Appeared in the proceedings of NOSSDAV 1996

Retransmission-Based Error Control for Continuous Media Applications

Christos Papadopoulos
christos@dworkin.wustl.edu
(314) 935-4163

Gurudatta M. Parulkar
guru@arl.wustl.edu
(314) 935-7534
http://www.arl.wustl.edu/~guru

Computer and Communications Research Center
Department of Computer Science and Applied Research Laboratory
Washington University
St. Louis MO 63130-4899

Abstract[1]

Retransmission-based error recovery has been, in general, considered inappropriate for continuous media (CM) applications, because of its latency. However, retransmission is still attractive because it requires minimal network bandwidth, (less than either peak-bandwidth allocation or FEC for bursty streams) and processing cost. Despite its latency, we believe that retransmission can be used successfully in many cases, especially if playout buffering is employed.

We have designed and implemented a retransmission-based error control scheme for CM applications, which aims to provide the best possible reliability at a minimal cost, without violating the application's timing constraints. We have enhanced selective-repeat retransmission with: (1) playout buffering to increase the time available for recovery, (2) gap-based rather than timer-based loss detection to minimize loss detection latency, (3) implicit expiration of sender retransmission buffers to eliminate acknowledgments, (4) conditional retransmission requests to avoid triggering late, unnecessary retransmissions, and (5) data integrity information delivery to the application to aid in concealment. Experimental results show that the mechanism significantly reduces observed loss without violating the application's delay constraints.

1. INTRODUCTION

Continuous media (CM) streams are characterized by periodic and relatively long lived (e.g., minutes, hours or more) data exchange. Examples of CM streams include video, audio, image animation, and others. The salient characteristics of CM streams are their periodicity and strict timing requirements. Some CM streams (especially those carrying visual information, like video and animation) require very high bandwidth (>100 Mbps) if transmitted in raw form. To save on bandwidth, such streams are often compressed, which leads to highly bursty, variable bit-rate (VBR) output streams. Transmitting a VBR stream over packet switched networks is difficult without packet loss due to congestion, or without wasting substantial bandwidth with a peak rate reservation. Moreover, compressed CM streams are far less tolerant to packet loss, because compression eliminates a significant amount of redundancy present in the uncompressed data. In addition, compressed data often contains control information (e.g, frame headers) whose loss may lead to misinterpretation or discarding of a large portion of otherwise correctly received data.

2. AVOIDING LOSSES

To minimize network losses while maximizing the statistical multiplexing gain, several forms of congestion control have been proposed. These methods include adapting the source bandwidth according to congestion in the network [3], renegotiating the network reservations according to the source requirements [13], creation of multiple concurrent streams with different rate/quality characteristics [1], and hierarchically encoded streams [6]. These schemes are promising, even though some require support from the entire network. Bandwidth adaptation in these methods is typically not instantaneous, and thus some losses may still occur while the sources and the network settle to a new congestion-free state. Losses may also be experienced due to other reasons not related to congestion, like route changes, interference, etc. Thus, it is desirable for applications to complement their congestion control method with some form of error control which can recover losses quickly, to help achieve a graceful degradation of quality[2].

[1]This work was supported in part by ARPA, NSF, Rome Labs, and an industrial consortium of Ascom Nexion, Bay Networks, Bell Northern Research, NEC America, NIH, NTT, Samsung, Southwestern Bell and Tektronix.

[2]We assume that quality degrades more gracefully if lost data is recovered and the application subsequently reduces its data rate to control congestion.

Traditional error control schemes mostly use retransmission and provide 100% reliability at the expense of latency. This is clearly the wrong model for CM applications, where late packets are as good as lost packets and some loss can be tolerated. However, retransmission has been widely dismissed even as a method to do partial recovery, in favor of other error control methods, including forward error correction (FEC) and concealment. We believe that perhaps retransmission was dismissed without investigating its full potential. We briefly discuss our reasons next.

