An Adaptive Multiple Retransmission Technique for Continuous Media Streams

Rishi Sinha
University of Southern California
3737 Watt Way, PHE 335
Los Angeles, CA 90089
+01-213-740-1604
rishisin@usc.edu

Christos Papadopoulos
University of Southern California
941 W. 37th Place, SAL 238
Los Angeles, CA 90089
+01-213-740-4780
christos@imsc.usc.edu

ABSTRACT

Retransmission can be used for loss recovery in continuous media applications but the number of retransmission attempts is bounded by the size of the playout buffer. For efficient recovery, a protocol must attempt as many retransmissions as possible but avoid late retransmissions. This typically requires that the playout buffer be sized in round-trip time (RTT) multiples plus some margin for error. RTT-based timers are then used to trigger retransmissions. However, this approach is problematic due to (i) the high variation in RTT commonly encountered in the Internet, which makes accurate estimation difficult, and (ii) the granularity of timers typically used prevents precise control.

We present two new retransmission-based protocols, for unicast and multicast respectively, which eliminate RTT estimation and timer-triggered events. As a result, our protocols are immune to errors due to jitter and timer granularity and recover more losses, while better suppressing unnecessary retransmission requests and retransmissions than timer-based protocols. At the same time, our protocols are simpler to implement and degrade more gracefully than timer-based protocols.

1. INTRODUCTION

Continuous media (CM) are characterized by the need for timely delivery of data to the destination, usually overriding the need for 100% reliability. However, due to the best-effort nature of the Internet a loss recovery mechanism is required to correct as many errors as possible without violating the timing constraints of the application. Traditional Go-Back-N ARQ [1] is not suitable because such methods are designed for 100% reliability and thus may violate the timely delivery of data.

In order to eliminate jitter introduced by the network, CM applications use a playout buffer to smooth playback. While data is held in the playout buffer a limited number of retransmissions may be attempted. For example in [2], one retransmission is attempted per lost frame. Depending on the allowable size of the playout buffer multiple attempts may be possible.

A simple approach to generate multiple retransmissions is to estimate the round-trip time (RTT) and use timers to trigger retransmissions. We call this a *timer-based* protocol. Solutions based on timers, however, are hard due to the following two reasons: (i) the high variation in round-trip times (RTTs) commonly encountered in the Internet, and (ii) the coarse granularity of timers typically used in protocol implementations. For example, TCP implementations make use of two timers, FASTTIMO and SLOWTIMO, which have a granularity of 200 ms and 500 ms respectively. In this paper we present two new protocols that generate multiple retransmissions and do not require RTT estimation or timer-triggered events. We call these *timerless* protocols. We show that these protocols are able to generate more retransmission requests within the available period than a timer-based alternative. At the same time, timerless protocols are better at suppressing unnecessary retransmission requests and retransmissions.

The rest of the paper is organized as follows. In Section 2 we present background and related work. In Section 3 we present the unicast version of our recovery protocol, while in Section 4 we present the multicast version. In Section 5 we present performance results of our unicast implementation. Evaluation of the multicast protocol is left for future work. Finally, we conclude in Section 6.

2. BACKGROUND

In this section, we briefly survey existing methods of loss recovery in CM. Perkins et al. [12] define a taxonomy of sender-based methods of loss repair, including interleaving (which is a form of concealment) and Forward Error Correction (FEC). FEC techniques are well established and have been found to perform well [3]. However, there is a

This research has been funded (or funded in part) by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

trade-off between FEC and ARQ in terms of the bandwidth used and the delay experienced in loss correction. FEC tends to be less bandwidth-efficient than ARQ, though it is typically able to recover loss in less than an RTT. There are well-established arguments for using FEC [4] or ARQ [5] under different conditions, as well as hybrid ARQ/FEC schemes [6]. For brevity, we do not repeat those arguments here. Retransmission-based error recovery in CM has also been dubbed Soft ARQ [7].

In retransmission-based schemes upon detection of a gap, the receiver decides whether to send a negative acknowledgement (NACK) based on the current estimate of RTT and the playout time of the missing frame [2]. However, in order to make multiple attempts, the protocol requires some means of detecting the loss of a request or a retransmission. The simplest way to achieve this is with a timer set to expire soon after the current RTT estimate, if the retransmission is not received. For this an accurate estimate of the RTT is critical and current retransmission time-out (RTO) estimation techniques [9] may not be adequate [8].

