Real-Time Streaming of Multichannel Audio Data over Internet[†]

Aoxiang Xu[1], Wieslaw Woszczyk[2], Zack Settel[2], Bruce Pennycook[2],
Robert Rowe[3], Philip Galanter[4], Jeffrey Bary[4], Geoff Martin[2], Jason Corey[2],
Jeremy R. Cooperstock[1,*]

* Corresponding author

1. Centre for Intelligent Machines and Department of Electrical and Computer Engineering
   McGill University, Montreal, QC, H3A 2A7, Canada +1-514-398-5992,
   {axu | jer}@cim.mcgill.ca
2. Music, Media and Technology Group, Faculty of Music, McGill University,
   555 Sherbrooke West, Montreal, QC, H3A 1E3, Canada
   {woszczyk | settel | pennycook | martin | corey}@music.mcgill.ca
3. Music Technology Program, Department of Music and Performing Arts Professions,
   New York University,
   robert.rowe@nyu.edu
4. Arts Technology Group, New York University, 251 Mercer, New York, NY 10012, U.S.A.
   {galanter | bary}@nyu.edu

Abstract

On September 26, 1999, a musical performance, taking place at McGill University, was transmitted to an audience at New York University, over the Internet. While Internet streaming audio technologies have been in use for several years, what made this event unique was the audience's experience of uninterrupted, intermediate quality, multichannel audio (AC-3). In order to achieve this result, a custom system was developed employing both TCP and UDP protocols, and providing its own buffering and retransmission algorithms. The motivation for this approach is explored, and experiments justifying the decisions made are explained.

[†] Appeared in Journal of the Audio Engineering Society,July-August, 2000.

1 Introduction

The growth of the Internet has not only reshaped our work lives, but, in recent years, has also begun to affect and redefine various areas of entertainment, in particular, that of music. Until the last decade, the idea of downloading music to one's home, at great convenience and essentially zero-cost, would have been unthinkable to many. Today, real-time Internet music streaming, such as that available from RealNetworks (realnetworks.com) and MP3 (mp3.com), is regularly enjoyed by millions of listeners.

The limiting factor in such transmissions is typically the available bandwidth of the underlying networks. As a result, Internet audio streaming technologies are invariably characterized by relatively low-bandwidth, low quality, stereo sound. Although this may satisfy the minimal requirements of casual audiences, it is by no means sufficient for a society of musicians and professional audio engineers [1], to whom high-fidelity, multichannel audio is a necessity. Fortunately, the recent emergence of advanced research networks provides an opportunity to explore the possibilities of real-time streaming of high quality, high bandwidth, multichannel audio over the Internet.

The remainder of this paper describes our efforts in designing such a system. Using this technology, we carried out two successful demonstrations, one for the 107th Audio Engineering Society (AES) Convention in New York and the second for the 1999 Canarie Advanced Networks Workshop in Toronto. Considering the emergence of megabit ADSL and cable-modem links to the home, we note that these trials are not merely fanciful research experiments, but rather, likely prototypes of the next generation of consumer entertainment delivery mechanisms.

2 The Demonstrations

The first demonstration, on September 26, 1999, involved the transmission of AC-3 audio accompanied by a separate MPEG-1 video stream, requiring a total bandwidth of approximately 3 Mbps. A live performance of the McGill University Swing Band, playing at McGill's Redpath Hall, was delivered to an audience at the Cantor Film Center of New York University, with a time delay of approximately three seconds.[1] The second demonstration, on November 28, 1999, delivered from McGill University to the Metropolitan Toronto Convention Center, repeated the setup used in New York, but decreased the time delay to approximately one second, thereby illustrating the potential of the technology for reliable, real-time transmissions. At the same time, we generated an additional 12 Mbps of competing traffic between the two sites to test the system's robustness to congestion.

An additional experiment, employing six channels of uncompressed, linear coded audio, 24 bits per sample at 96 kHz, could not be carried out due to hardware limitations, as discussed in Section 2.1.2.

[1] A highly conservative 15-second delay was employed during the first half of the performance, but this was later reduced to three seconds as the authors attempted to demonstrate the importance of buffering. However, due to its robust retransmission algorithm, the system continued to deliver lossless audio, despite competing traffic on the network, even as the accompanying MPEG-1 video stream (Cisco IP/TV) exhibited occasional dropped frames.

2.1 Hardware Configuration

Having provided the background for this research, we now turn to a more detailed description of the hardware involved in each of the experimental configurations.

2.1.1 AC-3 Compressed Audio Transmission

Figure 1 illustrates the experimental setup used in the AC-3 demonstration. The performance at McGill University was fed into a Sony PCM-800, which converted the six channels of analog input to PCM at 16 bits per channel, 48 kHz, which were in turn provided to a Dolby DP569 encoder via three AES/EBU streams. The resulting AC-3 data, encapsulated in an AES/EBU stream at 1.536 Mbps, was read by the *sender* program running on a Silicon Graphics Indy (R4600 with 64 Mbytes RAM). The sender was responsible for segmenting the audio data into discrete packets and transmitting them over the network, to another Indy R4600 with 150 Mbytes RAM), located at the Cantor Film Center (or the Toronto Convention Center). A *receiver* program, running on this machine, read the packets from the network, extracted the audio data, and provided it via the Indy's AES/EBU port, to a Dolby DP562 decoder, which decompressed the AC-3 data and delivered the audio to a playback system.

2.1.2 Uncompressed Audio Transmission

The setup for the uncompressed audio experiment called for the six channel audio sources to be connected to A/D converters manufactured by Data Conversion Systems (dCS), whose outputs are integrated by a prototype APX encoder unit. The encoder packs the resulting audio stream into a flexible data format, discussed in the following section, which is then read by the *sender* program running under Linux on a Pentium III PC.