2.1 Forward Error Correction (FEC)

FEC [2] is an open-loop error control scheme which allows trading bandwidth for lower error rate while maintaining latency close to the RTD. To perform FEC, appropriate redundant data is sent in addition to the application's original data. The redundant data is used at the receiver to reconstruct the original data if any network losses occur. Thus, loss is recovered without any end-to-end exchange between the sender and receiver, which makes FEC very attractive for applications with delay sensitive data. However, finding a good bandwidth/loss compromise is not always easy: computing the amount of redundancy and thus the bandwidth allocation for the FEC stream becomes difficult for bursty network losses, because the redundancy is proportional to the longest burst loss, which is very hard to predict. It is possible that the FEC bandwidth overhead required to recover bursty losses may be as much as 25 - 30% [17]. Thus, FEC can be a costly solution for high bandwidth bursty CM applications (this may not be true for low bandwidth CM streams like audio).

2.2 Concealment

Concealment [21] is not strictly an error control scheme because it does not actually recover lost data, but rather creates an approximation based on information before and/or after the loss. Concealment is strictly receiver based and does not require any end-to-end exchange. Examples of concealment include substitution of lost data with data from an earlier frame and various methods of interpolation and approximation. However, concealment creates artifacts, which may be detectable by the user, depending on the amount of data lost, the type of stream and the effectiveness of the concealment algorithm. High-quality concealment algorithms might be expensive for high bandwidth applications and may necessitate the use of specialized hardware. Finally, since the effectiveness of concealment depends on the amount and correct interpretation of received data, concealment becomes much harder with bursty loss.

Unless 100% reliability can be guaranteed, we believe that some form of concealment will be necessary at the receiver to hide the occasional unavoidable error. However, there is a clear tradeoff between loss and the effectiveness of concealment (for a given concealment method), so even if concealment is available, there is a strong incentive to keep losses low (perhaps by employing error control) to reduce the cost of concealment and achieve graceful degradation.

2.3 Retransmission

Retransmission-based error recovery has been, in general, considered inappropriate for CM applications. The main reason is that retransmission requires at least one additional round-trip delay to recover lost packets, which may be unacceptable to CM applications. One commonly cited example is that in a US coast-to-coast NTSC video stream, retransmitting a packet requires at least 40 - 60 ms, which is larger than the frame period (33 ms), and thus losses in a frame cannot be recovered in time.

While not applicable to cases where the RTD is large, it has been shown that retransmission is feasible in many cases where the RTD is relatively small (e.g., LANs and MANs), especially if a playout buffer is used to increase the time available for recovery [9,19]. Despite its latency drawback, we believe that retransmission-based error control is still an attractive solution because of its modest bandwidth and processing costs. For example, unlike FEC, retransmission requires network bandwidth proportional to the loss rate, not the data rate. In addition, processing costs associated with retransmission are low and all processing can be easily done in software (as we demonstrate in this paper). In fact, retransmission is still used in several high-speed protocols [10]. In contrast, while some FEC encodings are simple and can be done in software (e.g. XOR), they involve data touching which is expensive.

The playout buffer is an important component of any retransmission scheme aimed at a latency-constrained environment, and is perhaps the most important difference with traditional retransmission schemes. However, as we argue later, the costs associated with the playout buffer are small. It is important to note that the size of the playout buffer is a tradeoff between the gain in recovery time and the delay imposed on CM (especially interactive) applications. The playout delay in interactive applications is limited to a few hundred milliseconds, but this is still significantly larger than the RTD (and hopefully the jitter) in future LANs and MANs. Moreover, playout buffering can be made much larger in non-interactive (e.g, stored media) applications, allowing perhaps enough time for several retransmission attempts. An example where large playout buffering is possible, is a regional video-on-demand (VOD) entertainment server serving individual homes, where good video quality is essential.

Even if retransmission can be given enough recovery time to be feasible, other important questions remain to be answered. For example, how will retransmission react with congestion control? Will the influx of retransmissions after a network loss period cause more congestion and more loss? Can retransmission be used in a multicasting environment in a scalable fashion? We will not attempt to answer these questions in this paper. Such questions require detailed studies which are beyond the scope of this paper. However, we will attempt to give qualitative justifications why we believe that these problems can be solved.

The problem of interaction between retransmission and congestion is common to all retransmission-based schemes. One way to tackle this problem is for the application to set aside some bandwidth for retransmissions (as with FEC where the application reserves additional bandwidth for the FEC encoding). During congestion the application reduces its rate enough so that the new data plus retransmissions do not exceed the reservation (we again assume that delivering retransmissions, perhaps at the expense of reducing the rate of new data, is important, so that the stream will degrade more gracefully). Unlike FEC, however, the retransmission bandwidth is not used most of the time, which increases the statistical multiplexing gain in the network. In addition, as has been observed in [19] the network loss periods may be so short that it is possible that by the time a retransmission is injected in the network the congestion may have already cleared.

