involved must increase, so some measure of independence is to be expected.

7.2 Other work related to ECSRM

SRM [10] uses suppression of NACKs and suppression of re-sends to avoid implosion. Any host with the data required by a NACK may re-send the packet. A goal in SRM is to increase the probability that a host close to the point of loss is most likely to perform the re-send. However, (1) this feature is only probabilistic, not guaranteed, (2) the node nearest the point of loss may not really be suitable – it may be under-powered or on a slow or congested link, (3) we assume that applications using NACK suppression can afford to trade off time for scalability – so fast re-sends are not so critical, and (4) in the case of live telepresentations, continued connectivity to the sender is critical. Additional fault tolerance from this feature is not important. Additionally, SRM lacks control over the rate of incoming data to a given receiver, due to its distributed nature. At a given time several nodes may be sending NACKs, several may be re-sending data, and the sender may be sending new data. This may lead to an aggregate traffic level that is higher than some nodes can handle.

Integration of FEC and reliability was first proposed by Metzner [19]. Another approach that combines NACK suppression with FEC is Protocol NP [20]. Protocol NP proceeds in rounds. In each round the sender sends its data, polls receivers for the number of missing packets, and then sends that number of EC packets. It can carry on with sending new data from the next round, but will interrupt this to send EC packets requested from previous rounds. Receivers perform NACK suppression. Like ECSRM, a receiver does not NACK a particular packet, but rather indicates how many packets from a particular EC group have been missed.

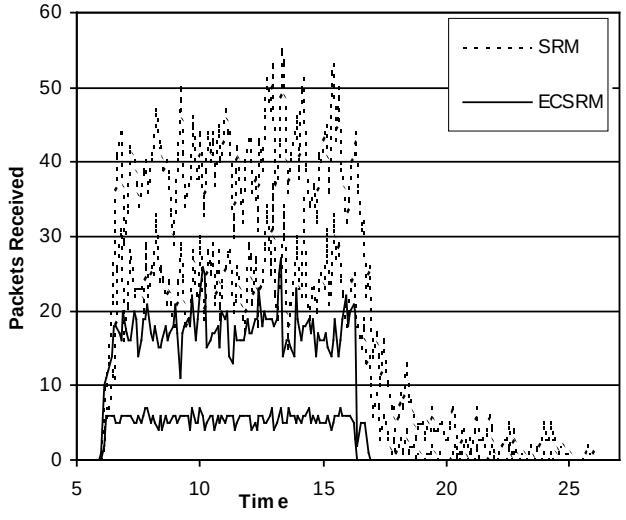

Figure 8. Minimum and maximum data packets received by receivers for SRM and ECSRM (k=16). Simulation results for 112 receivers and loss rates from 2% to 28.3%

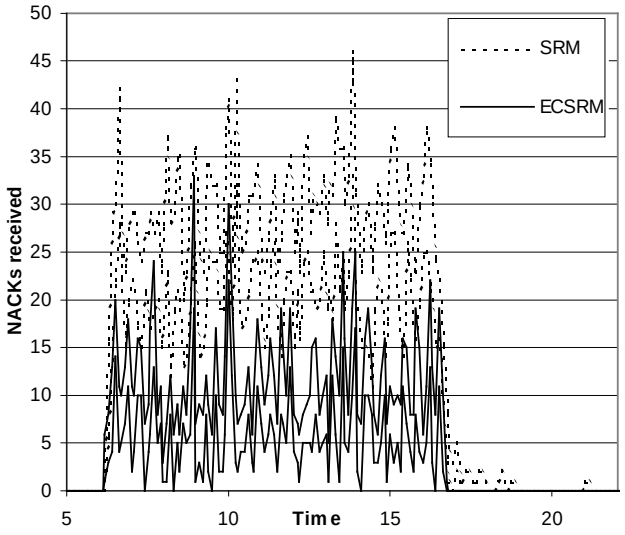

Figure 9. Minimum and maximum NACKs received by receivers for SRM and ECSRM (k=16). Simulation results for 112 receivers and loss rates from 2% to 28.3%

Floyd et al. include simulation results for SRM performance [10]. The performance gains yielded by integrated FEC with NACK suppression are described in detail by Nonnenmacher et. al. [20]. They show how adding FEC produces performance gains, and how integration of FEC into the protocol results in even more dramatic gains. Figure 8 and Figure 9 show simulation results for SRM and ECSRM with 112 receivers, k=16, and receiver loss rates from 2% to 28.3%. Traffic observed at each receiver was recorded, and the minimum and maximum plotted in these graphs. Figure 8 shows data packet traffic and Figure 9 shows NACK traffic. Note that ECSRM generates much less traffic, and completes much sooner. For more details about this particular simulation and related simulations, see [16].

