A Cooperative Packet Recovery Protocol for Multicast Video

N. F. Maxemchuk
AT&T Labs - Research
Florham Park, N.J.

K. Padmanabhan
Fujitsu Labs

S. Lo
Lucent Tech - Bell Labs
Murray Hill, N.J.

Abstract

In this paper we describe a protocol that improves the quality of multicast video or audio transmissions by recovering lost packets. The protocol is applied to a corporate intranet, and measurements indicate that significant improvements in audio/video quality are achieved.

A unique characteristic of this protocol is that it can be used on current MBone sessions without changing the operation of the source or receivers. The improvement in quality can be restricted to a subset of the receivers, so that a service provider, that does not necessarily generate content, can use the protocol to improve reception for its own customers. A second unique characteristic is that only a small fraction of the receivers that obtain the benefits of the protocol are active partipants. As a result the problems associated with large multicast groups, such as the number of control messages, is reduced. Finally, because of the differences between data and voice or video communications, many of the network parameters that vary between or during connections are changed rather than predicted. This is a much simpler operation.

1: Introduction

The Internet Multicast Backbone (MBone) has been used to distribute video and audio coverage of selected sessions in technical conferences[1]. As more members of the general population use the Internet, the MBone is increasingly being used to cover news events, for distance learning, and for business meetings. Low frame rate video and high quality audio are transmitted to widely distributed locations and are received on conventional desktop or home computers. At times the quality of the received signal is acceptable, and at other times it is unusable. The quality must be significantly improved before the potential for this medium can be fully realized.

Lost packets constitute one of the chief causes of degradation of MBone video. In the experiments reported here, losses exceeding 15% are common. There are significant differences in the delay requirements of multimedia applications [2] that make it possible to retransmit lost packets in some applications. For instance, a few seconds of delay are not noticeable in a real-time news report, but would be intolerable in an interactive voice conversation. Several techniques have been proposed to recover lost packets in video systems [3,4,5]. In schemes that require retransmissions there must be sufficient storage at receivers to store the signal while waiting to recover missing packets. In video applications storage has been an important concern, however, as explained in section 2.4, storage is not a major concern in MBone video.

Most loss recovery mechanisms are designed for data. However, there are significant differences between the signal characteristics and error recovery requirements of video and data. The first difference is that the packets from video sources are transmitted continuously while those from data sources occur in bursts that may not resume without a response from the receiver. As a result, in a video application lost packets are detected when the next packet arrives. It is not necessary to acknowledge each packet. The second difference is that video and audio signals can tolerate a limited amount of distortion introduced by some missing or erroneous packets, while most data applications require near perfect reception. As a result, the objective in video transmission is to recover the lost packets that are easiest to recover.

The different requirements for video error recovery make it possible to design simpler protocols, with fewer control messages, than the protocols used for data. A unique characteristic of our protocol is that it can operate on existing MBone transmissions without changing the source or receivers. The source and receivers use any conventional multicast protocol. Recovery occurs between agents in the network. The agents generate a repaired multicast signal that is received on a separate channel from the source's signal.

The remainder of this paper is divided into three parts. The protocol is described in section 2. In this work we concentrate on a simple implementation of the protocol, which is adequate for the numbers and distribution of users in most current applications. However, we also describe several variations that make it possible to extend the protocol to arbitrarily large groups. In section 3 we describe an implementation that we have used on a corporate intranet to distribute colloquia and meetings between different locations in the United States. The emphasis in this section is on practical concerns of integrating our protocol with existing software. In section 4 we report the results of using the protocol. We measure the packets acquired by adjacent receivers, one which receives directly from the source and another which receives the repaired channel. It is difficult to describe the subjective effect of the recover mechanism on the intelligibility of voice and the information conveyed in the images, however, the improvements in reception rate clearly illustrates the improvement. For instance, a receiver that acquires only 70% of the packets directly from the source acquires over 95% of the packets on the repaired channel.