Retransmission typically does not scale very well in multicast environments without some form of implosion control. However, retransmission has been shown to scale very well in multicast environments where data can be recovered locally from other receivers in the multicast group [18,15,11]. FEC is better suited for multicast because it does not suffer from implosion. However, some problems still remain: the FEC encoding must contain enough redundancy to provide satisfactory service to receivers who may see different loss rates, which may incur a significant bandwidth overhead.

To summarize, we believe that retransmission-based error control although not suited for all CM applications, is still an attractive, low-cost solution and remains a serious candidate for CM error control. Retransmission is well suited for most unicast stored media and many interactive applications where the RTD is low, and for multicast applications if the right implosion control framework is used. In special cases, retransmission can also be used in a multicast environments with large RTD: for example, in a live multicast connection if some of the receivers act as media recorders, they can still benefit from retransmissions even if they arrive too late to benefit other receivers.

From the remaining related work on retransmission for CM, the work closest to ours is Partially Reliable Streams [8]. We are not aware of any evaluation studies of PRS.

The remainder of this paper is organized as follows: In Section 3, we describe the features of our retransmission-based error control scheme that allow it to support delay-sensitive CM applications without violating their timing constraints. Section 4 presents details of our implementation. In Section 5 we present experimental results on Ethernet and our local 155 Mbps ATM testbed. Section 6 describes our extensions to the scheme to support multiple retransmissions, which are currently being implemented. Section 7 presents our conclusions.

3. MAKING RETRANSMISSION WORK

In order to minimize the retransmission bandwidth and latency we adopt selective repeat rather than go-back-N retransmission. The appropriateness of selective repeat for high-speed networks has been already demonstrated widely in literature [5,7,16]. To maximize the probability of recovery, we have added the following features:

3.1 Playout buffering

The time available for recovery may be increased with no perceptible deterioration in quality to the user, by introducing limited buffering at the receiver. This is called *playout buffering* and the buffering delay is called *playout or control delay.* In interactive applications the playout delay is limited by the perceptual tolerance of the user, which is around 200 ms [4]. In other words, a human user can tolerate a maximum channel round-trip delay of 200 ms in an interactive conversation[3]. In interactive applications the delay must be allocated to both endpoints, meaning that each may use up to 100 ms of playout delay. An example of using playout buffering is shown in Figure 1. We assume that frames are generated every 33 ms, so a playout buffer of up to three frames may be used, which increases the time available for retransmission is by 99 ms. This is sufficient for several retransmission attempts in LANs and MANs. Again, we note that the playout delay can be much larger in case of stored media retrieval (e.g., video-on-demand) without any adverse effect on the perceived quality. It hardly matters to a viewer if the playback of a movie starts

[3]This number is still debated. Some claim that this is actually a lot higher, perhaps as much as 400 ms.

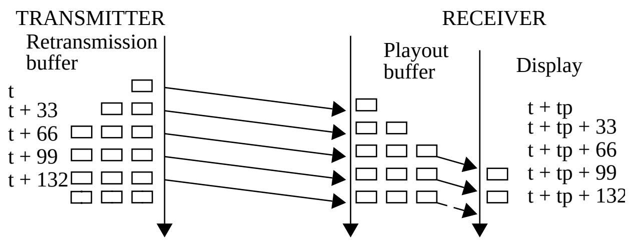

Figure 1: Playout buffering

a few seconds after the movie request if a continuous, error-free playback can be guaranteed from that point on. In practice the playout delay may be limited by the desired delay for control operations like fast-forward, pause, etc., which is approximately 0.5 - 1 second.

The size of the playout buffer is fairly small in most cases. For example, buffering 3 frames of NTSC compressed video (5 - 10 Mbps), requires a playout buffer of 62.5 - 125 kilobytes. For compressed HDTV (about 20 Mbps), the size of the playout buffer for three frames is 250 kilobytes. These numbers are well within the capabilities of modern workstations. Note that the digital frame buffer itself is much larger than this. We are optimistic that this will also be within the capabilities of television receivers and set-top boxes: some of today's high-end television sets already have enough memory to store frames for special effects (freeze-frame, slow motion, etc.).

