Network Stream Reliability and Synchronization

1. Research Team

Project Leader:	Prof. Christos Papadopoulos, Computer Science
Other Faculty:	Prof. Chris Kyriakakis, *Electrical Engineering* Prof. Alexander Sawchuck, *Electrical Engineering* Prof. Roger Zimmerman, *Computer Science*
Graduate Students:	Genevieve Bartlett, Rishi Sinha

2. Statement of Project Goals

In this project we are working on creating the network infrastructure on which to build a highly accurate, synchronized and loss-free substrate for immersive applications, which can scale to any number of participants located anywhere around the world. We achieve accurate synchronization (in the order of microseconds) by leveraging off the timing capabilities of the *Global Positioning System (GPS*). We achieve real-time operations by utilizing a *Real-time Operating System.* Finally, we provide high-performance, scalable multiparty reliability via a distributed protocol that employs *Network Assistance*. One of the immediate goals of this project is to support a distributed interactive musical performance, where several remotely located musicians can perform together.

3. Project Role in Support of IMSC Strategic Plan

The Remote Multi-channel Immersion (RMI) and its successor, the Distributed Immersive performance (DIP,) are an instantiation of the IMSC vision of immersipresence. RMI and DIP depend on the realization and synthesis of several components. This project important infrastructure needed to realize RMI/DIP, namely support for reliable transmission of multipoint *and* multi-channel haptic, audio and video streams, and a substrate for highly precise synchronization of the constituent streams using GPS timing. In addition, this project provides an infrastructure for bi-directional stream transmission that incurs minimal latency on top of the network latency, to support interactivity. Since latency must be kept low, this project provides for audio concealment and error recovery algorithms that are capable of eliminating the audible artifacts created during loss, which may distract the participants.

4. Discussion of Methodology Used

This project places a strong emphasis on prototyping and experimentation. As a consequence it addresses architectural, design, integration, and implementation issues. The technologies produced by this project have a broad scope, and are also used to support other technologies developed at IMSC. The details of these technologies were described in Volume Two of the Year Seven report, and are listed next:

- Retransmission-based error recovery protocols for continuous media

- Multipoint recovery protocols
- GPS synchronization with Real-Time OS support.
- Very low latency multi-channel audio streaming
- Error concealment of multi-channel uncompressed audio

This year we have developed retransmission-based protocols that employ multiple retransmissions to recover data only while such data has not yet expired. Two versions of these protocols have been designed: (a) a unicast version and a multicast version. Additionally, the concealment techniques developed last year were enhanced to adapt to clock drift between different soundcards. The result is the ability to play back streaming audio that remains synchronized and without audible artifacts when adjustments need to be made.

5. Short Description of Achievements in Previous Years

In previous years we demonstrated a working prototype of the selective retransmission protocol and demonstrated its effectiveness. The protocol attempted one retransmission. The improvements were demonstrated in the Yima server. Measurements showed a significant reduction in effective loss with our protocol in place. In addition, we demonstrated an initial prototype of our synchronization platform with GPS and Real-Time Linux. Measurements showed that our platform is capable of taking actions that are synchronized to within 20 microseconds. The selective retransmission protocol was enhanced with the capability to perform multiple retransmissions. The algorithms to extend the retransmission protocol to multicast environments were developed. An error concealment algorithm was also developed, which uses two techniques to approximate missing samples from an audio stream. The first technique is based on interpolation from multiple channels, and the second on Bezier curves to approximate the missing samples. The techniques were implemented and demonstrations with music showed that the artifacts due to loss were reduced drastically.

5a. Detail of Accomplishments During the Past Year

1. Low-Latency Audio Streaming

Minimizing latency in the capture and playback of audio streams presents problems that need to be overcome at the level of the sound card buffers. Associated with the latency minimization problem is the difficulty posed by the fact that different sound cards normally have some skew between each other due to the fact that the clocks on board these devices do not achieve the same nominal rate. The effect of this second problem is that there exists a mismatch in the rates of production and consumption in an audio stream between two sound cards. This manifests itself at the playback end as periodic under-runs in the amount of data available for playback, or as a steady buildup of the amount of data queued at the receiver. The effect of under-runs is degradation in the quality of the sound in the form of audible clicks. The effect of queue buildup is the loss of data that must result as part of the queue management process. If this loss is not carefully controlled and detected, it too results in audible discontinuities in the sound. If not controlled, it obviously leads to instability in the queue, which may affect the consumption of data in unpredictable ways. For example, if the queue uses a drop-tail discipline, it will drop the most current data when it arrives at a full queue while storing older data in the queue. This is an undesirable situation in a real-time stream, where current data must be given higher priority over old data.

The effect of rate mismatch between sound cards is illustrated in Figure 1 and Figure 2. These figures show the results from experiments we conducted in order to quantify the mismatch in rates between pairs of sound cards. Figure 1 shows the size of the socket queue at the receiver when the actual data rate of the sender is greater than the rate of consumption the receiver. Figure 2 shows the rate of under-run events at the receiver when the actual rate of the sender is less than the rate of consumption at the receiver.