2: The Protocol

Our protocol is a cooperative arrangement between a subset of the receivers in the network. The objective is to recovery packets that have been missed by some, but not all, of the cooperating receivers. In the simplest implementation, one of the receivers is responsible for retransmitting packets that have been missed by the others. This *retransmit server* is the receiver that is least likely to miss packets. In an intranet that is distributed between several locations, the retransmit server is located on the same local network as the source. If the retransmit server is at the source it has all of the packets, otherwise, it only has a subset of the packets and cannot respond to every retransmission request.

The basic protocol is simple. The packets transmitted by a source contain sequence numbers. When a network receiver, other than the retransmit server, receives a packet with a greater sequence number than expected, the missing packets are requested from the retransmit server. The request is repeated if the retransmission isn't received. After a fixed interval of time the network receiver, *repair server*, retransmits all of the packets that it has received in their proper sequence, on a new multicast address. Receivers that know the repaired multicast address use the same software to receive the signal as the receivers that acquire the signal directly from the source.

At present, the RTP protocol [6], that is used in many MBone audio and video applications, includes a sequence number in each packet. However, in one application of our protocol the source transmitted a "real audio" stream without sequence numbers. For this application the retransmit server encapsulated the real audio packets in RTP packets and transmitted them on an intermediate multicast address. The repair servers received the intermediate address, requested missing packets, de-encapsulated the packets, and retransmitted the repaired, real audio, signal on a third multicast address. Once again the receivers used the same software to receive the repaired signal as the signal from the source. Multicasting on the additional addresses is not as wasteful of network resources as it may appear. Multicasts are not transmitted on channels except when there are receivers. Therefore, when all of the receivers on a segment of a multicast tree receive the repaired signal, the original signal is not transmitted on that segment of the tree.

We can send a negative acknowledgments when packets aren't received, rather than a positive acknowledgement each time a packet is received, because of the continuous transmission characteristic of the media. The ability to send negative acknowledgements is particularly useful to keep the number of control messages small as the number of repair servers increases. It has been recommended [7] that data multicasts send extra packets in order to use negative acknowledgements.

We can leave the source and receivers unmodified because the objective is to improve the quality rather than achieve perfect reception. Therefore, the retransmit and repair server can be separate from the source and receivers. Not all of the requested packets are available for retransmission and not all of the recovered packets arrive at the receiver. Of course, the quality can be further improved by co-locating the retransmit server at the source or the repair server at a receiver.

2.1: Architectures

Our initial experimental systems have a very simple architecture. There is a single retransmit server located as close to the source as possible and a small number of repair servers located at major remote locations. This architecture is adequate for corporate intranets with ten or less locations, or for Internet service providers with a small number of hubs. Architectural issues are less critical with this protocol than with other multicast protocols because the protocol operates between major locations rather than between all of the receivers and the source. However, as the number of locations increases, the number and position of retransmit servers is an important performance issue. First, when there are several retransmit servers that can respond to a request it is more likely that the packet can be recovered. Second, retransmit servers can be positioned to protect expensive or overutilized links by limiting the number of repair servers that must traverse the link to obtain retransmissions.

For instance, a single repair server, on the receive end of an overutilized link can act as the retransmit server for the other repair servers that receive signals over that link. When a packet is lost on the link, instead of all of the repair servers sending negative acknowledgements on the link, one repair server sends a negative acknowledgement over the link and the other repair servers send a negative acknowledgement to that repair server. The repair server that doubles as a retransmit server continues to multicast a repaired signal for the local receivers.

This hierarchical architecture can be extended to a tree of retransmit and repair servers. Each repair server requests retransmissions from the single retransmit server that connects it to the root of the tree. Some repair servers also function as a retransmit server. The tree architecture is superimposed on the multiply connected grid that makes up the Internet. In simple, stationary systems trees can be configured manually by network operators. However, when the number of multicasts, or the size of multicast groups, grows, an algorithm is needed to select a tree and to reconfigure the tree if retransmit servers fail or the network utilization changes.