8 Conclusion

We have presented a scalable multicast architecture for live telepresentations, as exemplified by Multicast PowerPoint. Our approach splits the telepresentation into its underlying components: the slide master; individual slides; real-time annotations; and control information for advancing the presentation and triggering graphic effects. These objects are considered logical channels of information. In addition, we provide a logical channel for the full presentation – a combination of the slide master and the original slide set. The full presentation is often useful to receivers because it may be pre-fetched to reduce bandwidth requirements during the session, or it

may be requested during or after the session by late joiners. The logical channels are mapped, either separately or in combination, to multiple multicast addresses, which may be individually selected by receivers. Receivers benefit by this approach because they are able to tailor their needs based on their bandwidth capabilities and on when they join the telepresentation.

Our design supports different levels of object persistence: no persistence, sliding-window persistence, and session persistence. Different levels of persistence require different transmission strategies. Non-persistent data, such as control information, is transmitted using unreliable IP multicast. We have designed ECSRM to support sliding-window persistence. ECSRM combines NACK suppression with erasure correction, improving on the scaling properties and rate control of other protocols. Thus, dynamic session data, such as the currently-viewed slide, its annotations, and the next anticipated slide, are handled by ECSRM. Session persistent data, such as the slide master, slide content and the full presentation, are handled by Fcast, which uses erasure correction in a data carousel. The combined use of ECSRM and Fcast allows Multicast PowerPoint to support extremely large heterogeneous receiver sets, yet still accommodate late session joiners.

The most obvious question raised by Multicast PowerPoint is whether two protocols are, in fact, needed. One approach would be to use only ECSRM. Use of ECSRM for session-persistent data may marginally improve best-case throughput (Fcast performs well already in the worst case), but would introduce back-traffic and reduce scalability. A more attractive option would be to use only Fcast, so that no back-traffic is required and scalability is enhanced. However, it is not clear how to adapt Fcast to dynamic data, and even if it could be thus adapted, its performance would need to be studied to understand in what scenarios it could replace ECSRM. Devising dynamic schemes for Fcast and evaluating their performance appears to be a fruitful area of future research.

There are a number of other interesting areas to explore. We would like to divide up slides into smaller components: text, graphics, etc. Then, even if part of a slide has been lost, the remainder could still be rendered quickly. Additionally, we would like to examine layered transmission**,** where data is split up across a number of multicast addresses. Multiple layers have been used, for example, in video applications [18]. The idea is that if the number of addresses subscribed to by the receiver increases, then the overall data quality increases. Joining and leaving layers allows receivers to throttle the received data rate, and therefore may be used for multicast congestion control, which is an important open research question. Layering has already been shown to be very effective for an Fcast type scenario [27].

A future direction for ECSRM is to dynamically adjust suppression times based on the observation of NACK traffic. This could be accomplished independently by receivers or via a polling and key matching method. Statistics also could be collected from the receivers, similar to RTCP.

At present, our prototype requires all parties to have PowerPoint. In the future, we would like to transmit the slides as HTML so that any browser may be a receiver. We plan to release our prototype on the Internet in the near future.

Acknowledgments

This work was supported in part by the Air Force Office of Scientific Research under grant AFOSR F49620-97-1-0267 and a Microsoft Graduate Fellowship. Thanks to Jim Gray, Paul Sivilotti, and the reviewers for their helpful comments.

References

[1] N. Alon, M. Luby, A Linear Time Erasure-Resilient Code With Nearly Optimal Recovery. *IEEE Transactions on Information Theory*, **42**:6 (Nov 1996) 1732-1736.

[2] S. Achera, M. Franklin, S. Zdonik, Dissemination-Based Data Delivery Using Broadcast Disks. *IEEE Personal Communications*, (Dec 1995), 50-60.

[3] K. Birman, A. Schiper, P. Stephenson, Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, **9**:3 (Aug 1991), 272-314.

[4] J.C. Bolot, H, Crepin, A. Vega Garcia, Analysis of Audio Packet Loss in the Internet. *Proceedings of 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, (Apr 1995) 163-174, Durham, New Hampshire.

[5] J.C. Bolot, T. Turletti, I. Wakeman, Scalable Feedback Control for Multicast Video Distribution in the Internet. *Proceedings of ACM SIGCOMM'94*, (Oct 1994) 58-67 London, England.

[6] J. M. Chang, N. F. Maxemchuck, Reliable Broadcast Protocols. *ACM Transactions on Computing Systems*, **2**:3 (Aug 1984) 251-273.