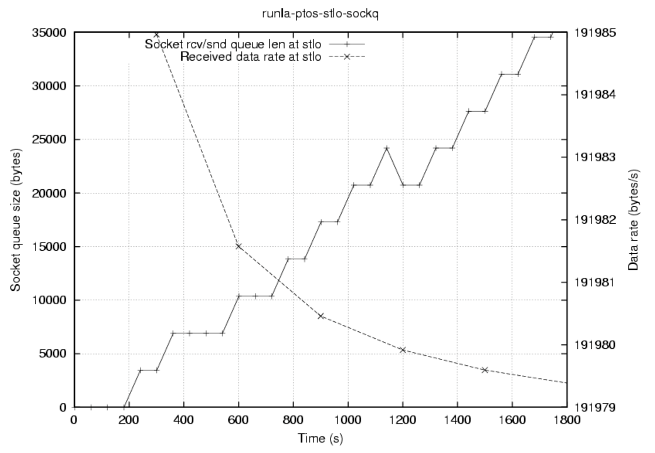

Figure 1: Socket queue buildup - fast sender, slow receiver.

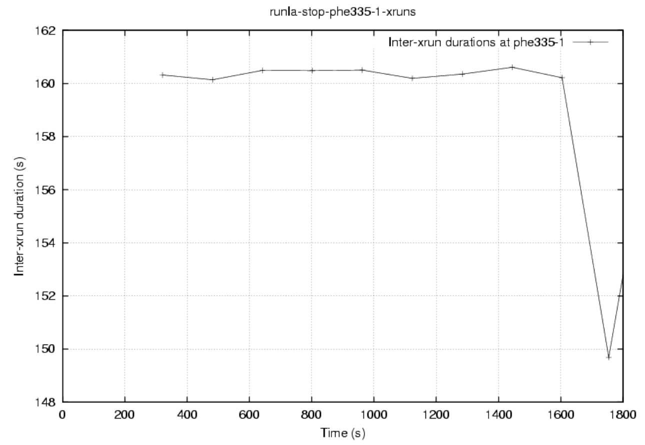

Figure 2: Under-run frequency - slow sender, fast receiver

Figure 1 shows that the size of the socket queue increases steadily with time. It also shows the average rate at which data arrives from the sender, which is seen to be non-increasing. This implies that the consumption of data is slower than the arrival of data.

Figure 2 shows the time interval between successive under-run events over time. It can be seen that this interval is fairly constant. This implies that the receiver runs out of data periodically due to the fact that it consumes data faster than the sender sends it. The quantity of the rate mismatch implied by the first graph is consistent with that implied by the second.

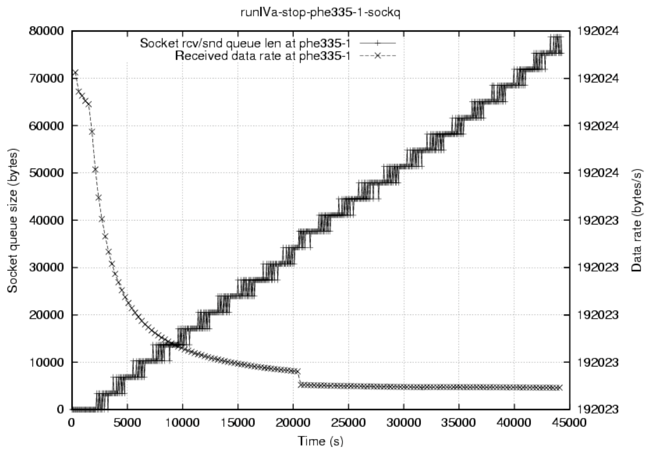

Figure 3: Socket queue buildup - fast sender, slow receiver.

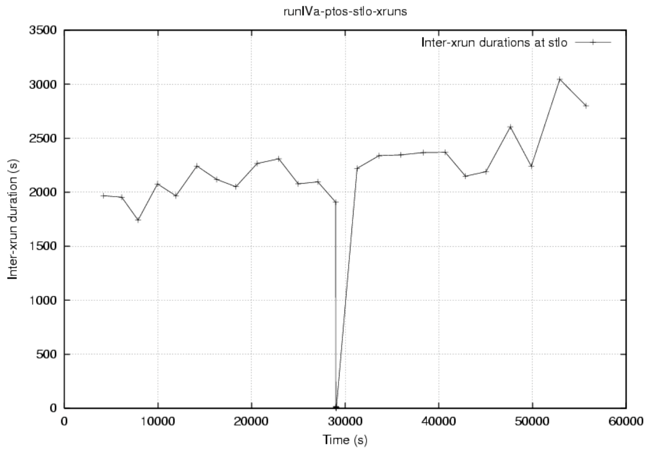

Figure 4: Under-run frequency - slow sender, fast receiver.

Figure 3 and Figure 4 show corresponding graphs for a different pair of sound cards – both of them are high-end multi-channel digital-I/O sound cards. These graphs show qualitatively similar behavior as the previous ones, but the amount of rate mismatch displayed by these cards is much less. Figure 3 and Figure 4 are over a period of over 12 hours, while Figure 1 and Figure 2 are over 30 minutes only. The rate of socket buffer growth is thus seen to be much slower, and the frequency of under-run events is also seen to be much lower in case of these cards. Thus, while more expensive equipment can mitigate the problem, it does not eliminate it.

These findings imply that the subsystem for capture and playback of the audio stream must be carefully designed to minimize latency and adapt to mismatched sound cards. We have implemented a design that minimizes both delay and under-runs. This design aims to always have data available to the sound card buffer, even in case of absence of actual data from the sender, which is a condition that may arise due to the loss of a train of packets, network delay affecting a train of packets, or OS scheduling delay preventing the timely transfer of packets from the socket buffer to the application.