2.2: Retransmissions

When a NACK is received should the retransmit server multicast the retransmissions or send point-to-point replys? It is possible for the server to wait until all of the receivers have had a chance to report a missing packet, calculate the link costs associated with multicast and point-to-point transmissions, then select the appropriate mode. However, most reliable multicast mechanisms, such as the white board[8], multicast retransmissions, and certain analytical results indicate that all retransmissions should be multicast[5]. As a practical matter, when packets are lost in the network, they are normally lost by several sites, making it likely that multicast is the least costly way to supply retransmissions.

The main reason for not multicasting retransmissions to very large groups is that it is very likely that some of the receivers will miss most packets, resulting in almost every packet being multicast to the entire group several times. The hierarchical structures, described in the previous section, eliminate this problem. Each retransmit server is responsible for a small group of receivers and multicasts retransmissions only to that group. In addition, the application of the protocol to repair servers, rather than to all of the receivers, reduces the size of the group.

2.3: NACK Suppression

Hierarchical trees reduce the number of links traversed by a NACK. Instead of sending NACK's to the retransmit server near the source, the NACK's are sent to a nearby retransmit server. The number of NACK's can reduced further by reducing the number of repair servers that send a NACK to a retransmit server. When a packet is lost on a multicast tree, all of the receivers that depend upon that branch miss the packet. We can prevent all of the receivers that have missed the packet from transmitting a NACK by having some receivers send a NACK immediately upon detecting a loss and having others delay their request. The delay is adjusted so that a retransmission resulting from a immediate NACK is received before the delayed NACK is transmitted. The fraction of the receivers that send immediate NACK's can be controlled by having receivers that miss a transmission send an immediate NACK with a certain probability. The probability of a receiver transmitting immediately is adjusted to minimize the average number of NACK's that are transmitted.

The optimum values for the delay and the probability of delaying transmission can be calculated by making independence assumptions. As in setting many parameters in the Internet, these calculations are of limited value. A more practical approach is for each receiver to select a delay value and probability of delay. If the receiver receives retransmissions that are generated by immediately transmitted NACK's after transmitting a delayed NACK, then the delay is too short. If it obtains retransmissions that were generated by other delayed NACK's before transmitting its own, then the delay is too long. In each case, the delay is adjusted. Since the network delay continuously changes, the delay should be continuously adjusted, and is different for each receiver. Similarly, a receiver should increase its probability of sending an immediate NACK when retransmitted messages are initiated by delayed NACK's and should decrease its probability otherwise.

2.4: Repair Buffer

A repair server delays packets received on a multicast channel both to compensate for the variation in network delay and to attempt to recover missing packets. After the delay the repair server generates a new multicast sequence with some packets missing, but with interpacket times that are close to those from the original source. If the delay is too short, some packets that are eventually received or that can be recovered are missing from the reconstructed sequence. As the delay becomes longer, the buffering required becomes larger and the system may become less useful for applications, like remote class rooms, that require some interaction.

Setting the delay in video and audio applications is less critical than in data applications. In audio and video applications it is almost as acceptable to provide a good signal from now on as it is to always provide a good signal. Therefore, it is reasonable to realize when the system is operating badly and correct the problem rather than trying to predict what will be required. It is adequate to set the delay by counting the fraction of the packets that arrive after they have been bypassed on the replay channel. If the fraction becomes large, the delay must be increased, and when it is small the delay can be decreased.

A significant difference between an MBone broadcast and many other retransmission systems that have been constructed for video applications is that the amount of buffering is not a critical concern. MBone video is designed to operate within the bandwidth that is currently available in most office or campus environments, without preventing a networks continued use for data applications. Typically, bandwidths between 128 kbps and 384 kbps are used. As a result, a 10 second delay requires less than a half of a megabyte of memory. In addition, because of the rates, conventional computer memory, rather than special purpose high speed memories, can be used for this application. The memory required for a ten second delay is well within the capabilities of most home computers. In fact, it has become reasonable to use a standard computer as an MBone VCR[9] and store hour long programs.