The hardware design used a high-speed parallel port connection to transfer uncompressed audio data to and from the APX unit. Unfortunately, the transfer rates achieved with various parallel port configurations was found to be substantially less than that advertised by manufacturers. In practice, parallel port I/O consumed all available clock cycles, leaving no time for the transmission (or reception) of data through the network card, nor the servicing (or generation) of retransmission requests. A new version of this hardware, currently in development, will replace the parallel port interface with a Firewire port, thereby allowing for lower I/O overhead. Although hardware limitations have so far prevented us from demonstrating an end-to-end system in operation, our network protocol achieved reasonable performance for the transport of simulated uncompressed audio data between the two sites.[2]

The sender program is responsible for transmitting the data over the Internet to a receiver program, running on an identical PC, where the audio data is extracted and delivered to an APX decoder, which unpacks the data and passes the streams to D/A converters for playback.

[2] It is our intention to carry out a demonstration of this system at the upcoming 109th AES convention.

2.2 Audio and Video Format

The AC-3 audio data (encoded Dolby Digital 5.1) [2] is encapsulated in a stereo AES/EBU (48 kHz, 16 bit) stream provided by the Dolby Encoder. AC-3 carries five full-range channels and one subwoofer, but because of compression, does not require the full channel capacity[3] of AES/EBU. [3] While it is therefore possible to reduce bandwidth demands by discarding the padded zeros of the AES/EBU stream, we chose not to do so, as our research goal involves higher bandwidth applications, such as the transmission of uncompressed six-channel PCM data.

For transmission of uncompressed audio data, the APX encoder reads all six channels of data from the A/D converters and packs them into custom data blocks, pictured in Figure 2. Each block includes 192 samples of the six-channel audio data, arranged in layers of successively decreasing sample resolution. In this manner, the first layer consists of the most significant bit of each sample for each of the six channels. This is then followed by the second layer, which contains the next most significant bit of each sample, and so on. This layered coding is very similar to the method of packetized voice, as specified by ITU-T G.764, designed for the convenience of congestion control. Further details concerning the effect of this layering on system performance are discussed in sections 4.5 and section 5.4 of the paper.

In addition to the audio stream, we utilized Cisco's IP/TV system for the encoding, transmission, and decoding of MPEG-1 video at approximately 1.5 Mbps and MPEG-2 video at approximately 3.0 Mbps[4]. As IP/TV does not yet provide support for external synchronization, nor the transport of high fidelity, multi-channel audio, the alignment of audio and video streams was performed manually, immediately prior to the demonstrations. It should be noted that MPEG-2 Advanced Audio Coding (AAC) supports synchronized playback of multi-channel audio with greater fidelity than AC-3 [4], but this format is not currently supported by IP/TV.

2.3 Network Route

For the first demonstration, the audio and video streams were transmitted over the high-performance networks managed by Canarie (Canada) and Internet2 (US) corporations. The network route used during this demonstration, along with the available bandwidth of each link, as measured by *pathchar*[5], is illustrated in Figure 3. For the second demonstration, the network route was confined to that portion of the original path between Montreal and Toronto.

---

[3] Dolby Digital encodes its data into frames, where each frame contains the audio for all coded channels over a time interval of 1536 samples. As the AES3 carrier is a two-channel interface, it contains 3072 sample locations per 1536-sample time interval. Dolby Digital encoders such as the DP569 generate an AES3 output such that the Dolby Digital frames each begin on a 3072-sample boundary, with a burst duration dependant on the data rate. In the case of a 448 kbps stream, the burst duration is 896 16-bit samples, plus a four-word preamble used to signal the type of digital content being conveyed. The remaining samples are digital zeros. Thus, the stream uses only 448 kbps of the total 1.536 Mbps capacity of the AES3 channel.

[4] Due to limited processor resources at the time, we only utilized the MPEG-1 stream for these demonstrations.

[5] Note that there has been considerable discussion as to the accuracy of measurements provided by this tool.

Although these networks have relatively high capacity, there were several bottlenecks that limited the available bandwidth. For an AC-3 audio stream, these bottlenecks may not appear significant, as our bandwidth demands amounted to only a fraction of the available capacity. However, transient network congestion induced by other high bandwidth applications would prove fatal to a simple audio transmission protocol. We were challenged to ensure robustness, despite the lack of bandwidth guarantees and the unavailability of any bandwidth reservation protocols on the network. Just as on a public Internet, we had to compete with all other traffic, while ensuring the delivery of our AC-3 audio stream.

3 The Audio Transmission System

The design of our transmission protocol was the most critical component of the system. Since the audio data was to be streamed and played back in real time, the protocol had to ensure delivery of data to the receiver with low latency, low error and loss rate, and most importantly, without breaks in continuity. It is this final point that ultimately determined the success of the demonstration, as any interruption of the output stream would cause an immediately noticeable break in the music.

Our design was based in part on the Adaptive File Distribution System (AFDP) [5], which provides efficient and reliable delivery of a file to multiple hosts on an Internet. The fundamental difference between AFDP and our application is that the former is not bound by real-time constraints. For a streaming audio application, we are willing to trade a small amount of reliability in exchange for real-time performance.

3.1 Transmission Protocol

TCP [6] and UDP [7] are the dominant network protocols used for computer communication. TCP, a connection-oriented, reliable, and full-duplex protocol, uses an acknowledgement and retransmission scheme to guarantee delivery of data. If a segment is not acknowledged by the receiver, the sender will continue to retransmit until the data is received successfully. Furthermore, TCP ensures an ordered reception of packets by delivering packet p_n only after all previous packets, p_i, $i < n$, have been received.