The protocols we present in this paper do away with timers such as those used in RTT estimation and therefore avoid one of the hardest problems with retransmission for CM.

3. TIMERLESS UNICAST RECOVERY

Instead of using timers to trigger multiple retransmissions, we exploit the continuous flow of packets from the sender to deliver information about which retransmissions have been sent. The inter-packet duration is typically small enough to minimize detection latency.

We call the unicast version of our timerless protocol MR-UNI (for Multiple Retransmissions for Unicast). In addition to the sequence numbers normally used to number frames, MR-UNI uses a second sequence number, which labels NACKs. The first sequence space is used to number and detect loss of original transmissions. The second sequence space detects lost retransmissions and NACKs. In the following discussion, "sender" and "receiver" refer to the CM producer and consumer respectively.

In MR-UNI, each NACK carries a NACK sequence number (NACKSEQ) and the sequence number (SEQ) of the lost frame. Each data frame from the sender carries its frame sequence number (SEQ) as usual, and also the highest NACKSEQ serviced by the sender until that moment.

The rules for assigning and servicing NACK sequence numbers are as follows:

1. The receiver increments a counter for every distinct NACK it sends and numbers the NACK with the value of this counter.
2. The receiver may send duplicate copies of a NACK, but these have the same NACK sequence number, and are not considered "distinct" NACKs.
3. The sender services each distinct NACKSEQ only once; duplicate NACKs are ignored.
4. The receiver may assign a new number to a NACK sent previously. This creates a fresh NACK, numbered according to rule 1.
5. The sender must service each distinct NACKSEQ it sees, regardless of whether the frame being requested has been retransmitted before.

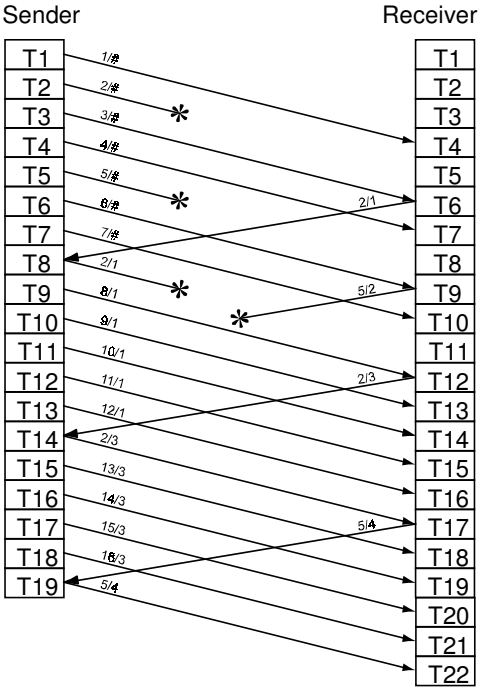

Figure 1: Operation of MR-UNI.

On detecting a gap in the NACKSEQ stream the receiver concludes that there was a retransmission failure (due to the loss of NACKs or retransmissions). If there is sufficient time to attempt another retransmission, the receiver generates a new NACK with distinct NACKSEQs. On receiving this new NACK the sender retransmits the requested data.

Figure 1 shows an example timeline for the operation of the MR-UNI protocol. The notation *x*/*y* indicates SEQ/NACKSEQ. For packets from the sender to the receiver this indicates the current frame sequence number and the highest NACK sequence number serviced so far. For packets from the receiver to the sender (i.e., NACKs), it indicates the sequence number of the missing frame and the sequence number of the NACK. The symbol "#" indicates

null information. Losses are shown as truncated arrows. The NACKs generated at T6 and T9 are the result of a discontinuity in the regular frame sequence numbers while those generated at T12 and T17 are the result of a discontinuity in the NACK sequence number. Note that this second type of discontinuity can arise either due to loss of a retransmission (detected at T12) or due to loss of a NACK (detected at T17).

The last statement highlights a significant difference between the two sequence number streams from the sender (SEQ and NACKSEQ). While the first changes as new data is generated and transmitted, the second changes only when loss occurs. Thus, the protocol detects lost retransmissions but may miss lost NACKs. For example, in Figure 1, the loss of NACK 5/2 (time T9) would go undetected if frame 2/1 were not lost (time T8).

To deal with NACK loss, we repeat NACKs periodically. In addition, we make NACKs cumulative, listing all NACKSEQs pending at the receiver.