Recent measurements of periodic packet transmissions on the Internet indicate that even during the busy period on international circuits almost all of the packets that are not lost in the network arrive within one second[10]. Therefore, less than 5 seconds, and a quarter of a megabyte of memory, are adequate to make two attempts to recover a lost packet.

A more important consideration than the amount of buffering required to support the delay is the application. A ten second delay is acceptable when listening to a news broadcast or a lecture, but is not tolerable in an interactive, conversational system. In a remote classroom, retransmissions can be used to improve the quality of reception for most of a lecture, but during a question and answer period, the delay should be reduced. The result of reducing the delay is that during this period the quality is reduced. It is our opinion that reduced quality is more acceptable during an interactive session, in which the speaker can be asked to repeat the information, than during an event in which the speaker cannot receive feedback.

3: An Implementation of a Client/Server Retransmission Architecture

We now describe a working implementation of the protocol that we have built to verify some of these concepts. The system is built entirely out of two kinds of modules, a client and a server. The client performs the repair function, it receives multicast signals from a source, obtains as many missing packets as possible from a retransmit server, and retransmits more complete multicast signals. The server performs the retransmit function, it receives multicast signals from a source and retransmission requests from the client, and retransmits packets when they are available.

3.1: Lost Packet Detection and Retrieval

Both the client and server modules operate on packet streams in the RTP format (encapsulated in UDP) [6]. RTP is an end-to-end application level protocol for real time data such as audio or video in multicast or unicast modes. Many of the real time multimedia applications on the Internet are based on RTP [*R*_1467,11], and our implementation works with any such application. The most important feature of the RTP packet format for our purposes is the incorporation of a 16-bit sequence number field in the packet header. We use this field at each client and server module to detect lost or out-of-order packet arrivals, as well as to store and retrieve packets in the playback and retransmit buffers.

3.2: Client Architecture

The client module listens to up to two transmission sources, typically one for audio and one for video (unicast or multicast), repairs any gaps in the packet stream by communicating with a specified retransmission server (unicast), and retransmits the signals on new multicast addresses. The primary structure used by the client module is a playback buffer shown in Fig. 1. Its main function is to provide storage for packets arriving while a lost packet is being retrieved by retransmission. Two associated tables, one each for audio and video packets, keep track of pending retransmission requests and their empty slots in the playback buffer.

PLAYBACK BUFFER (Circular)

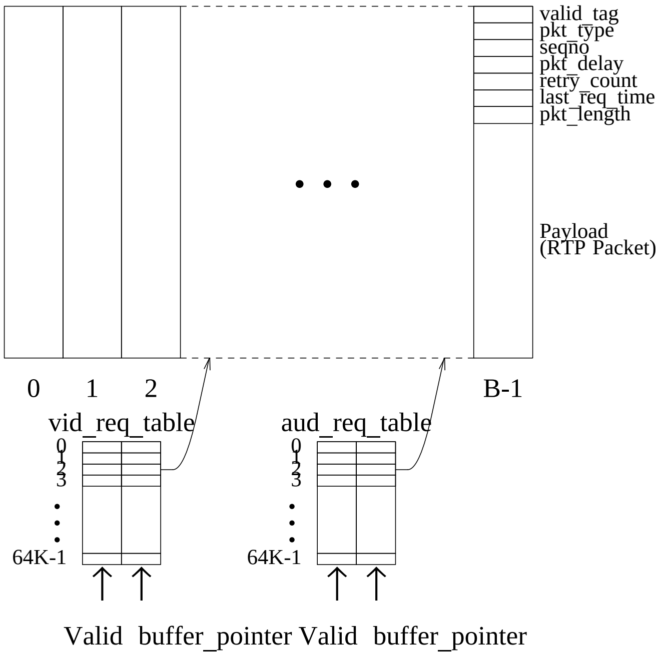

Figure 1. Playback Buffer in the Client Side Software

The playback buffer is a circular buffer of up to B packets, where each packet consists of a payload field as well as several header fields. The payload is an entire RTP packet, and the different header fields determine the state of the packet. The `valid_tag` indicates whether a slot has a packet or a ‘‘hole’’ corresponding to a lost, and as yet unretrieved, packet. Four different packet types (`pkt_type`) are implemented in our current software: video, audio, data and control packets. All four types of packets (only two may be present) are serialized through the buffer to preserve their arrival sequence, before being written out to four different UDP ports.