[7] D.D. Clark, D.L. Tennenhouse, Architectural Considerations for a New Generation of Protocols. *Proceedings of ACM SIGCOMM '90*, (Sept 1990) 201-208 Philadelphia, PA.

[8] J. Crowcroft, K. Paliwoda, A Multicast Transport Protocol. *Proceedings of ACM SIGCOMM '88*, (1988) 247-256, Stanford, CA.

[9] H. Erikson, MBONE: The Multicast Backbone. *Communications of the ACM*, **37**:8 (Aug 1994) 54-60.

[10] S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang, A Reliable Multicast Framework for Light-weight Sessions

and Application Level Framing. *ACM SIGCOMM '95*, (Aug 1995) 342-356 Cambridge, MA.

[11] J. Gemmell, Scalable Reliable Multicast Using Erasure-Correcting Re-sends. Technical Report, MSR-TR-97-20, Microsoft Research, Redmond, WA (June 1997).

[12] J. Gemmell, J. Liebeherr, D. Bassett, An API for Scalable Reliable Multicast. *International Conference on Computer Communications and Networks*, (Sept 1997) 60-64 Las Vegas, NV.

[13] M. Handley, J. Crowcroft, Network Text Editor (NTE): A Scalable Shared Text Editor for the Mbone. *Proceedings of ACM SIGCOMM'97*, (Aug 1997) 197-208 Canne, France.

[14] H.W. Holbrook, S.K. Singhal, D.R. Cheriton, D.R. Log-based Receiver-Reliable Multicast for Distributed Interactive Simulation. *Proceedings of SIGCOMM '95*, (Aug 1995) 328-341 Cambridge, MA.

[15] S.K. Kasera, J. Kurose, D. Towsley Scalable, Reliable Multicast Using Multiple Multicast Groups. *Proceedings of ACM SIGMETRICS '97*, (1997) 64-74 Seattle, WA.

[16] R. Kermode, Smart Network Caches: Localized Content and Application Negotiated Recovery Mechanisms for Multicast Media Distribution, *PhD Dissertation*, MIT, 1998.

[17] R. Malpani, L.A. Rowe Floor Control for Large-Scale MBONE Seminars. *Proceedings of The Fifth Annual ACM International Multimedia Conference*, (Nov 1997) 155-163 Seattle, WA.

[18] S. McCanne, M. Vetterli, V. Jacobson, Low-complexity Video Coding for Receiver-driven Layered Multicast. *IEEE Journal on Selected Areas in Communications*, **16**:6 (Aug 1997) 983-1001.

[19] J. Metzner, An Improved Broadcast Retransmission Protocol. *IEEE Transactions on Communications*, **32**:6 (Jun 1984) 679-683.

[20] J. Nonnenmacher, E. Biersack, J. Towsley Parity-Based Loss Recovery for Reliable Multicast Transmission. Technical Report, TR-97-17, Department of Computer Science, University of Massachusetts, Mar 1997.

[21] S. Paul, K.K. Sabnani, J.C.-H. Lin, S. Bhattacharyya Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, **15**:3 (Apr 1997) 407-421.

[22] S. Ramakrishnan, B.N. Jain, A Negative Acknowledgement With Periodic Polling Protocol for Multicast over LANs. *Proceedings of IEEE Infocom '87*, (Mar/Apr 1987) 502-511.

[23] L. Rizzo, L. Vicisano, A Reliable Multicast Data Distribution protocol based on software FEC techniques. *Proceedings of the Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems, HPCS'97*, (June 1997) Chalkidiki, Greece.

[24] L. Rizzo, Effective Erasure Codes for Reliable Computer Communication Protocols. *Computer Communication Review*, **27**:2 (Apr 1997a) 24-36.

[25] E. Schooler, J. Gemmell, Using Multicast FEC to Solve the Midnight Madness Problem. Technical Report MSR-TR-97-25, Microsoft Research, Redmond, WA (Sept 1997).

[26] R. Talpade, M.H. Ammar, Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service. *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*, (June 1995) 144-151 Vancouver, Canada.

[27] L. Vicisano, J. Crowcroft, One to Many Reliable Bulk-Data Transfer in the Mbone. *Proceedings of the Third International Workshop on High Performance Protocol Architectures, HIPPARCH '97*, (June 1997) Uppsala, Sweden.

[28] B. Whetten, T. Montgomery, S. Kaplan, A High Performance Totally Ordered Multicast Protocol. *Proceedings of the International Workshop on Theory and Practice in Distributed Systems*, (Sept 1994) Springer-Verlag, 33-57.

[29] R. Yavatkar, J. Griffioen, M. Sudan, A Reliable Dissemination Protocol for Interactive Collaborative Applications. *ACM Multimedia 95*, (Nov 1995) 333-343 San Francisco, CA.