4. TIMERLESS MULTICAST RECOVERY

In this section we describe a multiple retransmission scheme for multicast that is based on the same principles as the unicast scheme (MR-UNI) described above. As with MR-UNI, the distinguishing feature of this protocol is that it does not rely on timers, but uses supplemental information from the sender about the retransmissions made so far to enable the receivers to trigger multiple retransmissions.

A naïve implementation would use multiple instances of the MR-UNI protocol, one per receiver. This is not efficient since the sender must keep track of the NACK sequence number state for each receiver and may potentially send duplicate retransmissions if many receivers request the same packet. It would be more efficient to use a common sequence space for retransmissions.

We propose a multiple retransmission protocol for multicast (MR-MCAST), with the following features:

- **Sender naming**. With each original frame the sender includes information about *which* frames were retransmitted in the past and *how many times* they were retransmitted. There is no second sequence number space as in MR-UNI. We call this sender naming because the receiver no longer numbers (names) NACKs.
- **NACK cycling**. At a given time, the sender may have information about several retransmissions. If the number grows large, the sender may distribute this information across N consecutive frames. We call this sender-side process NACK cycling. The number N is chosen keeping in mind the minimum path MTU in the multicast group, in order to avoid fragmentation of packets.

In MR-MCAST, NACKs are unicast to the sender but retransmissions are multicast to all receivers. The protocol suppresses unnecessary responses to multiple NACKs for the same frame. NACK implosion at the sender is, however, an issue. We do not include explicit implosion controls to avoid the latency overhead included with most implosion control mechanisms. Low recovery latency is typically important in interactive groups, and we expect such groups to be small, such that NACK implosion will not become a problem. If, however, it becomes necessary to deal with NACK implosion, it is possible to employ a hierarchical

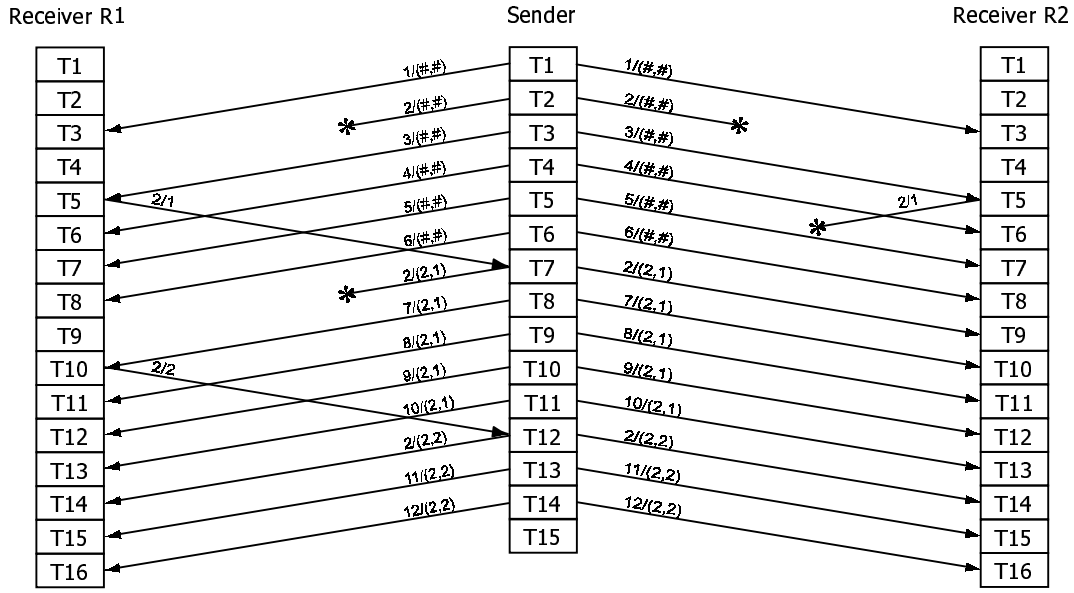

Figure 2: Operation of MR-MCAST

scheme to suppress redundant NACKs at aggregation points between the sender and the receivers. Such schemes have been well studied and documented in the reliable multicast literature [10][11].

We now describe the MR-MCAST protocol in more detail. In addition to the sequence number of the data, each frame from the sender contains a vector. Each element of the vector is composed of two fields – PASTSEQ, the sequence number of a frame retransmitted in the past, and PASTREPS, the number of times this frame has been retransmitted. PASTSEQ is in the same sequence space as the original data. Receivers request retransmissions not only by identifying the sequence number desired, but also by specifying a *repetition* (REPS) count. This number serves to suppress unnecessary retransmissions at the sender by identifying NACKs requesting the same data.