Inter-packet times are maintained in a field (`pkt_delay`) within each packet. This timing information is necessary because many video applications (e.g., vic) do not implement a playback point algorithm, and render frames on the display as soon as an end-of-frame packet arrives. In order to preserve the same motion as the signal at the source, we preserve the interpacket times while playing them out of the client.

The `retry_count` field records the number of retransmission requests that have been sent out for a lost packet. The `last_req_time` field records the time when the last retransmission request for a lost packet was sent out. This field is used to determine when to send out another request for this packet.

The size of the playback buffer, B, is the maximum number of packets that can be stored, and is determined by the maximum allowed playback delay. As we will see from the algorithms described in the next section, the size of the active part of this buffer varies based on the client-server round trip delays and the burstiness of the traffic. The active part of the buffer is often be much less than B. At 128 Kbps, the default rate for vic, and a packet size of 1024 bytes, the default setting, the interpacket time is about 65 ms. If $B = 4096$, we are able to store about 4 minutes worth of packets. For a 1 Mbps video transmission, which may be used on a corporate intranet, 1024 byte packets are transmitted more frequently, and the maximum delay reduces to about 30 seconds. For a 71 Kbps audio session, the default pcm2 mode of the vat application [11], the interpacket time is about 40 ms, and a 4K buffer size provides a 2.5 minutes of delay.

3.3: Client Process

The client module operates by repeatedly servicing the input and output ports. An **input_process** continuously polls in a round robin fashion the ports associated with the video source, the audio source, and the retransmission server. RTP data packets are examined to see if they are originals or retransmitted copies, and in the former case, whether any packets are missing. A moving average of the round trip time to the server is updated if a retransmitted copy is received. If missing packets are detected retransmission requests are sent to the server for those packets.

A corresponding process on the output side of the playback buffer, the **output_process**, has two major functions. The first is to send out additional retransmission requests for holes that may exist in the playback buffer. The second is to send packets on the the output multicast channel in the correct time sequence.

3.4: Client-Server Message Protocol

A retransmission request from the client module to the server identifies the source, audio or video, the sequence number of the packet, and the retry number. Our retransmission requests have the same format as RTP messages, with the missing sequence number in the sequence number field of the RTP packet, and the source identifier and retransmission request number encoded in the payload field [6]. Copies retransmitted from the server to the client consist of the requested RTP packet with an added field in front containing the source identifier and the retransmission number. Once this field is extracted, the rest of the packet is identical to what the client would have received from the video or audio port.

3.5: Server Architecture

The server module listens to two unicast or multicast sources for audio and video, stores the last M incoming packets from each source in a separate buffer and responds (by unicast or multicast) to requests from one or more clients for retransmissions. Since RTP has a 16 bit sequence number field, at most 64K packets need to be stored for each source. However, following the argument made for the playback buffer architecture, a 4K packet buffer provides about 4 minutes of video storage at 128 kbps, and is the default retransmit buffer size in our system.

However unlike the client's playback buffer, no circular buffer is needed here because the buffer will be read from random locations based on the client requests. So packets are just stored and retrieved from the location given by their sequence number modulo M, an extremely fast and simple process when M is a power of 2. This is critical to being able to service more than just a few clients. The assumption here is that the client and server will not be off by more than M packets. In fact, this asynchrony between the two is a user-defined parameter in our system — if a request is off by more than *out_of_sync* ($\leq M$) packets, it is ignored. The default value for this parameter is 512 packets.