3.2 Gap-based loss detection

Most transport protocols rely on timers to detect losses. In timer-based loss detection, the sender associates each packet (or group of packets) with a timer. If the timer expires before an acknowledgment is received, the packet is retransmitted. The time-out values are typically large (several times the RTD), which adds significant delay to loss detection. Therefore, unless the timer values can be determined very accurately, timer-based loss detection is not appropriate for continuous media applications.

We believe that gap-based loss detection [12] at the receiver combined with NACKs is more appropriate for CM applications. In gap-based loss detection, each packet carries a sequence number. A gap is detected when a packet arrives with a sequence number higher than expected. In gap-based loss detection a gap is detected only after another packet is received. Therefore losses are detected quickly if data is sent frequently (as in high-bandwidth CM applications), provided that burst losses are not too large. In addition, loss detection does not require per-packet timers and is not dependent on accurate estimation of the round-trip delay (which may fluctuate during congestion).

It is important to note, however, that gap-based loss detection is applicable only if the underlying network preserves packet sequencing. For networks that do not support FIFO delivery of data, gap detection becomes more difficult, but may still be used if a decision is made about how many out-of-sequence packets can be received before a packet is considered lost. Gap-based loss detection may take longer than time-out loss detection during prolonged network loss periods. However, sending a retransmission request before the network loss period is over may cause the request or the retransmitted data to be lost again. With gap-based loss detection, loss is detected after a packet makes it through, so there is a good possibility that the network loss period is over. Gap-based loss detection works well in LANs and MANs, where the RTD is significantly smaller than the granularity of timers used in timer-based loss detection. Reducing timer granularity can be expensive.

3.3 Implicit expiration of sender retransmission buffers

We assume that the period of CM is constant and the sender knows the period of the CM stream. We also assume that the sender finds out the receiver's playout buffer size during connection setup. The sender can then estimate how long to keep data in its retransmission buffers before the data expires. Expired data can be safely discarded. Therefore, an explicit ACK from the receiver is not required to discard retransmission buffers. A simple way to implement this is for the sender to maintain a retransmission buffer with the same number of slots as the receiver's playout buffer. Thus, whenever new data is sent the oldest data in the retransmission buffer can be discarded because it has expired. Implicit data expiration eliminates delays associated with waiting for ACKs. It also eliminates retransmission of packets that would arrive late: if a retransmission request is delayed and arrives after the data has been discarded, no packets are retransmitted. Therefore the sender need not check if packets will arrive at the receiver in time before it retransmits. The sender simply retransmits if the requested data is still in the retransmission buffer.

3.4 Conditional retransmission

Since data in continuous media has a limited lifetime, there is no point requesting retransmission if the retransmitted packets will not arrive in time. In other words, retransmission should be aborted if the time left before presentation is less than the RTD. This may happen after multiple retransmission attempts have failed, or if the RTD increases (perhaps due to a change in route). Late retransmissions are undesirable because they waste network bandwidth and CPU cycles, contribute to congestion and may delay new data. To avoid late retransmissions, the receiver keeps an estimate of the round-trip delay and the presentation time for each frame and ensures that retransmission requests are generated only if the presentation time is greater than the current RTD estimate.

3.5 Data integrity information delivery to the application

Since the time available for recovery is limited, unless enough resources are reserved there is no guarantee that data delivered to the application will be error-free. Our scheme maintains explicit information about the integrity of the received data, which is delivered to the application along with the data. This information includes the location of missing pieces of data, if any. Such information can be valuable to the application in deciding how to deal with incomplete data (e.g., in applying concealment).

4. IMPLEMENTATION

In order to test performance, we have implemented the error control scheme described in the previous section in a custom datagram transport protocol very similar to UDP. The protocol has been implemented in the NetBSD and SunOS Unix kernels[4]. Currently the protocol runs on Pentium and Sparc class machines. The packet delivery mechanism is IP over both Ethernet and a 155 Mbps ATM network. The protocol currently supports a single retransmission, but work is under way to extend it to support multiple retransmissions (see Section 6). The sending application hands data to the protocol in units of frames. Frames can be arbitrarily large, but the protocol breaks frames into packets before transmission. Retransmission is done in units of packets. The receive end of the protocol estimates the RTD by periodically bouncing timestamped packets off the sending host, and maintaining a smoothed average. The application sets the size (number of frames) of the retransmission and playout buffers and the preferred packet size using the `setsockopt` system call. The application receives frame status information using the `getsockopt` system call.