Figure 5 illustrates the operation of this implementation at the receiver of the stream. As packets arrive over the network into the socket buffer, there may be drops in the buffer due to an excess of packets. In addition, when these packets are passed to the application, some of them may be dropped periodically in a controlled way if the sender's sound card runs at a higher realized sampling rate than the receiver's card. The loss concealment module conceals the effect of these losses on the stream before the stream is passed to the sound card for playback. It is also possible that the receiver's sound card runs at a higher realized sampling rate than the sender's card. In this case, instead of dropping packets periodically, the application would have to insert a controlled number of packets periodically, and the loss concealment module would again be called into operation in order to smooth the transitions between packets introduced due to this action.

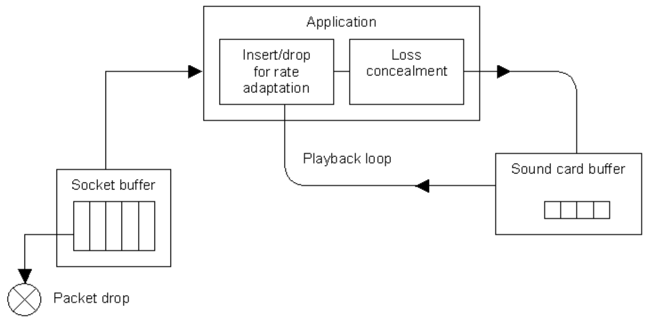

Figure 5: Buffering in playback module.

2. Unicast and Multicast Retransmission Protocols

2.1 Multiple Retransmissions For Unicast

In this section, we present a scheme for multiple retransmissions for loss recovery in unicast applications, abbreviated as MR-UNI. This scheme uses receiver naming of retransmissions. The goal of any multiple-retransmission loss recovery scheme is to detect and recover losses of (a) original transmissions, (b) negative acknowledgments and (c) retransmissions. The two main ideas of this mechanism are:

- Second sequence space. MR-UNI uses a second sequence number space in order to detect lost retransmissions and negative acknowledgments while using the first sequence number space to detect lost original transmissions as usual.
- Redundant negative acknowledgments. To decrease the probability of lost negative acknowledgments, the MR-UNI receiver resends negative acknowledgments a number of times in a deliberately redundant fashion. The number and spacing of these repetitions can be tuned to achieve the desired balance between loss recovery and network and processor load. In addition, there is added redundancy in the sending of negative acknowledgments in that the transmission of any negative acknowledgment by the receiver also causes all other currently pending negative acknowledgments to be sent.
-

The second sequence space is a sequence number for numbering negative acknowledgments (NACKs). Each NACK carries a NACK sequence number. When the sender retransmits a frame in response to a NACK, it is said to have serviced the sequence number of that NACK. The rules for assigning and servicing NACK sequence numbers are:

1. The receiver increments a counter for every unique NACK it sends, and numbers the NACK with the value of this counter.
2. The receiver may send duplicate copies of a NACK. These will naturally have the same NACK sequence number, and are not "unique" NACKs.
3. The sender services each NACK sequence number only once; duplicates are ignored.

4. The receiver may assign a new number to a NACK sent previously. This creates a fresh NACK, and is numbered according to rule 1.
5. The sender must service each new NACK sequence number it sees, regardless of the frame being requested for retransmission.
6.

On detecting a gap in the NACKSEQNO stream arriving from the sender, the receiver concludes that the negative acknowledgments belonging to the gap did not result in successful retransmissions (because of the loss of either the negative acknowledgments themselves or the retransmissions). In such an event, the receiver assigns new identifying sequence numbers to the negative acknowledgments in the gap, essentially regenerating negative acknowledgments for the same pieces of media data. This is different from simply resending copies of the existing negative acknowledgments. Note that negative acknowledgments will never be sent for data that is not expected to arrive by its playback time based on the current estimates of round-trip time.

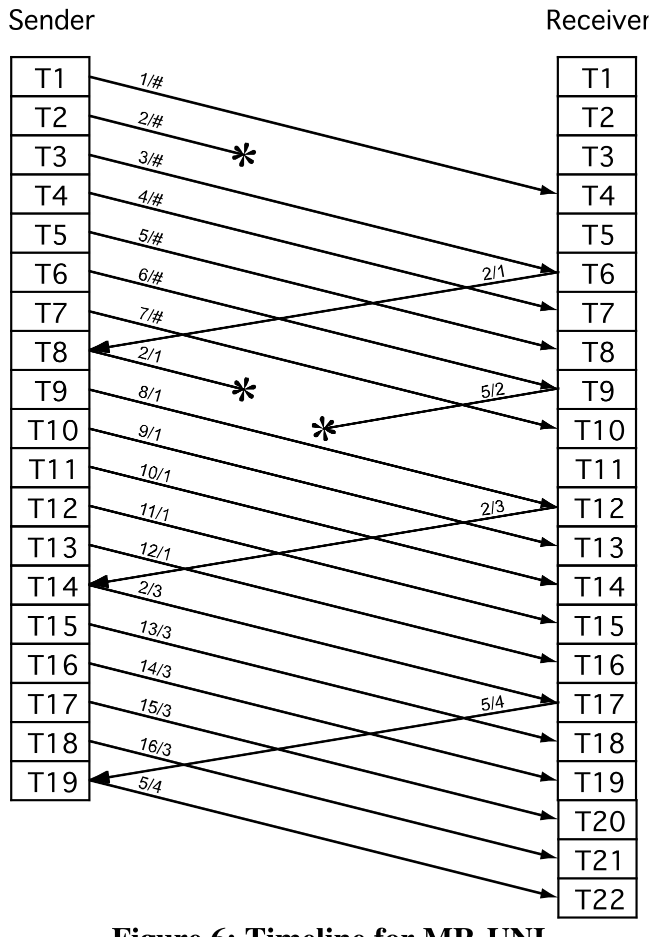

Figure 6: Timeline for MR-UNI

To see this process with an example, consider Figure 6. This shows the loss of retransmissions and negative acknowledgments, with a regeneration of the negative acknowledgment for the lost data.