4: Results of the Experiment

The first application for which we have used this protocol is MBone video (384 Kbps) and audio (64 Kbps) transmissions between two Lucent locations, one at Indian Hill (IH), near Chicago, and the other at Murray Hill (MH), New Jersey. The live weekly program originates at Indian Hill, and the main goal is to improve the reception for viewers in Murray Hill, who often encounter packet losses that average 6-15% over a 60 minute session, with short term losses as high a 30-40%. Most of the viewers at Murray Hill are on a single physical net, and in this case the configuration shown in Fig. 3a is a natural and very efficient mechanism for compensating for losses on the IH—MH link. A server module is located on the same net as the source at IH, and a single client located at Murray Hill receives the original transmissions from the source. It repairs the audio and video packet streams by requesting unicast retransmissions from the server, and multicasts the repaired stream on a *different* address with a very small time-to-live field. Viewers in Murray Hill tune in to this second multicast address to receive the high quality repaired audio and video programs. Fig. 4 illustrates how effective this arrangement was over one session. It shows the number of losses per thousand packets seen by a receiver over 20 minutes, for the original transmission and for our architecture. A maximum of two retransmission requests are sent out for a lost packet, though most of the lost packets are retrieved by just one retransmission request.

This architecture extends trivially to a multi spoke configuration shown in Fig. 3b, which is limited only by the number of clients a single server can handle. Our preliminary server implementation seems capable handling at least ten clients, which translates to a few hundred viewers when there are a few tens of receivers in each location. This configuration seems quite reasonable in a corporate intranet environment, with a few clients (one or two at each site) and lots of receivers served by each client. An important issue here is the second multicast address that each client uses to broadcast the repaired stream to its subset of listeners. If the time-to-live fields of this local multicast (say 1 or 2) are small enough that the multicast trees from two clients do not overlap, then all the clients can retransmit the repaired multicasts on the same addresses.

If the number of spokes is large at a hub, it may be more efficient for the server to multicast the retransmission to all the clients, instead of dealing with unicast replies. (Fig. 3c) Recall that our server module can send retransmissions out to a unicast or a multicast address. This would work particularly well if all the spokes experience roughly the same loss behavior, which is not always the case. So there is the disadvantage that during any interval the most lossy spoke would generate retransmission traffic on all the spokes.

5: Conclusion

We have described a packet recovery protocol to compensate for losses while multicasting real time services on the Internet. A unique characteristic of this protocol is that it can be implemented on the current Internet, without modifying the receivers and transmitters that take part in the multicast session.

This protocol does not experience the same degree of difficulty as other multicast protocols designed for large number of receivers because only a small fraction of the receivers take part in the protocol. As a result, there are not as many control messages and recovery