While these mechanisms are well suited to applications such as *ftp*, *telnet*, and *http*, they are unsuitable for the transmission of real-time media. This unsuitability is related to the characteristics of Internet communication, typified by widely fluctuating transmission delays. For example, although the average round trip time (RTT) for data sent between McGill and NYU was in the order of 50 ms, it was not unusual to observe RTTs of several hundred milliseconds during periods of high network traffic. Such delays can cause TCP to activate its congestion avoidance mechanism [8][9] or lead to timeouts at the sender, substantially degrading performance [9][10].

From the perspective of someone listening to an audio broadcast, if one packet fails to arrive before its audio content is scheduled to be played, it is imperative that successive packets are not delayed unnecessarily, otherwise, a discernable break in the music will occur.

For these reasons, real-time applications tend to favor UDP, a connectionless, unreliable protocol with no flow control mechanism. However, UDP by itself is clearly insufficient, and must be supplemented by additional protocol layers to ensure efficient and robust delivery of data.

RTP (real-time transport protocol) [11] provides end-to-end network transport functions suitable for applications transmitting real-time data such as audio and video over multicast or unicast network services. Its specification states that "*RTP is intended to be malleable to provide the information required by a particular application and will often be integrated into the application processing rather than being implemented as a separate layer*." In fact, applications typically run RTP on top of UDP [12]. As time was limited, rather than implement the entire RTP specification in our prototype, we opted to develop a lightweight version, tailored for our reliable unicasting needs. Our implementation includes most of RTP's basic services, including payload type identification, sequence numbering, timestamping and delivery monitoring.

RTCP (real time control protocol) [10] is an accompanying protocol of RTP designed to exchange control information related to real time data transmission. Either UDP or TCP can be used as the underlying transmission protocol, depending on the requirements of the application. Since RTCP was designed with large-scale multimedia applications in mind, the protocol offers considerable control information that is not required for our purposes. Furthermore, our application cannot afford the loss of any control information (i.e. NACK packets) if it is to ensure uninterrupted audio delivery, so a TCP connection for control packets is necessary. In order to implement an effective control mechanism within short time constraints, we chose to develop our system based on the lightweight AFDP protocol [5]. For future work, it may be useful to incorporate RTCP support as a standard.

3.2 Program Overview

Figure 4 presents the structure of the sender and receiver programs and illustrates the interactions between them. The main program structures are the same for both the AC-3 and uncompressed versions of the program. At each host, three network sockets are created for communication between the two programs:

1. A TCP connection initiated by the receiver, used to request transmission of audio data from the sender.
2. A UDP channel, used for the transmission and retransmission of the audio data.
3. A TCP connection initiated by the receiver, used for the exchange of control packets between the two programs and for the occasional retransmission of missing audio data, as will be explained below.

The sender process maintains an audio queue containing pointers to the data packets being transmitted. This process has two threads. One thread continuously reads the output of the audio encoder, stores this data in packets, and adds them to the audio queue. The second thread is responsible for the transmission and retransmission of audio data to the receiver.

The receiver process consists of a single thread, which continuously checks the UDP and TCP sockets for incoming data and stores these received packets in an audio queue. At every iteration through the loop, a periodic timer is checked. Whenever the timer expires, audio data is

dequeued and sent to the playback device and the remainder of the queue is checked for missing data. If necessary, a retransmission request is then issued to the sender.

All the data transmission by our system takes place in the form of user-defined data and control packets. Data packets, normally sent by UDP, except during retransmissions, consist of an identifying header and a payload, which carries audio data. The identifying header includes sequence number, timestamp and the payload data type. It also includes the most recent average transmission rate at the sender. This can be used by the receiver to estimate the current network congestion. Control packets, sent by TCP to provide guaranteed delivery, are used primarily for retransmission requests and exchanging monitoring information between the sender and receiver.

3.3 Data Buffering

As discussed in the following sections, retransmission is used for the recovery of lost packets as needed to ensure reliable end-to-end transport. This requires that audio data be buffered at both the sender and the receiver. Both ends maintain the audio data in a ring buffer, implemented as a linear array of pointers to the data packets, in order to allow direct access to the corresponding data segments.

As each block of audio is read at the source, the sender stores this data in a new packet and inserts a pointer to it into the appropriate cell, indexed by sequence number. When the head of the ring buffer reaches the tail, the oldest audio packets are freed, leaving space for newly acquired data. In this manner, the send buffer is always kept full, allowing for a maximal retransmission window, apart from a short period at the beginning of the program.

At the receiver, the incoming data packets are saved and a pointer to each is added to the buffer in a cell determined by the packet's sequence number. Once the number of data packets stored inside the buffer reaches the playing threshold, this data is dequeued and sent to the playback device. At this point, a periodic timer is started, and on each expiration of the timer, the oldest packets are dequeued from the ring buffer and played. For a lossless network, the receiving rate should be equal to the playback rate and hence, the number of packets in the receive buffer would be constant. However, in the real world, the receive buffer fluctuates because of packet loss caused by network congestion.

4 Packet Recovery and Congestion Control Mechanisms

Since packet loss is inevitable, we require a lightweight recovery protocol in our program. This protocol must be reliable, subject to best-effort limitations, and fast, such that the receiver recovers the missing data in time for playback. Our ability to satisfy these demands is determined jointly by the network architecture, link capacity, competing traffic, and the algorithms employed. It is only this last variable over which we have any control.