Figure 6 displays timelines for the generation and consumption of media data and negative acknowledgments. The notation x/y indicates SEQNO/NACKSEQNO. Losses are shown as interrupted arrows. The negative acknowledgments 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 negative acknowledgment sequence number. Note that this second kind of discontinuity can arise either due to the loss of a retransmission (detected at T12) or because of the loss of a negative acknowledgment (detected at T17).

The last statement highlights the significant difference between the two sequence number streams from the sender (SEQNO and NACKSEQNO). While the first is bound to change as new media data is generated and transmitted, the second incurs a change only due to continued losses. This is a significant fact about the MR-UNI protocol as described so far: we need further losses for assured loss detection. More specifically, loss of retransmissions is discovered without further losses, but loss of negative acknowledgments can go undetected without further losses. For example, in Figure 6, the loss of negative acknowledgment 5/2 (time T10) would go undetected if frame 2/1 were not lost (time T10).

To reduce the probability of such undetected losses of negative acknowledgments, we redundantly transmit negative acknowledgments in a number of repetitions exponentially spaced apart in time. In addition, transmission of any negative acknowledgment is accompanied by the transmission of all other pending negative acknowledgments. The reason for this is that it can speed up the detection of lost negative acknowledgments and actually fetch in time data that might otherwise be recovered too late. The algorithm for MR-UNI is described in Figure 7.

Definitions:

NACK sequence number:
Every negative acknowledgment is assigned a sequence number by the receiver. These sequence numbers form a different space than the frame sequence numbers. The receiver always assigns the NACK sequence number to negative acknowledgments in monotonically increasing fashion.

Stale sequence number:
A sequence number whose playback time has expired (receiver-side) or which is too old to be stored any longer (sender-side).

Covered sequence number:
A sequence number between the least (i.e., oldest) and greatest (i.e., last) sequence numbers stored in the receiver's buffer, both inclusive. The covered region may include gaps in the sequence number space.

Early sequence number:
Any sequence number greater than the greatest sequence number stored in the receiver's buffer but small enough to be accommodated. Note that the definition of early includes the next expected sequence number (ignoring expected sequence numbers in gaps).

Future sequence number:
Any sequence number too large to be accommodated in the receiver's buffer.

Formats:

Sender:
Each outgoing frame (denoted as FRAME) has these two fields: SEQNO, NACKSEQNO. SEQNO is the sequence number of this frame. NACKSEQNO is either NULL or the highest NACK sequence number serviced by the sender so far, if any.

Receiver:
Each negative acknowledgment (denoted as NACK) has these two fields: SEQNO, NACKSEQNO. SEQNO is the sequence number of the frame desired. NACKSEQNO is the NACK sequence number for this NACK. The receiver always assigns the NACKSEQNO to negative acknowledgments in monotonically increasing fashion.
The receiver buffers frames to allow time for possible retransmissions. The record in this buffer for sequence number x is denoted as RECVBUF[x] and is undefined if x is receiver-side stale or future. Each record has these two fields: EMPTY,

NACKSEQNO. EMPTY is 1 if the record does not contain a frame and 0 if it does. NACKSEQNO is valid if EMPTY = 1, and is the sequence number associated with the last negative acknowledgment sent for x.

Variables:

Receiver:
RECVSENDNACKS. This flag is marked to 1 if sending of negative acknowledgments is required. The receiver always sends all negative acknowledgments when any one or more needs to be sent.
LASTNACKSEQNO. This variable holds the NACKSEQNO from the last frame received, if it was non-NULL.

Algorithm:

Sender:
[a] To send a frame for the first time:
SEQNO contains the sequence number of this frame. If the sender has sent no retransmission so far, NACKSEQNO contains NULL.
Otherwise, NACKSEQNO contains the highest sequence number serviced by the sender so far, including this frame.
Buffer the frame for future retransmissions, and eject the oldest frame in the buffer.
Sleep.
[b] On receiving a negative acknowledgment:
Locate NACK.NACKSEQNO in the retransmission history.
If found, sleep.
If not found, locate NACK.SEQNO among the buffered frames.
If not found, sleep.
If found, do:
{add a record for NACK.NACKSEQNO to the retransmission history; send a frame with FRAME.SEQNO = NACK.SEQNO and FRAME.NACKSEQNO = the highest NACK sequence number serviced so far, including this retransmission}.
Sleep.