Figure 2 shows a timeline for an exchange between a sender and two receivers (R1 and R2). We take the case of N=1. Frames from the sender are denoted by SEQ/(PASTSEQ, PASTREPS), and NACKs by (SEQ/REPS). The following are the events of interest in the figure:

- Both receivers lose the frame with sequence number 2, sent at T2.
- Both receivers generate a NACK at T5. The NACKs are unicast.
- R2's NACK is lost.
- R1's NACK arrives at the sender at T7 and the sender immediately multicasts a retransmission.
- The retransmission is received by R2 at T9, despite the fact that R2's NACK was lost.
- The retransmission is lost for R1.
- A new NACK is generated by R1 at T10.
- The NACK arrives at the sender at T12 and a new retransmission is multicast.

In MR-MCAST, the sender responds to a NACK only if it has the frame in its buffer and the NACK's REPS number exceeds the number of retransmissions that have been made for this frame. The fact that retransmissions are multicast suppresses unnecessary retransmissions for the same frame by rejecting NACKs with a REPS count that is too low. For example, in the previous figure if the T5 NACKs from both R1 and R2 did arrive at the sender, only the first would be serviced, because they both bear the same REPS count. Receivers discover the current value of REPS from information contained in the stream of original frames. When a receiver discovers that its request was serviced but the retransmission did not arrive, the receiver increments the REPS count and sends a new NACK, as is illustrated for receiver R1 at time T10.

Note that the loss of a NACK in MR-MCAST is still an issue – the loss of a NACK may not be detected without further losses. For this reason, we still need the protective redundancy mechanisms of MR-UNI. However, there is already some redundancy built into MR-MCAST with respect to NACK generation – when multiple receivers experience the same loss, each of them generates a NACK which is unicast to the sender. Only one of these NACKs needs to successfully arrive at the sender in order to trigger a retransmission. As illustrated in the figure, R2 receives its retransmission despite the loss of its NACK since R1's NACK successfully arrived at the sender.

5. EVALUATION

In this section we evaluate the performance of MR-UNI described above, comparing it with a unicast timer-based method. To do so we implemented both protocols and tested them in lab experiments. We have not yet implemented the multicast version. This is left for future work.

Since there are no well-established RTO calculation algorithms for CM, we used the TCP RTO calculation and exponential backoff for the timer-based protocol. This is a well-established mechanism, so we used it as our reference. While this mechanism performs well in TCP for non-latency-sensitive applications, it is not clear how well it will perform in latency-sensitive continuous media applications.

5.1 Analysis

We can obtain the theoretical correction probability by assuming that packet losses are independent and occur with some fixed probability. If the packet loss probability is p, the probability that a retransmission attempt will be successful is $(1-p)^2$, because that event is the successful transmission of both the NACK and the retransmission. Thus, given that a loss has occurred, the number of retransmission attempts required to recover the lost packet is a geometrically distributed random variable with parameter $(1-p)^2$. The expected number of retransmission is $1/r$, where $r=(1-p)^2$. If the maximum number of attempts allowed is x, the probability of correcting a loss is $1-(1-r)^x$. Conversely, if we want a loss correction probability of at least q, we should allow at least $\ln(1-q)/\ln(1-r)$ attempts.

5.2 Experimental Setup

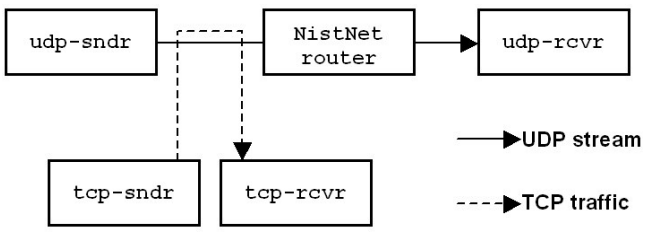

Figure 3: Experimental setup.

We implemented both protocols as a loss-recovery layer over a fixed-rate dummy CM streaming application. Unlike standard implementations of timer-based protocols that use timers with resolution of 200 – 500 ms, our timer-based protocol used accurate timers with a resolution of a few milliseconds. Also, the actions indicated by timer expiration are executed as soon as the timer expires, rather than being deferred or batched. This was done to ensure we give the timer-based protocol the best chance of recovery.