4.1 Forward Error Correction

In TCP, reliable data transport is achieved through the use of *stop-and-wait* and *go-back-N* algorithms. However, because of their potentially high latencies, these mechanisms are inappropriate for the transmission of real-time media

We thus considered two other methods for coping with transmission loss: forward error correction (FEC) and automatic repeat request (ARQ). In FEC, a block code or parity code is applied to a set of k packets to generate a set of $n > k$ packets. The receiver then only needs to receive any k-packet subset of the resulting n-packet block in order to recover perfectly the k original packets. The advantage of FEC is that the loss recovery happens entirely in the receiver and the sender does not need to know which packets were lost. Since no retransmission is required, no time penalty is imposed by the loss recovery. For this reason, FEC is often considered to be a preferred error recovery method for real time interactive media communication [13][14][15].

The drawback of FEC is the increased bandwidth required by the redundant data. Furthermore, its ability to recover lost packets is strictly dependent on the loss characteristics of the underlying network. Specifically, FEC cannot recover from a large burst of packet loss, which we observed frequently during our trials (see Section 5.2). For this reason, FEC was inappropriate for our implementation. An additional relevant factor is that we were only concerned with reliable, one-way transmission, hence, there was no need for low-latency interactivity.

However, it is in our interest to extend the current system to support interactive applications such as distributed musical performance and remote music education, in which there is a strict latency constraint on the delivery of audio in both directions. While figures vary considerably depending on experimental factors, studies of human tolerance of delay [16][17][18][19] suggest that effective musical interaction requires latency bounds of approximately 40ms.[6] Under such time constraints, FEC is worthy of further consideration as an option for error recovery. Obviously, different requirements motivate alternate decisions.

4.2 Automatic Repeat Request

In ARQ, the sender explicitly retransmits the packets that are requested by the receiver. Although this introduces a delay of at least three one-way trip times, previous research has demonstrated the effectiveness of ARQ for the transmission of real-time media, especially in the case of multicasting [20]. For our application, ARQ has proven quite effective provided that network capacity was not saturated.

Although we have so far demonstrated unicast alone, our system can be extended easily to support multicast, permitting its application to the simultaneous distribution of a performance to several locations. In this case, the Mbone [12] tree structure provides a suitable hierarchical model for the transmission of resend requests, in which the requests generated at low levels of the tree are collected and grouped by their parent nodes before being relayed higher in the tree. This is essentially the mechanism used by RTCP to prevent overwhelming the sender by negative acknowledgement (NACK) implosion.

---

[6] According to informal studies reported by Michael Brook, Bob Adams and Richard Boulanger, solo performers find feedback delays from daisy-chained MIDI instruments as low as 2-6 ms to be noticeable and in fact annoying. However, for multi-party interaction, Rasch's studies [16][17] found that among members of a woodwind instrument ensemble, deviations of onset from 30 to 50ms were perceived as being tightly synchronized. Regardless, delays arising from typical videoconferencing systems are a minimum of one order of magnitude greater.

Our implementation of ARQ employs a selective-repeat technique. Each data packet is labeled by the sender with a unique sequence number. The receiver detects missing data by checking periodically for a gap in the sequence numbers of received packets. When such a gap is found, the receiver issues a NACK to the sender as a retransmission request [5]. In general, such a gap-detection routine must allow for a small settling period of incoming data so that NACKs are not generated immediately as a consequence of out-of-sequence packets. However, with the very conservative buffering employed in our demonstrations, this was simply not an issue of concern.

We illustrate the ARQ-based packet recovery protocol by intentionally dropping 30 seconds of audio data in the middle of an AC-3 transmission, as shown in Figure 5. At the onset of simulated packet loss, the receiving rate drops to zero. The receiver then issues a retransmission request and shortly thereafter, the retransmitted packets begin to arrive.

Since the packet loss in this case is not caused by network congestion, the simulation is, of course, completely artificial, and serves only to demonstrate the high-level operation of the program. In practice, retransmission of lost data cannot be guaranteed to arrive in a timely manner during periods of network congestion, a point we address in the following section.

4.3 Retransmission Protocol

Having decided on the use of ARQ, we were then faced with the choice of retransmission protocol for lost audio packets. Based on the discussion of section 3.1, UDP would seem to be an obvious candidate. However, because of its inherent unreliability, we must also apply our loss recovery protocol to the retransmitted data. Furthermore, the temporal constraint that each audio packet must be received before its playback time mandates that losses of the resent packets be detected quickly and that further repeated retransmissions be carried out with minimal latency.

For the high bandwidth, uncompressed audio data, a UDP based retransmission protocol is used. Once a NACK is issued for missing packets, a timer is started. When this timer expires, the missing packets are rechecked, and if any are still missing, another NACK is sent. This process could continue until it is too late to receive the missing packets. However, a disadvantage of this method is that multiple retransmissions will increase network load, thereby leading to further packet loss. In order to prevent overloading the network with repeatedly resent data, we abort our retransmission efforts on any given packet after three attempts.

Under conditions of high congestion, our experiments demonstrate that TCP is unable to retransmit lost data effectively, due to its network-friendly congestion control algorithms. For greedy, high bandwidth, real time applications, UDP is thus the only choice for retransmission. However, for low bandwidth audio data such as AC-3, we find that TCP is well suited to deal with retransmissions, as the risk of delay associated with its congestion avoidance mechanism is preferable to continued packet loss under UDP. The reason for this hypothesis is simple: Even if the packets resent by TCP do not arrive in time for playback, the continuing UDP transmission of later audio data will not be affected. Thus, a break in the music, if one is forced to occur, will be of minimal duration. Our hypothesis was confirmed by experiments, described in section 5.3, which compare the performance of TCP and UDP for loss recovery of an AC-3 transmission

under different levels of congestion. This led us to choose TCP as the retransmission protocol for the AC-3 demonstrations.