Receiver:
[a] On receiving a frame:
Set RECVSENDNACKS = 0.
If RECVBUF[FRAME.SEQNO] is not defined, sleep.
Else, accept the frame into RECVBUF[FRAME.SEQNO] and set RECVBUF[FRAME.SEQNO].EMPTY = 0.
If FRAME.SEQNO is early and not the next expected sequence number, for each sequence number x in the gap, do:
{set RECVBUF[x].EMPTY = 1; assign a new value for RECVBUF[x].NACKSEQNO; set RECVSENDNACKS = 1}.
If FRAME.SEQNO is early, pop the appropriate number of frames from the receiver's buffer.
If FRAME.NACKSEQNO is non-NULL and is greater than LASTNACKSEQNO, then for each i after LASTNACKSEQNO and till FRAME.NACKSEQNO, do:
{determine the sequence number x corresponding to NACK sequence number i; assign a new value for RECVBUF[x].NACKSEQNO; set RECVSENDNACKS = 1}.
If RECVSENDNACKS = 1, execute [b] below.
Sleep.

[b] Sending all negative acknowledgments:
For every covered sequence number x, do:
{if RECVBUF[x].EMPTY = 1, send a negative acknowledgment with NACK.SEQNO = x and NACK.NACKSEQNO = RECVBUF[x].NACKSEQNO}.
RECVSENDNACKS = 0.
Set the timer.

[c] Timers:
Step [b] is repeated when a timer expires. The number of repetitions and the timeouts are not specified in this algorithm.

Figure 7: Algorithm for MR-UNI

2.2 Multiple Retransmissions For Multicast

Since multicast is the most natural method for distributing a media stream to multiple receivers, efficient multiple retransmission for multicast is an important part of the real-time interactive infrastructure we propose. In this section we describe a multiple retransmission scheme for multicast that is based on the same principle 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 of a multiple retransmission scheme for multicast is to use multiple instances of the unicast protocol MR-UNI, running one instance per receiver. This is not scalable to a large number of receivers, since the sender is forced to keep track of the NACK sequence number state for each receiver. This leads us to examine another simplistic scheme, where each individual receiver still follows the MR-UNI protocol independently, but the sender does not maintain state about each receiver's NACK sequence numbers. The sender only keeps track of the last retransmission serviced and the NACK sequence number associated with it. This simple method does not work, since receivers will number their negative acknowledgments with sequence numbers chosen independently of each other. The sender's feedback about NACKs serviced has meaning for only one receiver.

We characterize this as a naming issue. We divide the naming methods into two classes – receiver naming and sender naming. In receiver naming, the receiver labels each NACK with a sequence number or possibly other kind of name, and these names are used to control the process of multiple retransmission. In sender naming, the same control is achieved with names assigned to retransmissions sent by the sender.

In the following discussion, we will consider names assigned as follows:

- Sender naming: if a retransmission is the n^{th} time the frame with sequence number SEQNO is retransmitted from the sender, the sender names that retransmission with the label (SEQNO, n).
- Receiver naming: a receiver names each unique NACK it sends out with the value of a counter. This is the same as the NACK sequence number described for MR-UNI.

We have already seen how receiver naming can be used to control multiple retransmissions in unicast. Sender naming can be used to achieve similar goals if the sender sends along with each data frame the following piece of information – the name of the last retransmission made at the sender. This stream of information provides an equivalent amount of information to the receiver as in MR-UNI, with the two following exceptions:

- Receiver naming provides information to the sender about losses of NACK. Since NACKs are numbered sequentially in receiver naming, the loss of a NACK can be detected at the sender. (Only the loss event can be deduced, not the contents of the NACK, so this information cannot be used to immediately recover from the NACK loss).
- For the sender-naming scheme described above to be equivalent to MR-UNI, we must assume that NACKs are not reordered, and they are serviced in the order that they arrive at the sender.

Thus, with some caveats, it is possible to achieve multiple retransmissions in unicast with either sender naming or receiver naming. However, the situation in multicast is quite different. We saw before that a straightforward application of the ideas of MR-UNI do not work in multicast, and this is because it uses a receiver naming scheme to control retransmissions.

We propose a multicast recovery scheme called MR-MCAST, based on sender naming. We will demonstrate that sender naming solves the problem of multiple retransmissions in multicast.

Towards constructing this protocol, we consider the sender-named protocol described above, i.e., the sender includes with each data frame the name of the last retransmission made. We assume that retransmissions are multicast and NACKs are unicast. Let us apply this to the simple scenario of two receivers, R1 and R2, and a sender S. Let us assume that receiver R1 did not receive frames with sequence numbers 3 and 5. It would send out NACKs for these frames. Let us say that after some time, it sees a retransmission of frame 5 and then starts to see the name (5,1) being advertised by the sender. This of course means that the last retransmission made by the sender was for frame sequence number 5. In a unicast scenario with the assumption of in-order delivery, R1 could safely assume that the retransmission of frame 3 was unsuccessful, either because the NACK was lost, or the retransmission was lost. This is what made it possible to implement MR-UNI using sender naming as described above. However, in a multicast scenario, R1 cannot make this conclusion, even with the assumption of in-order delivery. This is because the following events may also give rise to this same observation of name (5,1) at R1: The other receiver R2 did not receive frame 5, and sent a NACK for it. R2 is much closer to the sender, and its NACK reached the sender before either of R1's NACKs. Thus, R1 sees the name (5,1) before it sees (3,1). Thus, there is not enough information at R1 to draw a conclusion about its NACK for frame 3.