The operation of the protocol is depicted in Figure 2. On the sending side, the application writes frames to the protocol with period *T*. The protocol breaks the frame into packets and immediately begins transmitting the packets to the receiver. When packet transmission is completed, a copy of the frame is inserted into the retransmission buffer, after discarding the oldest frame in the buffer. At the receiving host, incoming packets are reassembled into frames and

[4]Kernel implementation allows faster loss detection and recovery, but a user-level implementation is certainly possible.

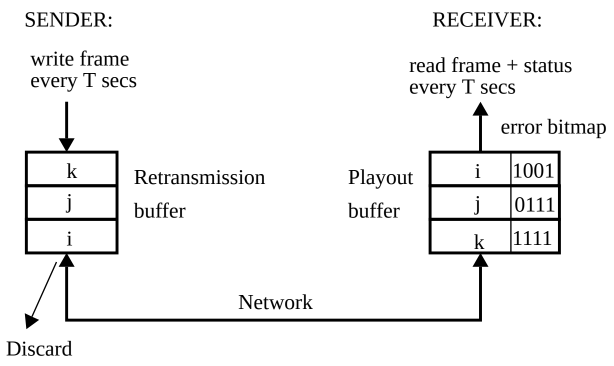

Figure 2: Implementation of error-control scheme

appended to the playout buffer. If a gap is detected during reassembly and if there is sufficient time for recovery, a retransmission request is sent immediately. The application issues read requests for frames with the same period *T* as the sender.

4.1 Processing Overhead

In the common case, (i.e. when there are no losses) the protocol processing is on par with UDP. When there is loss, the protocol processing is comparable to that of other selective repeat protocols [5,7,16]. However, in addition to the processing usually associated with selective repeat protocols, our scheme incurs the following overhead:

Conditional retransmission decision: the receiver requires information about the period of the CM stream and an estimate of the RTD to make retransmission decisions. The former is a parameter passed to the protocol by the application during connection setup. In our implementation RTD estimation is done using timestamped packets and a smoothing function similar to [14].

Retransmission and playout buffer management: The buffer management overhead in our scheme is comparable to that in other selective repeat schemes. The main difference is that our scheme uses a small FIFO queue of buffers rather than a single buffer. However, the extra overhead of managing such a FIFO queue is small.

Playout buffer status update and delivery to application: The playout buffer status consists of a bitmap indicating the presence or absence of packets. Note that such a bitmap is part of the selective repeat implementation. Unlike other selective repeat protocols, in our scheme the bitmap is made available to the application. This involves preparation of the a *frame status*, which is a data structure containing the loss bitmap and the packet size (assuming all packets are of constant size). With the socket interface, an additional system call is required to deliver the frame status to the application (which takes only a few tens of microseconds). Ideally, the application interface should support data and status delivery in a single system call.

5. STATUS AND EXPERIMENTS

To aid in our experimentation, we have implemented a tool to emulate network delay and loss. The tool consists of a machine with a kernel modified to receive packets from the sender, delay and/or drop packets, and finally forward packets to the receiver. The tool is used as shown in Figure 3 to emulate different network delays and loss models. Using this tool we have conducted experiments with different RTD and loss probabilities. In our experiments, we apply the same loss model in both the forward and the reverse direction. This is probably a pessimistic assumption since is assumes both directions are equally congested. Note that this perhaps has a negative impact on performance by dropping too many retransmission requests.

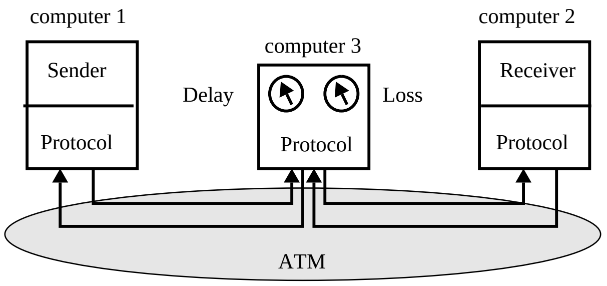

Figure 3: **Protocol evaluation environment**

5.1 Experiment 1: Random loss

We first note that the observed loss rate with one retransmission attempt and random loss can be derived analytically from: $L_{observed} = l^2_{link} \cdot (2 - l_{link})$ where l_{link} is the network loss probability[5]. Thus, for example, if the network loss probability is 10^{-4}, retransmission will reduce the observed loss to 10^{-8}, down four orders of magnitude.