4.4 Naïve Congestion Control

Retransmitted data naturally increases bandwidth utilization. For the transmission of uncompressed audio at 13.8 Mbps, some of the network links near the sender approached saturation. Under these conditions, it was thus dangerously easy to overload the network. What happens in this situation can be described as a positive feedback loop: A competing data stream causes congestion, which leads to packet loss in our application. Retransmission is then requested to recover the lost packets. If the competing data stream persists during the retransmission, congestion becomes even more severe, leading to a further increase of packet loss, and so forth, eventually leading to congestion collapse.

In order to avoid this scenario, we define certain thresholds on the network traffic parameters. Once these thresholds are exceeded, the program simply ignores any packet loss. Unfortunately, this method is rather naive. It may activate too early and drop packets that are potentially recoverable or activate too late to prevent congestion collapse. One would prefer a smoother response to congestion.

As discussed in section 3, TCP implements an effective congestion avoidance algorithm. It will back off and decrease the transmission rate whenever congestion is encountered. While the nature of our application does not lend itself to a strong congestion control procedure, it would be disastrous to do without one altogether. Therefore, we chose to implement a weaker algorithm, based on layered coding congestion control.

4.5 Layered Coding Congestion Control

Layered coding congestion control is a common method used in real time multimedia streaming [21][22][23]. The idea is to reduce sampling resolution, and hence, bandwidth usage of the media stream, while keeping the frame rate constant, when congestion is encountered. It is significantly different from the congestion control used in TCP, in which case the transmission rate is reduced during periods of congestion. Layered coding relieves network congestion by reducing the quality of the transmitted media, while preserving the frame rate at which it is sent. While beneficial, we note that this is a relatively weak solution, which cannot deal with congestion as effectively as the control algorithm employed by TCP.

Applying the layered coding technique to the uncompressed audio stream, we can drop the *n* least significant bits of each audio sample to reduce its bit rate, where *n* varies monotonically with observed congestion. For example, we can reduce the sample resolution from 24 to 16 bits for a 33% decrease in bandwidth requirements. Since the least significant bits are arranged at the end of each audio block, as shown in Figure 2, congestion control is easily accomplished by truncating data packets at the desired resolution.

The dynamic variation of sample resolution is the most important component of the congestion control algorithm, and depends on many parameters. In our experimental trials between McGill and NYU, we found that a packet loss of 10% represents a reasonable threshold at which to assume possible network congestion. Therefore, as soon as this threshold is reached within any

window, we reduce sample resolution by one bit. For every further 4% increase in packet loss, we drop an additional bit, which is equivalent to reducing the original 24-bits/sample stream by 1/24 ≈ 4%. In section 5.4, we present some measurements demonstrating the effectiveness of the layered coding congestion control in improving packet loss recovery and relieving network congestion.

5 System Performance

The characteristics of the underlying network are very important in determining the design of our system and its resulting performance. In order to better understand these characteristics of the network between McGill and NYU, we conducted numerous measurements of throughput, round trip time, jitter, and the effect of packet size and burst length on throughput.

5.1 Packet Size

A previous study [5] demonstrated that packet size has a significant effect on LAN throughput, with larger packet size leading to greater received data rates. We expected to find similar results in an Internet environment, so measured the throughput achieved for various packet sizes transmitted at a fixed rate between McGill and NYU. When the transmission rate was low (1.5 Mbps and 13 Mbps), packet size had little effect, but interestingly, at high transmission rates (greater than 20Mbps), we were able to achieve maximum throughput with a *medium* sized packet of approximately 6000 bytes. Based on these results, we opted to use a packet size of 1500 bytes for the AC-3 stream, and 6912 bytes for the uncompressed format, since the effect of packet size is not significant in these two cases.

5.2 Packet Loss

The pattern of packet loss is very important since it determines the loss recovery algorithm we should use. Despite contrasting experiences of researchers working with the high-speed Abilene backbone [24], our observations were consistent with previous results demonstrating that packet losses are typically bursty [25][26]. Figure 6 shows one of our packet loss measurements at 13.8 Mbps, with a packet size of 6912 bytes over a 20 minute trial. This means that the transmission rate of the packet is 250 packets per second. While the overall packet loss rate is negligible (0.6%), the number of dropped packets in some of these episodes is alarmingly high. Although, there are only 13 episodes in which the run of losses is longer than five packets, these account for 57% of the total packet loss. Under these conditions, a simple FEC is inappropriate for effective recovery.

Under conditions of packet loss caused by network congestion, it is likely that the congestion is short-lived, in which case, retransmitted data can arrive without difficulty. However, it is also possible that the retransmission will face similar or worse congestion than that which caused the original packet loss, especially if the resend occurs in close proximity to the initial transmission. By introducing a receive buffer that is longer than the expected burst duration, retransmission congestion problems can be reduced significantly.

The remainder of this section explores our design decisions concerning the management of packet loss as well as the experiments conducted to validate the design.

5.3 AC-3 retransmission

In earlier sections, we suggested that a TCP based retransmission might be better suited for AC-3 data than a simple UDP based scheme. In order to verify this hypothesis, we carried out simulations using both TCP and simple[7] UDP retransmissions to recover from packet loss caused by manually generated network congestion. Figure 7 summarizes the results of these tests, comparing TCP and UDP retransmission protocols under varying levels of network congestion. The generated congestion has a ten second duration while the receiver allows a maximum of five seconds delay before an audio packet must be played.

Under conditions of mild network congestion, TCP demonstrates a 100% recovery rate, while UDP, due to its unreliable nature, achieves slightly less. Thus, without a considerable amount of hand-tuning of the UDP recovery mechanism, TCP is preferable under modest network loads. This should come as no surprise, since TCP has been optimized for such environments.