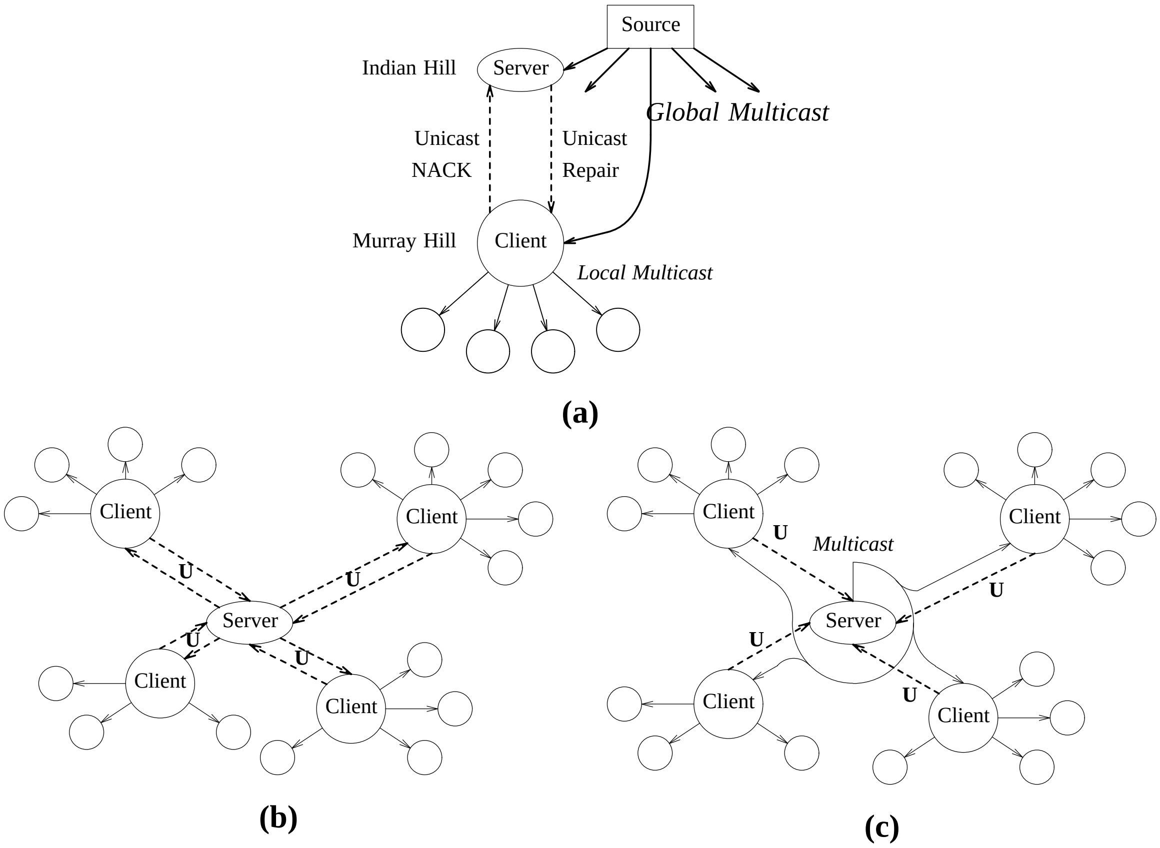

Figure 2. Configurations of Clients and Servers. (a) Configuration used in the initial experiments. (b) Generalization of hub and spoke architecture used in (a). (c) Modification of (b) to multicast retransmitted messages.