Figure 3 shows the setup used for our lab experiments. The constant-rate UDP stream runs from `udp-sndr` to `udp-rcvr` across a PC router that uses NistNet [14] to emulate network loss and delay. HTTP traffic between `tcp-sndr` and `tcp-rcvr` is used to emulate background traffic. This traffic competes with the UDP traffic, introducing jitter between `udp-sndr` and `udp-rcvr`. HTTP traffic is implemented as a Poisson process that requests a 20 KB file served by `tcp-sndr`. The average rate of the Poisson process is varied to obtain varying traffic volumes. In experiments where it was not necessary to vary the amount of competing traffic, we used TCP traffic consisting of three independent Iperf [13] flows, each achieving 1 Mbps throughput under steady conditions without the UDP traffic.

The RTT between `udp-sndr` and `udp-rcvr` was set to 30 ms. The RTT for the HTTP traffic was also set to 30 ms using NistNet. The UDP stream consists of 1516-byte packets sent at 536 packets/s, which results in a throughput of 6.5 Mbps. The UDP stream also experiences loss due to NistNet. In the discussion that follows, *forward and reverse loss* refer to the packet loss percentage on the forward and reverse UDP paths, *request rate* is the average rate of the Poisson process of HTTP requests and *buffer size* is the size of the playout buffer on the UDP receiver, which is equal to the size of the buffer on the sender, specified in milliseconds.

Table 1 Experiments conducted.

I.	Vary forward and reverse loss; buffer size = 240 ms; competing traffic using Iperf.
II.	Vary forward loss; reverse loss = 0; buffer size = 240 ms; competing traffic using Iperf.
III.	Vary buffer size; forward loss = reverse loss = 5%; competing traffic using Iperf.
IV.	Vary HTTP request rate; forward loss = reverse loss = 5%; buffer size = 240 ms; competing traffic using HTTP.

Table 1 lists the conditions for each experiment we conducted. For each set of parameters, we conducted five 20-minute runs with each protocol. In the following graphs, each point plots the mean of the results from the five experiments, with the standard deviation shown as vertical error bars.

5.3 Results

Figure 4 through Figure 7 show the percentage of losses recovered by the timerless and timer-based protocols in the experiments listed in Table 1. We see that the timerless protocol consistently outperforms the timer-based protocol. This is due to the fact that accurate RTT estimation and timely RTT-based triggers are indeed a problem under the modest background load considered in our experiments. Figure 4 and Figure 5 show the results when the percentage of lost UDP packets is varied. Although *sustained* losses at some of the levels considered are unlikely to occur in the Internet, these experiments stress the protocols in order to reveal hidden trends in their behavior. We can see that the timerless protocol degrades much more gracefully with increasing loss than the timer-based protocol, whose curve descends sharply. Figure 6 and Figure 7 show that the performance improvements of the timerless protocol continue over a range of playout buffer sizes and background traffic volumes. The latter contributes to variability in the RTT.

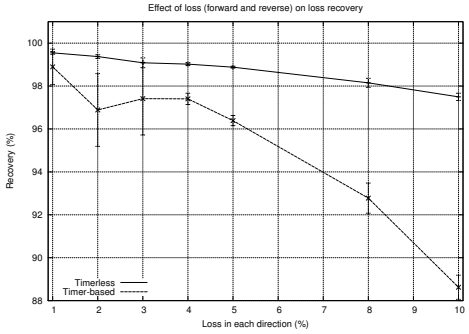

Figure 4: Effect of loss rate (bidirectional loss).

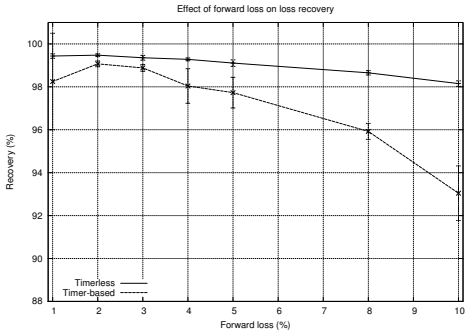

Figure 5: Effect of loss rate (unidirectional loss).