However, as network congestion increases, TCP does not cope as well. In fairness, TCP was designed to be *well behaved* to the network under such conditions, whereas our needs are somewhat greedy. As can be seen in Figure 7, for our particular experimental configuration, UDP is better able to recover from network congestion brought on by competing traffic in excess of 30 Mbps. Although TCP will eventually deliver 100% of the packets, they do not arrive in time for playback. Thus, the effective recovery rate of TCP is significantly lower than our UDP protocol.

Interestingly, we note that despite the quantitative superiority of UDP (70% recovery vs. 17% for TCP) under severe congestion, there was little observable difference in terms of the audio quality. To a human listener, the resulting gaps in the audio stream are easily discernable, regardless of whether those gaps last for 80 ms or 800 ms. In fact, it may be preferable to mute for several seconds[8] rather than play a segment corrupted by missing packets.

5.4 Layered Coding Congestion Control for Uncompressed Audio

In order to evaluate the effectiveness of the layered coding congestion control mechanism for uncompressed audio, we tested its performance under conditions of varying network congestion.[9] Again, we produced congestion manually, by pumping a dummy stream of up to 17 Mbps into the network at the source, in parallel with our 13.8 Mbps audio stream, for a duration of

---

[7] For a simple UDP retransmission scheme, the lost data packets are retransmitted only once.

[8] The suggestion to invoke this primitive form of *error concealment* is based solely on our subjective experience from the experiment.

[9] Note that the experiments reported here were performed under different network conditions from those of the previous section. Whereas the data for Figure 7 was produced shortly after the first AES demo, with a lightly loaded receiver connected directly to the NYU backbone, the data for Figures 8 and 9 was obtained more recently, with the receiver running on a public machine several hops away from the backbone. This accounts for the discrepancy in additional data required to produce the same levels of congestion in the various experiments.

approximately 30 seconds. Figure 8 presents the results of lost packet recovery both with and without layered coding congestion control. While the program is unable to recover all lost packets in time for playback under conditions of severe congestion, it performs significantly better with the layered coding congestion control mechanism than without it. From the listener's perspective, there is a region at the left of the graph in which the layered coding congestion control is able to achieve an unbroken playback stream, with occasional reduction in sample resolution, whereas a simple retransmission strategy results in occasional breaks in the music.

Of course, the primary motivation for employing a congestion control mechanism is its damping effect on bandwidth utilization during periods of heavy network traffic. Although the positive feedback loop still exists, its gain is reduced. This effect is demonstrated in Figure 9, representing the rate of packet loss, which is a reasonable indication of network congestion, as a function of time. One can see that without congestion control, packet loss increases rapidly and remains above 50% for almost half of the congestion period. From our experience, timely recovery becomes exceedingly difficult once the loss rate exceeds this level. However, with congestion control, the overall level of packet loss is lower, as is the rate of its increase during congestion.

6 Conclusions

Before discussing the lessons learned from these experiments and demonstrations, it is helpful to distinguish between the requirements for our one-way audio transmission (e.g. broadcasting) and those of a two-way interactive audio transmission (e.g. teleconferencing). While the former are generally insensitive to delay, this is not the case for interactive applications, such as remote music teaching, where the maximum tolerable latency is very low. For example, studies by telephony companies suggest that a one-way delay of 100ms is perceptible to most listeners [27] and that a delay of 400ms in human conversation is "annoying." For musical interaction this number is considered to be significantly lower [16][17].

6.1 Effective Loss Recovery

Although different applications require completely different architectures, the fundamental problem of audio over Internet is flow control, that is, efficient transmission and reliable recovery of lost data. For high bandwidth, unidirectional music streaming, we found automatic repeat request (ARQ) to be the most effective loss recovery mechanism, allowing for perfect playback of the audio stream, provided network congestion remained moderate. Unlike forward error correction (FEC), which incurs the overhead of additional bits on every transmission, ARQ need only increase its sending rate to recover from lost data. Not only is this approach more appropriate under conditions of bursty packet loss, it is also helpful in minimizing bandwidth requirements, which, in the case of uncompressed audio, are already very high.

As a counterpoint, in a recent demonstration of HDTV over Internet [28], researchers at the University of Washington successfully made use of FEC for a 40 Mbps compressed transmission. However, their network path afforded two orders of magnitude more bandwidth, with their host-to-network connectivity operating on Gigabit (vs. 100 Mbit) Ethernet. Obviously, these differences do not permit an apples-to-apples comparison. While FEC proved

effective for the HDTV trial, it could easily lead to congestion problems when spare bandwidth is limited.

In general, one must make a trade-off between audio quality and latency. For interactive audio applications with a hard upper limit on latency, it is difficult to implement a perfect loss recovery protocol. If total delay is to be kept only marginally greater than network latency, it is nearly impossible to use an ARQ implementation since any retransmission request and resend involves a minimum of one round-trip delay. In this case, one would have no choice but to employ FEC. For example, in remote music teaching, if the participants are able to tolerate occasional breaks in the music, perhaps by repeating the broken session, then FEC should be employed to minimize latency and improve interactivity. In fact, we are in the process of implementing an FEC recovery protocol for AC-3 applications requiring low latency, since the lower bandwidth demands of AC-3 are unlikely to create congestion problems.