In this experiment we have compared experimental results to those predicted by the above expression. We also investigated the effect of varying the RTD on our protocol. In one sample experiment the packet forwarding machine was configured to randomly drop packets with probability 0.1. This probability was set artificially high to test the protocol. Then 10,000 packets were sent and the observed loss was recorded. The results are shown in Figure 3. The graph shows that our protocol maintained an error rate of about 0.019 as predicted, for round-trip delays less than the playout delay. For higher RTDs there is not enough time to recover, so the observed loss becomes equal to the link loss.

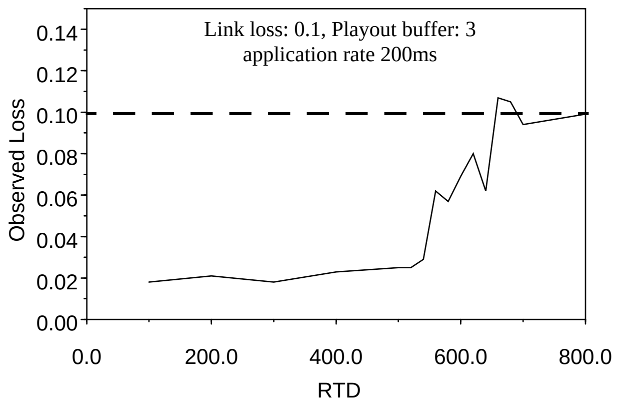

Figure 4: **Random drop experiment**

5.2 Experiment 2: Simple bursty loss

In this experiment we configured the packet forwarder to alternate between two states: a loss state and a no-loss state. A simple model is used to emulate bursty network losses, where the loss and no loss periods are chosen randomly from the intervals: (mean loss period plus or minus loss deviation) and (mean no-loss period plus or minus no loss deviation). We show results in Table 1. The results show that retransmission does decrease the observed error rate and the decrease depends on the duration of the loss and no loss periods.

Table 1: Burst Loss, part A

No Loss Period (ms)	No Loss Dev (ms)	Loss Period (ms)	Loss Dev (ms)	Packet Loss with no RTX	Packet Loss with RTX
50	5	5	2	0.082	0.017
100	20	10	5	0.1	0.0061
100	20	5	2	0.054	0.003
200	20	5	2	0.025	0

5.3 Experiment 3: Bursty loss with MPEG load

In the next experiment we used the loss model described in [19]. This model was derived by simulating several MPEG streams across an ATM video multiplexer. The mean loss period of this model was about 4 ms, and the

[5]Assuming random loss in both directions. The expression is easily derived, so we will not give its derivation here.

mean no-loss period about 115 ms. With this loss model running on the packet forwarder, we run an application transmitting constant-size frames at 20 frames per second. Frame size was 48 KB and packet size was 1KB. The playout buffer was set to 3 frames. A total of 1000 frames were transmitted for each measurement. The experiment has two parts: one with retransmission turned off, and the other with retransmission turned on. Table 2 shows two runs of the experiment without retransmission. The first column shows the percentage loss due to packets being transmitted during the loss period. The second column shows how many bursts were lost during each run and the third column shows the average size of a lost burst in terms of packets. Table 3 shows runs from the same experiment, but this time with retransmissions turned on. Each run was performed with different RTD, which is shown on the first column. The remaining columns are the same as the previous runs. The results show that with retransmission the observed loss is reduced by about an order of magnitude compared to the loss without retransmission. The average burst loss size is quite large, meaning that a significant chunk of the affected frames is lost. The average burst loss size does not seem to change with retransmission, even though the number of lost bursts is significantly reduced. This leads us to conclude that most of the lost bursts with retransmission result from lost requests. This is an indication that multiple retransmission attempts will definitely help in recovering more losses.