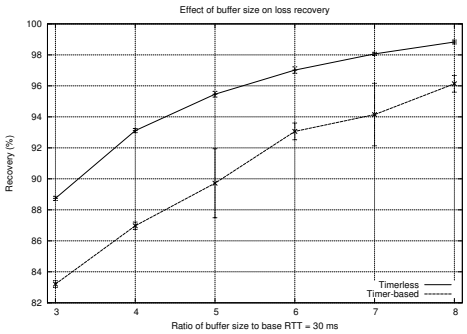

Figure 6: Effect of playout buffer size.

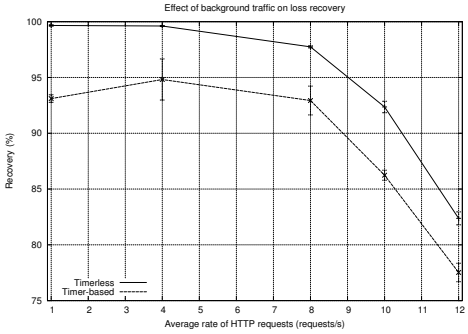

Figure 7: Effect of traffic volume.

6. CONCLUSIONS

In this paper we proposed two new retransmission-based error recovery protocols, one for unicast and another for multicast that do not rely on timers to perform multiple retransmissions. We motivated our designs by describing the problems with timer-based protocols, which include difficulties in maintaining accurate RTT estimates and timeouts in the face of jitter and the coarse granularity of timers typically used to implement protocol timers.

The unicast version of our protocol employs a second sequence space used for retransmissions, which allows speedy loss detection. The receiver is in control of this space, leading to *receiver naming* of lost data. In the multicast version, however, using a separate space for each receiver is inefficient, so we number retransmission attempts to help detect lost retransmissions. The sender is responsible for numbering retransmissions, which results in *sender naming* of lost data.

We have implemented the unicast version of our timerless protocol and evaluated its performance by comparing it to a timer-based protocol (which we also implemented). We demonstrated through lab experiments that our technique performs better than timer-based implementations even when accurate timers are used. We showed that the performance of the timer-based protocol drops sharply with increasing loss probability, while the timerless protocol degrades more gracefully. In addition, the timerless protocol maintains an advantage over a range of playout delays. While the performance difference between the two protocols is not dramatic, it is important since any packet loss degrades the playback quality. Moreover, we believe that our timerless protocols are simpler, and thus easier to implement than timer-based protocols.

We plan to evaluate the multicast version of our protocol in future work.

7. REFERENCES

[1] S. Lin, D. J. Costello Jr. and M. J. Miller. Automatic-Repeat-Request Error-Control Schemes. IEEE Communications Magazine, vol. 22, no. 12, December 1984.

[2] C. Papadopoulos and G. M. Parulkar. Retransmission-based Error Control for Continuous Media Applications. Proc. 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), April 1996.

[3] V. Hardman, M. A. Sasse, M. Handley, and A. Watson. Reliable Audio for Use over the Internet. Proc. Internet Society' s International Networking Conference (INET), June 1995.

[4] J.-C. Bolot and A. V. Garcia. The Case for FEC-based Error Control for Packet Audio in the Internet. ACM Multimedia Systems, 1997.

[5] M. Zorzi. Performance of FEC and ARQ Error Control in Bursty Channels under Delay Constraints. Proc. IEEE Veh. Technol. Conference, 1998.

[6] R. H. Deng and M. L. Lin. A Type I Hybrid ARQ System with Adaptive Code Rates. IEEE Transactions on Communications, vol. 43, no. 2/3/4, 1995.

[7] M. Podolsky, S. McCanne, and M. Vetterli. Soft ARQ for Layered Streaming Media. Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, Special Issue on Multimedia Signal Processing, Kluwer Academic Publishers, April 2000.

[8] D. Loguinov and H. Radha. On Retransmissions Schemes for Real-time Streaming in the Internet. Proc. of IEEE INFOCOM, May 2001.

[9] V. Jacobson. Congestion Avoidance and Control. Computer Communication Review, vol. 18, no. 4, August 1988.

[10] J. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. Proc. IEEE INFOCOM, March 1996.

[11] C. Papadopoulos, G. Parulkar and G. Varghese. LMS: A Router-Assisted Scheme for Reliable Multicast. To appear, IEEE/ACM Transactions on Networking.

[12] C. Perkins, O. Hodson and V. Hardman. A Survey of Packet Loss Recovery Techniques for Streaming Media. IEEE Network Magazine, September/October 1998.

[13] Iperf. http://dast.nlanr.net/Projects/Iperf/

[14] NistNet. http://snad.ncsl.nist.gov/nistnet/