This leads us to the ideas we use to overcome this problem. In the MR-MCAST scheme we propose, there are two main ideas, and we will show how we deal with the above problem using these mechanisms:

- Sender naming (as described above). The sender includes with each frame of data explicit information about which sequence numbers were retransmitted in the past and how many retransmissions were made. There is no second sequence number space as in MR-UNI.
- NACK cycling. At a time, the sender may have information about numerous past retransmissions. We intend to convey all of this information to the receivers in order to resolve the ambiguity mentioned above. Since there is no fixed number of these units of information, it is not feasible to design to include every single one of them in every outgoing data packet. Instead, we again use the fact that there is a continuous stream of frames emanating from the sender, and we distribute all this information across various frames of data. We call this sender-side process 'NACK cycling,' since the sender decides to include a fixed maximum number N of units of information in each frame, and cycles through the pool of information, distributing it among various frames. The number N is chosen keeping in mind the minimum path MTU in the multicast group, in order to avoid fragmentation of packets.

As a result of adding the above enhancement to the simple sender-named protocol, the receivers have available to them enough information about the retransmissions to detect lost retransmissions and NACKs. Also, this sender-named protocol obviously applies to unicast too, since that is just a special case of multicast. However, the extra information provided (compared to MR-UNI) is not necessary unless we expect the assumption of in-order delivery to be violated severely. Thus, the simpler MR-UNI suffices for unicast.

Also, in MR-MCAST, NACKs are unicast to the sender and retransmissions are multicast to all receivers. A mechanism is included in the protocol, described below, that suppresses unnecessary responses to multiple NACKs for the same frame. Since retransmissions are multicast, this takes care of loss recovery over all receivers despite NACK suppression.

However, since the suppression of NACKs is at the sender rather than at some point between the receiver and the sender, there is a potential problem of NACK implosion at the sender. There are two reasons why we allow this scheme of NACK transmission despite the possibility of this problem.

1. We expect the multicast groups to consist of a small enough number of participants that the implosion of NACKs will not be a severe problem. For example, a distributed orchestra is expected to have tens of participants rather than thousands.
2. We forgo protection against NACK implosion in favor of speed of recovery of lost packets.

However, if it becomes necessary to combat the problem of NACK implosion, it is possible to employ a hierarchical scheme for NACK delivery to the sender, which will 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.

We will now describe the MR-MCAST protocol more concretely. The receiver is given direct information about past retransmissions, and can very easily judge whether a request for retransmission needs to be repeated. Figure 8 shows a timeline for an exchange between a sender and two receivers (R1 and R2), with losses and recovery illustrated. We take the case of N = 1. The following are the events of interest in the figure:

- The loss of frame sequence number 2, sent at T2. Neither receiver receives it.

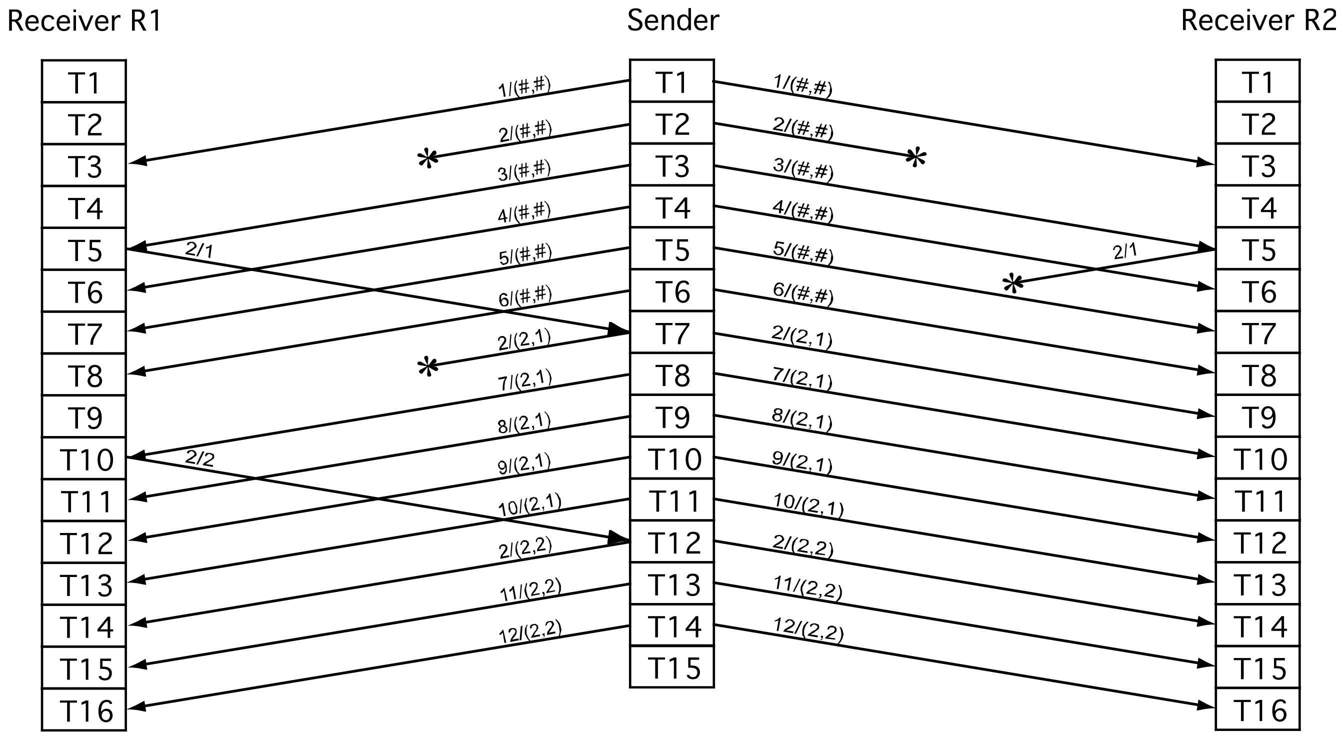

Figure 8: Timeline for MR-MCAST

- The generation of NACKs by T5 at both receivers because of this loss. The NACKs are unicast.
- The loss of R2's NACK.
- The arrival of R1's NACK at the sender at T7, and the subsequent retransmission for the requested frame. The retransmission is multicast.
- The receipt of this retransmission by R2 at T9, despite the fact that R2's NACK was lost.
- The loss of this retransmission for R1.
- The generation of another NACK for the frame by R1 at T10.
- The arrival of this NACK at the sender at T12 and the repeat retransmission of the frame.