Table 2: Burst loss without retransmission

Packet Loss	Number of bursts Lost	Average burst loss size (packets)
0.0253	72	17
0.030	95	15

Table 3: Burst Loss with retransmission

Average RTD (ms)	Observed Packet Loss	Number of bursts Lost	Average burst size (packets)
8	0.0019	8	11
15	0.0013	4	15
21	0.0024	8	14
42	0.0034	11	15
61	0.0075	25	14
82	0.0047	17	13

6. EXTENDING PROTOCOL TO MULTIPLE RETRANSMISSIONS

In the previous sections, we have argued that gap-based loss detection works well with continuous media streams, because gap-based loss detection takes advantage of the continuous flow of data to detect errors quickly, without resorting to timers. While this works well for detecting lost data packets, it does not allow us to detect lost retransmission requests and retransmissions. Therefore, we would like to extend the protocol to attempt multiple retransmissions, while maintaining the fast detection abilities of the original scheme. Again, note that we do not want to use timers, because of the difficulty of accurately estimating the RTD. A pessimistic estimate of RTD leads to increased latency, while an optimistic estimate causes redundant retransmissions.

To extend gap-based loss detection to retransmissions, we add another field to the packet header. This field carries the sequence ID number of the last retransmission request serviced by the sender. Note that the retransmission ID number is totally independent of the packet sequence number, and a single request may be issued for a group of lost packets. The use of the new field is depicted in Figure 3.

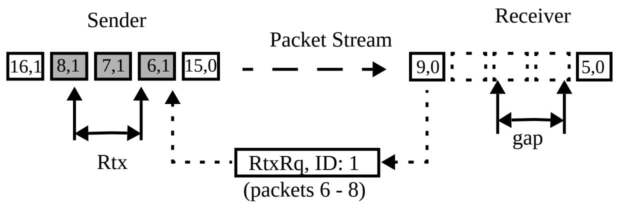

Figure 5: The use of RtxRq ID field

The steps performed by the receiver and the sender are as follows:

Receiver:

- Upon detection of a gap in the CM stream, the receiver performs a retransmission test (i.e., it checks if enough time is available to recover lost data), creates a retransmission request and assigns it a new request ID from a monotonically increasing sequence number space. A copy of the request is saved in a retransmission request stack.
- If the stack contains other outstanding requests, a retransmission test is performed for each one. Since requests are stored in LIFO order, if the retransmission test fails for a request all other requests down in the stack can be safely discarded. All requests for which the retransmission test succeeds are bundled with the new request and immediately sent to the sender.
- Periodically, the receiver traverses the request stack performing a retransmission test as described before. Requests that pass the test are sent again. All others are discarded. The period of this operation is set to be less or equal to the value of RTD.
- When a retransmission is received, the receiver removes the corresponding request from the stack.

- When there are outstanding requests, the receiver examines the retransmission ID field of all incoming data packets. If the retransmission ID field is equal or higher than the next expected retransmission ID but only some or none of the retransmitted packets were received, the receiver performs a retransmission test and sends a request with a new ID for the missing packets. The old request is removed from the stack.

Sender

- The sender keeps a local variable containing the maximum request ID serviced. Upon reception of a request, the sender checks the request ID to ensure that this is a new request. If not, the request is discarded.
- If the request has not been serviced, the sender sends the requested retransmissions and updates its local state, as well as the request ID field in all subsequent data packets.

The most significant cost of this scheme is the additional field in the packet header. Other costs include possible redundant requests that may be sent to the sender. The scheme, however, totally eliminates duplicate and late retransmissions while allowing fast loss detection and recovery. While the scheme still makes use of timer, the value of the timer is not very critical because redundant retransmission requests resulting from an optimistic timer value do not cause redundant retransmissions. Timer values can be adjusted for each individual case: values lower than RTD can be used if recovery time is short or loss rate is high, and values slightly higher may be used otherwise.

We are currently in the process of implementing these changes to our protocol. We will report results in the future.

7. CONCLUSIONS

In this paper we have presented an error control scheme for continuous media applications based on retransmission. We have implemented the scheme in a test datagram protocol. Our experiments have shown that the scheme significantly reduces observed loss without violating the timing constraints of continuous media applications, and has minimal cost. We have also described how to extend the scheme to multiple retransmission attempts. We believe we have presented strong evidence that retransmission can be effective in reducing loss in a wide range of CM applications. We hope that our scheme will be used with other protocols for CM (e.g., RTP [20]).