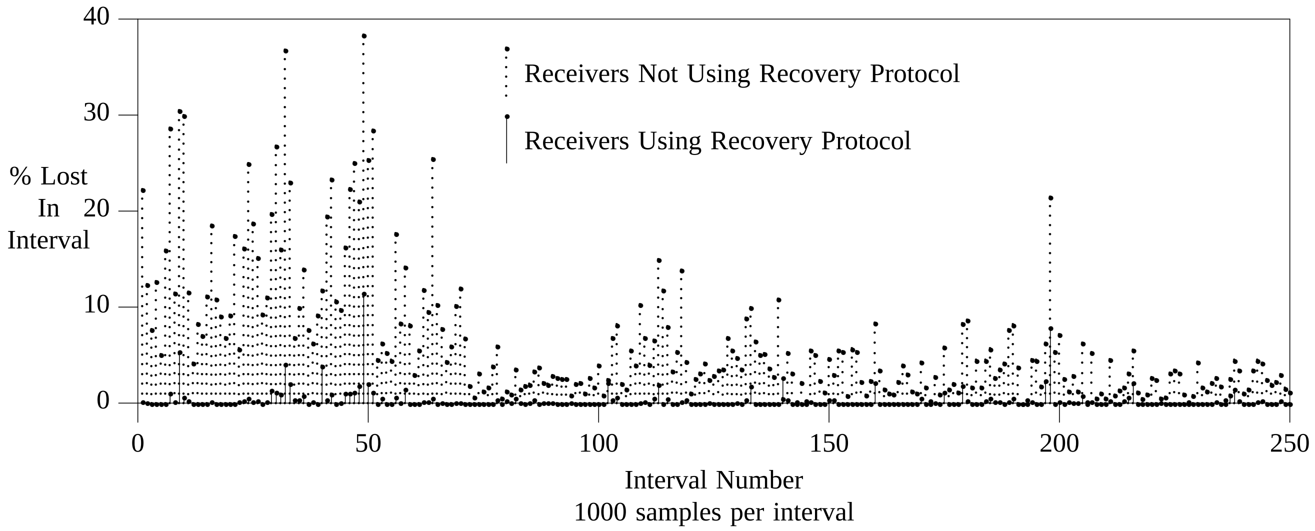

Figure 3. Percent of the packets lost in intervals containing 1000 packets at a receiver that uses the recovery protocol and at an adjacent receiver that does not use the protocol

messages and NACK suppression and tree design procedures are not as critical.

In addition, the objective of improving the quality as opposed to recovering the entire content simplifies the procedures needed to deal with the variable components of the system. For instance, the delay at the repair server can be increased if packets start arriving after they have been bypassed. The result is that the quality is improved after the delay is increased rather than trying to repair the signal that was transmitted while the delay was too short.

This protocol has been implemented on Lucent Technologies' corporate intranet, and is regularly used for multicast sessions between locations in Indian Hill and Murray Hill. Measurements from these sessions show that the protocol is extremely effective in improving the quality of received video and audio transmissions.

The protocol is ideally suited for an Internet service provider. If the ISP has a point of presence near the program source, it can improve the quality of reception at its other points of presence, even though it is not part of the content generation and even though the signal may not traverse its own network between its two points of presence. By keeping the multicast addresses that are used by the repair server private, or by encrypting the repaired signal, the ISP can provide the repaired signal to selected receivers.

Acknowledgements

The current version of the multicast protocol was implemented by Shelley Tselepsis, while at Columbia University. The authors would like to thank J. Brassil and A. Choudhury for several discussions during the course of this work; They would also like to acknowledge the help of C. Votava at Indian Hill and C-S. Lin at Whippany in setting up the multicast sessions between Indian Hill (IL), Whippany (NJ), and Murray Hill

References

[1] L. F. de Moraes, S. B. Weinstein, "The Internet Multicast from ITS: How it was Done and Implications for the Future," IEEE Commun. Mag., Jan. 1995, vol. 33, no. 1, pp. 6-8.

[2] J. C. Pasquale, G. C. Polyzos, G. V. Xylomenos, The Multimedia Multicasting Problem," UCSD/CSE Tech. Report, CS93-313.

[3] B. J. Dempsey, M. T. Lucas, A. C. Weaver, "Design and Implementation of a High Quality Video Distribution System using XTP Reliable Multicast," IWACA 94, Heidelberg, Germany, Sept. 1994.

[4] L. Delgrossi, C. Halstrick, R. G. Hertwich, H. Stuttgen, "HeiTP - a transport protocol for ST-II," Proc of IEEE Globcom '92, 1992, pp. 1369-1332

[5] S. Pejhan, M. Schwartz, D. Anastassiou, "Error Control Using Retransmission Schemes in Multicast Transport Protocols for Real-Time Media", IEEE/ACM Trans. on Networking, vo. 4, no. 3, June 1996, pp. 413-427.

[6] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force, Audio-Video Transport Working Group, Jan.1996.

[7] H. W. Holbrook, S. K. Singhal, D. R. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation," Proc. of ACM SigComm '95.

[8] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, L. Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing," Submitted for Publication.

[9] Wieland Holfelder, "MBone VCR - Video Conference Recording on the MBone", Proceeding of Multimedia'95, November 5-9, San Francisco, CA.

[10] N. F. Maxemchuk, S. Lo, "Measurement and Interpretation of Voice Traffic on the Internet," Proc. of 1997 IEEE Conf. on Commun., June 8-12, 1997, Montreal, Canada.

[11] V. Jacobson, S. McCanne, *Visual Audio Tool*, Lawrence Berkeley Laboratory. Software on-line at http://www-nrg.ee.lbl.gov/vat.