6.2 TCP as Transmission or Retransmission Protocol

Although, TCP is not an ideal protocol for bulk data transmission, various modifications can be made to improve its throughput [29]. For example, the size of the sliding window advertised by the receiver limits the amount of data that the sender can transmit pending an explicit acknowledgement [30]. To compensate, we increase the TCP buffer size on both sender and receiver, thereby improving throughput [29][31]. The selective acknowledgement protocol, along with various other extensions [10][32][33], is designed to improve the performance of TCP in the event of multiple missing segments within a window. Although, these modifications cannot change the fact that TCP, over a lossy network, is limited by its congestion control algorithm, we believe that a properly tuned TCP based packet recovery protocol could recover most of the packet loss experienced by an AC-3 stream. However, this would likely not be feasible for high bandwidth uncompressed audio, nor interactive music applications in which minimal latency is imperative.

Our experiments further indicated that vanilla TCP is a good candidate as the retransmission protocol for AC-3 data, provided packet loss is not severe. In this case, TCP is only used for relatively small amounts of data communication, so delays related to its robust algorithms tend to be of minimal consequence. Exhaustive testing, in which the system ran uninterrupted for several days, demonstrated the robustness of this approach.

Based on the strengths of TCP, we are developing a loss recovery protocol on top of UDP. This would provide many of the desirable features of TCP, in particular, flow control and reliability, while offering a less restrictive congestion avoidance mechanism. For example, to maximize performance, the slow start algorithm [30] will not be implemented. However, some congestion control is still required, in particular under severe packet loss, as best-effort (i.e. greedy) delivery over the Internet is highly problematic [34].

6.3 Multicasting

Although our experiments concentrated on one-to-one communication, there are likely far more applications for this work that involve simultaneous transmission to multiple recipients. Parallel unicasting to multiple hosts, as is the practice of many network-based applications, simply does

not scale, and hence, multicasting is required [5]. Fortunately, the algorithms we have employed lend themselves easily to a multicast adaptation, and we will be exploring this direction in the near future.

6.4 Coping with Congestion

An important determinant in the evaluation of a software system is its performance under extreme conditions, that is, stress testing. In our application, this is related to the system's performance under conditions of severe congestion. While no system can assure 100% data transfer in this case, we can relieve congestion by employing the appropriate congestion control algorithms. We have already demonstrated that a system with proper congestion control suffers less packet loss during a severe congestion than a system without congestion control.

Theoretical and experimental study has shown that a single traffic stream with no congestion control can cause congestion collapse in a busy network, disastrous to all users [28]. Our results are consistent with these studies. In response, the network community is promoting new routing algorithms that punish those traffic streams that do not employ congestion control mechanisms.

The design and implementation of a congestion control algorithm for end-to-end real time media communication is an active research area. While the congestion control utilized by TCP is mature and proven, it is not suitable for real time applications. Instead, most ongoing studies follow the approach of layered coding. Unfortunately, a general purpose solution is impossible, as different media such as raw video, linear PCM audio, and AC-3 audio, each have different characteristics that may or may not lend themselves to any one layered implementation. For example, our simple implementation of layered coding appears to be effective for uncompressed audio transmission, but cannot be applied to AC-3.

As a final note on this topic, if the bandwidth of an audio stream is high relative to the available capacity, slight reductions achieved through layered coding will likely prove less effective than TCP congestion control. From our experience, unless layered coding can reduce bandwidth substantially without a serious decrease in observed quality, a better solution for congestion control may be to cut the transmission entirely for a certain period.

6.5 Quality of Service

In the study, we devoted considerable effort to the development of various solutions to cope with packet loss due to network congestion. With quality-of-service (QoS) reservations protocols such as RSVP [35] becoming well known, the Internet of the future will hopefully render much of these efforts obsolete. However, until the underlying network, from any source to every destination, offers sufficient bandwidth to support requested services and is compliant with these new protocols, the solutions we offer here will have continued relevance to the networks community.

6.6 Final Words

Although considerable effort went into the design and development of the system, the authors must confess to being surprised by how well it worked. Given the unreliability of network

communications and the unavailability of bandwidth reservations, at least a few glitches were expected. Indeed, during development and tuning, various problems with the network configuration were noted, each of which could have proved fatal to the demonstration.

Following a cautious initial demonstration employing a fifteen second buffer to the Cantor Film Center, the delay was reduced to a mere three seconds. When this failed to introduce interruptions to the audio stream, the network news feed into NYU was resumed, thereby generating considerable competing traffic at the receiver end. At this point, several interruptions to the Cisco IP/TV video stream (requiring similar bandwidth to our audio system and utilizing the same delay) were observed. However, at no point during the demonstration did the audio break from a continuous stream.

Our experiments highlighted some of the strengths and weaknesses of TCP and UDP as network protocols for real-time transmissions. Clearly, each has its place, but neither, alone, is sufficient for applications of this nature. Ongoing efforts to combine the advantages of each, in conjunction with a choice of layered coding congestion avoidance mechanisms, may hold promise for a wide variety of future applications. It is our hope that the work started here will motivate the development of improved protocols that are better suited for high-fidelity, high-bandwidth, and low-latency communications over the Internet.

Acknowledgements

The authors wish to extend their gratitude to all of the participants who helped specify, test, and tune the system, in particular, Quan Nguyen and Peter Marshall, who provided constant assistance at the network and administration level, as well as Chris Smythe, member of the Multichannel Audio Research Laboratory at McGill, who provided invaluable technical assistance throughout the project. A very special thanks are due to Professor Gordon Foote, Director of the McGill Jazz Orchestra for his inspired musical presentations.

Our appreciation is due to the many individuals who took part in the actual performance, to the network providers, and to Steve Vernon and Nancy Byers-Teague of Dolby Laboratories, Mike Story and Duncan Mcleod of Data Conversion Systems, and Norman Tessier and Kim Dimick of Cisco Systems, each of whom lent considerable equipment and offered countless hours of support to the demonstration. We also wish to thank the AES Technical Council for suggesting this exciting project and shepherding it through its various phases of development and the AES 107th Convention Committee as well as the Canarie 5th Annual Advanced Networks Workshop Committee, for providing international fora for these exciting demonstrations.