REFERENCES

[1] Ammar, M., Cheung, S., Li, X., “Improving fairness in Multicast Video Distribution,” Tenth Annual IEEE Workshop on Computer Communications, September 1995.

[2] Biersack, E., “Performance Evaluation of Forward Error Correction in ATM Networks,” ACM Sigcomm ‘92, pp. 248-258, August 1992.

[3] Bolot, J., Turletti, T., “A Rate Control Mechanism for Packet Video in the Internet,” IEEE Infocom ‘94, pp. 1216-1223, June 1994.

[4] Brady, P., “Effects of Transmission Delay on Conversational Behavior on Echo-free Telephone Circuits,” Bell System Technical Journal, Vol. 50, No. 1, pp.115-134, January 1971.

[5] Cheriton, D., “VMTP: Versatile Message Transaction Protocol,” RFC 1045, Stanford University, Feb. 1988.

[6] Tihao Chiang and Dimitris Anastassiou, “Hierarchical coding of digital television,” IEEE Communications Magazine, vol. 32, pp. 38-45, May 1994.

[7] Clark, D., Lambert, M., Zhang, L., “NETBLT: A Bulk Data Transfer Protocol,” RFC 998, MIT, 1987.

[8] Delgrossi, L., Halstrick, C., Herrtwich, R., Hoffmann, F., Sandvoss, J., Twachtmann, B., “Reliability Issues in Multimedia Transport,” Second workshop on High-Performance Communication Subsystems (HPCS’93), Williamsburg, VA, September 1993.

[9] Dempsey, B., Liebeherr, J., and Weaver, A., “On Retransmission-based Error Control for Continuous Media Traffic in Packet-switching Networks,” Technical Report CS-94-09, University of Virginia, Charlottesville, Virginia, Feb. 1994.

[10] Doeringer, W., Dykeman, D., Kaiserswerth, M., Meister, B., Rudin, H., Williamson, R., “A Survey of Light-Weight Transport Protocols for High-Speed Networks.” IEEE Transactions on Communications, Nov. 1990.

[11] Floyd, S., Jacobson, V., McCanne, S., Liu, C., Zhang, L., “A Reliable Multicast Framework for Light-Weight Sessions and Application Framing,” Proc. of ACM Sigcomm ‘95, pp. 342-356, Cambridge MA 1995.

[12] Gong, F., Parulkar, G., An Application-oriented Error Control Scheme for a High-Speed Transport Protocol,” Technical Report WUCS-92-37, Washington University, 1992.

[13] Grossglauser, M., Keshav, S., Tse, D., “RCBR: A simple and Efficient Service for Multiple Time-Scale Traffic”, ACM Sigcomm ‘95, pp.219-230, August 1995.

[14] Jacobson, V., Braden, R., Borman, A., “TCP Extensions for High Performance,” RFC 1323, May 1992.

[15] Lin, J., Paul, S., “RMTP: A Reliable Multicast Transport Protocol”, Infocom ‘96, pp.1414-1424, March 1996.

[16] Netravali, A., Roome, W., Sabnani, K., “Design and Implementation of a High-Speed Transport Protocol,” IEEE Transactions on Communications, Nov. 1990.

[17] Oguz, N., Ayanoglu, E., “Performance Analysis of Two-Level Forward Error Correction for Lost Cell Recovery in ATM Networks”, Infocom ‘95, pp. 728-737, April 1995.

[18] Papadopoulos, C., Parulkar, G., “Implosion Control in Multipoint Transport Protocols”, Tenth Annual IEEE Workshop on Computer Communications, September 1995.

[19] Ramamurthy, G., Raychaudhuri, D., Performance of Packet Video with Combined Error Recovery and Concealment,” Infocom’95, pp. 753-761, April 1995.

[20] Shulzrinne, H., Casner, S., Frederick, R., Jacobson, V., “RTP: a Transport Protocol for Real-Time Applications,” Internet Draft draft-ietf-avt-rtp-07, March 1995.

[21] Masahiro Wada, “Selective recovery of video packet loss using error concealment,” IEEE Journal on Selected Areas in Communications, vol. 7, pp. 807-814, June 1989.