In MR-MCAST, the sender responds to a negative acknowledgment - of the form SEQNO/REPS – only if it has the frame buffered and its record of past retransmissions for this frame shows that REPS exceeds the number of retransmissions that have been made for this frame. The fact that retransmissions are multicast is used to suppress unnecessary retransmissions for the same frame by rejecting NACKs with a REPS count that is too low. (For example, in the figure, if the T5 NACKs from both R1 and R2 did arrive at the sender, only the first one would get serviced, because they both bear the same REPS count.) Receivers choose REPS so as to ensure they get a retransmission, as is illustrated for receiver R1 at time T10.

Note that the loss of a negative acknowledgment is still a problem – the loss of a negative acknowledgment is not detected without further losses. For this reason, we still need the protective redundancy mechanisms of MR-UNI. (This topic is more completely discussed in the next subsection.) However, there is already some redundancy built into MR-MCAST with respect to NACK generation, which helps reduce the severity of the problem of undetected NACK loss. This is due to the fact that 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 for all the receivers to receive a retransmission. This is illustrated in the figure by fact that R2 receives its retransmission despite the loss of the NACK it sent at T5. This is due to R1's NACK successfully arriving at the sender. The algorithm for MR-MCAST is described in Figure 9.

Definitions:

Stale sequence number, *covered sequence number*, *early sequence number* and *future sequence number*: as defined in Figure 7.

Formats:

Sender:

Each outgoing frame (denoted as FRAME) has these three fields: SEQNO, PASTSEQNO[1...N], PASTREPS[1...N]. SEQNO is the sequence number of this frame. For each x in 1...N, PASTSEQNO[x] is a sequence number that has been retransmitted sometime in the past and PASTREPS[x] is the number of times it has been retransmitted so far. For each x in 1...N, PASTSEQNO[x] must not be sender-side stale. NULL values can be indicated in PASTSEQNO[x] and PASTREPS[x] for any x in 1...N.

The sender buffers frames for possible retransmissions. The record in this buffer for sequence number x is denoted as SENDBUF[x]. Each record has this field: REPS. This is the number of times this frame has been retransmitted.

Receiver:

Each negative acknowledgment (denoted as NACK) has these two fields: SEQNO, REPS. SEQNO is the sequence number of the frame desired. REPS is a number chosen by the client to try to ensure the frame gets retransmitted and it is chosen according to the algorithm given below.

The receiver buffers frames to allow time for possible retransmissions. The record in this buffer for sequence number x is denoted as RECVBUF[x] and is undefined if x is receiver-side stale or future. Each record has these two fields: REPS, EMPTY. REPS is either NULL or equal to the NACK.REPS field from the last negative acknowledgment sent for x, if any. EMPTY is 1 if the record does not contain a frame and 0 if it does.

Variables:

Receiver:

RECVSENDNACKS. This flag is marked to 1 if sending of negative acknowledgments is required. The receiver always sends all negative acknowledgments when any one or more needs to be sent.

Algorithm:

Sender:

[a] To send a frame for the first time:
SEQNO contains the sequence number of this frame. RETXSEQNO[1...N] and RETXREPS[1...N] contain as much information about past retransmissions as the sender chooses to send. The details of this are not specified in this algorithm.
Buffer the frame for future retransmissions, and eject the oldest frame in the buffer.
Set SENDBUF[FRAME.SEQNO].REPS = 0.
Sleep.
[b] On receiving a negative acknowledgment:
Locate NACK.SEQNO among the buffered frames.
If not found, sleep.
If found, and if SENDBUF[NACK.SEQNO].REPS < NACK.REPS, then increment SENDBUF[NACK.SEQNO].REPS by one and send a frame with FRAME.SEQNO = NACK.SEQNO and RETXSEQNO[1...N] and RETXREPS[1...N] containing as much information about past retransmissions as the sender chooses to send.
Sleep.

Receiver:

[a] On receiving a frame:
Set RECVSENDNACKS = 0.
If RECVBUF[FRAME.SEQNO] is not defined, sleep.
Else, accept the frame into RECVBUF[FRAME.SEQNO] and set RECVBUF[FRAME.SEQNO].EMPTY = 0.
If FRAME.SEQNO is early and not the next expected sequence number, for each sequence number x in the gap, do:
{set RECVBUF[x].EMPTY = 1 and RECVBUF[x].REPS = 1 and RECVSENDNACKS = 1}.
If FRAME.SEQNO is early, pop the appropriate number of frames from the receiver's buffer.
For each i in 1...N, if FRAME.PASTSEQNO[i] is non-NULL and is covered and RECVBUF[FRAME.PASTSEQNO[i]].EMPTY = 1 and RECVBUF[FRAME.PASTSEQNO[i]].REPS <= FRAME.PASTREPS[i], then do:
{set RECVBUF[FRAME.PASTSEQNO[i]].REPS = FRAME.PASTREPS[i] + 1; RECVSENDNACKS = 1}.
If RECVSENDNACKS = 1, execute [b] below.
Sleep.
[b] Sending all negative acknowledgments:
For every covered sequence number x, do:
{if RECVBUF[x].EMPTY = 1, send a negative acknowledgment with NACK.SEQNO = x and NACK.REPS = RECVBUF[x].REPS}.
RECVSENDNACKS = 0.
Set the timer.
[c] Timers:
Step [b] is repeated when a timer expires. The number of repetitions and the timeouts are not specified in this algorithm.

Figure 9: Algorithm for MR-MCAST

2.3 Dealing with NACK Loss

We have seen in the description of the MR-UNI and MR-MCAST protocols that loss of packets flowing from the sender to the receiver(s) can be quickly detected. The loss of a first-time transmission is detected when a subsequent packet arrives at the receiver and informs it of a gap in the frame sequence number stream. The loss of a retransmission is detected when a subsequent packet arrives at the receiver and informs it that the sender made the retransmission. (The exact way in which this information is conveyed varies between MR-UNI and MR-MCAST. In MR-UNI, it appears as a gap in the stream of NACK sequence numbers from the sender. In MR-MCAST, it appears explicitly as the number of retransmissions made for the lost frame.)

Now consider the case of NACK loss. This is loss of data flowing from the receiver to the sender and is, in general, more difficult to detect at the receiver. Unless specific safeguards are present, the loss of a NACK in MR-UNI will not be detected at the receiver until the next loss of a packet from the sender to the receiver. Since such a loss may never occur, the NACK loss may go undetected. The situation is similar in MR-MCAST, except that is somewhat mitigated by the existence of multiple receivers that may send a NACK for the same frame. If at least one of these NACKs successfully reaches the sender, the loss of other NACKs will be overcome.

This subsection deals with the safeguards we put in place to prevent undetected NACK loss. Before we describe them, we illustrate detectable and undetectable NACK losses in MR-UNI without these safeguards. The examples in MR-MCAST are similar.

Detected NACK loss: a NACK loss is followed by the loss of a first-time transmission or retransmission from the sender. This will cause the receiver to generate a new NACK. If this NACK is not lost, the sender will service it. The resulting gap in the NACK sequence number stream will inform the receiver about the loss of the first NACK. (Note that the sender may also be able to detect the loss of the first NACK when it sees the second one. However, this does not have significant benefits, and requires NACK ordering assumptions; thus we do not use this.) On the other hand, if the second NACK is also lost, both losses will be undetected if no further losses occur.

Undetected NACK loss: a NACK loss is the last loss event in a session. The loss will not be detected at the receiver because the NACK sequence number stream from the sender does not change. It is evident that the loss will not be detected at the sender either. We refer to this as an isolated NACK loss.

The solution to the problem of undetected NACK loss lies in the following:

1. Timeout for NACKs. Though the basic recovery algorithms still work without timers as described, we use a timer at the receiver to resend NACKs when the retransmission is not received after some threshold interval. This is the basic means of recovering from isolated NACK loss. Issues that need to be resolved here are the calculation of the timeout interval and the consideration for implosion in multicast. The actual method of calculation of round-trip time in multicast is an issue, as the nature of the traffic is not request-reply, and the addition of request-reply probes for timeout estimation may be a burden on the sender, depending on the size of the multicast group.
2. Cumulative NACKs. With this enhancement, NACK transmissions are cumulative, i.e., with every NACK, the receiver repeats the active NACKs it is currently waiting for responses to. This enables a quicker recovery when the NACK loss is followed by further losses. However, it does not provide recovery from or detection of isolated NACK loss.
3. Redundancy in NACK transmission. It may be practical to send multiple copies of a NACK to increase the probability that a NACK will reach the sender successfully. Since NACKs are small, this may be feasible.

6. Other Relevant Work Being Conducted and How this Project is Different

There are several methods for error recovery for streaming media. Perkins et al. [1] present a good summary of such techniques. These techniques, however, use timer-based protocols, which are susceptible to jitter. Our technique does not rely on timers to decide when to send a retransmission. Finally, we are not aware of prior work to minimize recovery latency using retransmission for multicast.

7. Plan for the Next Year

We have implemented and evaluated the unicast version of our protocol. We will do the same for the multicast version and test it in the laboratory and the wide-area Internet. The protocols will be used to conduct psycho-acoustic tests to evaluate their utility and effectiveness. The protocols will be tested with HDTV streaming. Additionally, we will investigate FEC protocols for error recovery.

8. Expected Milestones and Deliverables

We expect to deliver implementations for protocols for unicast and multicast error recovery. In addition, we expect to integrate these protocols with the concealment methods we have developed last year, to deliver an integrated solution that deals with synchronization, concealment and error recovery while minimizing latency.

9. Member Company Benefits

Our work utilizes equipment donated by Intel, IBM, Sun, and others. Similarly, the experiments are carried out on equipment donated by the same companies. Implementations of our protocols are available to Member companies.

10. References

[1] Colin Perkins, Orion Hodson and Vicky Hardman, A Survey of Packet Loss Recovery Techniques for Streaming Media, IEEE Network Magazine, September/October 1998.