References

1. AES White Paper. Technology Report TC-NAS 98/1: Networking audio and music using Internet2 and next-generation Internet capabilities. Journal of the Audio Engineering Society, vol. 47, April 1999.
2. Advanced Television Systems Committee. ATSC A/52. Digital Audio Compression (AC-3) Standard, Dec. 20, 1995. http://www.atsc.org/Standards/A52.
3. T. Beard, US patent #5451942, Method and apparatus for multiplexed encoding of digital audio information onto a digital audio storage medium, Issued Sept. 19, 1995.
4. T. Grusec, L. Thibault, and G. Soulodre, Subjective Evaluation of High-Quality Audio Coding Systems: Methods and Results in the Two-Channel Case, 99th Convention of the Audio Engineering Society, Journal of Audio Engineering Society. (Abstracts), Vol.43, preprint 4065, Oct. 1995.
5. J. R. Cooperstock and S. Kotospoulos. Why use a fishing line when you have a net? An adaptive multicast data distribution protocol. Proc. of USENIX '96. San Diego, 1996.
6. J. Postel. Transmission control protocol. RFC 793, USC/Information Sciences Institute, Sep. 1981.
7. J. Postel. User datagram protocol. RFC 768, USC/Information Sciences Institute, Aug. 1980.
8. V. Jacobson. Congestion avoidance and control. Proceedings of SIGCOMM'88, Palo Alto, August 1988.
9. M. Allman, V. Paxson, W. Stevens. TCP congestion control. RFC 2581, USC/Information Sciences Institute, April, 1999.
10. K. Fall, and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. Computer Communication Review, Vol.26, No. 3, July 1996.
11. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, USC/Information Sciences Institute, Aug.,1996.
12. S.McCanne, V. Jacobson. vic: A flexible framework for packet video, ACM Multimedia, San Francisco, CA, Nov. 1995.
13. N.Shacham, P. McKenney. Packet recovery in high-speed networks using coding and buffer management. Proc. IEEE INFOCOM, Vol.1, San Francisco, CA, June,1990.
14. J. Bolot, A. Veg-Garcia. Control mechanisms for packet audio in the Internet, Proc. IEEE. INFOCOM, San Francisco, CA, April, 1996.
15. J. Bolot, S. Fosse-Parisis, D. Towsley. Adaptive FEC-based error control for Internet Telephony. Proc. IEEE INFOCOM, New York, March, 1999.

16. R. Rasch, The perception of simultaneous notes as in polyphonic music. Acustica, 40, 21-33, 1978
17. R.Rasch, Synchronization in performed ensemble music. Acustica, 43, 121-131,1979
18. D. Levitin, K. MacLean, M. Mathews, L. Chu, and E. Jensen. The perception of cross-modal simultaneity. International Journal of Computing Anticipatory Systems. (in press) 2000.
19. J. Dixon, A comparison of audio and video synchronization rates (Quarterly Technical Report). New York: Society of Motion Picture and Television Engineers, 1987.
20. S. Pejhan, M. Schwartz, D. Anastassiou. Error control using retransmission schemes in multicast transport protocols for real-time media. IEEE/ACM Transactions on Networking, Vol.4, No.3, June, 1996.
21. N. Shacham. Multipoint communication by hierarchically encoded data. Proc. IEEE. INFOCOM, Vol.1, Florence, Italy, May, 1992.
22. L. Vicisano, J. Crowcroft, L. Rizzo. TCP-like congestion control for layered multicast data transfer, Proc. IEEE INFOCOM, San Francisco, CA, March, 1998.
23. G. Karlsson and M. Vetterli. Packet video and its integration into network architecture, IEEE. Journal on Selected Area in Communications. Vol.7, Num. 5, June, 1989.
24. D. Richardson. Personal Communications, Dec. 20, 1999.
25. M. Borella, D. Swider, S. Uludag, G. Brewster. Internet packet loss: measurement and implications for end-to-end QoS. Proceedings of the 1998 ICPP Workshops on Architectural and OS Support for Multimedia Applications/Flexible Communication Systems/Wireless Networks and Mobile Computing, 1998.
26. V. Paxson. End-to-end Internet packet dynamics. IEEE/ACM Transactions on Networking, Vol.7, No.3, June, 1999.
27. S. Cheshire. Latency and the Quest for Interactivity. November 1996. http://www.stuartcheshire.org/cheshire/papers/LatencyQuest.html
28. Internet HDTV. University of Washington Computing and Networking. http://www.washington.edu/hdtv/.
29. R. Cohen, S. Ramanathan. TCP for high performance in hybrid fiber coaxial broad-band access networks. IEEE/ACM Transactions on Networking, Vol.6, No.1, Feb, 1998.
30. R. Stevens. TCP/IP illustrated. Vol. 1, Addison-Wesley, Reading, Mass., 1994.
31. J.C. Mogul. IP network performance, in Internet System Handbook. D.C. Lynch and M.T.Rose, Ed. Addison-Wesley, Reading, Mass., 1993.
32. M. Mathis, J. Mahdavi, S. Floyd, A. Romanow. TCP selective acknowledgement options. RFC 2018, USC/Information Sciences Institute, Oct. 1996.
33. M. Mathis, J. Mahdavi. Forward acknowledgement: refining TCP congestion control. Proceedings of SIGCOMM 96, August 1996.

34. S. Floyd. Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Transactions on Networking, Vol. 7, No. 4, Aug. 1999.

35. R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin. Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification. Network Working Group RFC 2205, September 1997.

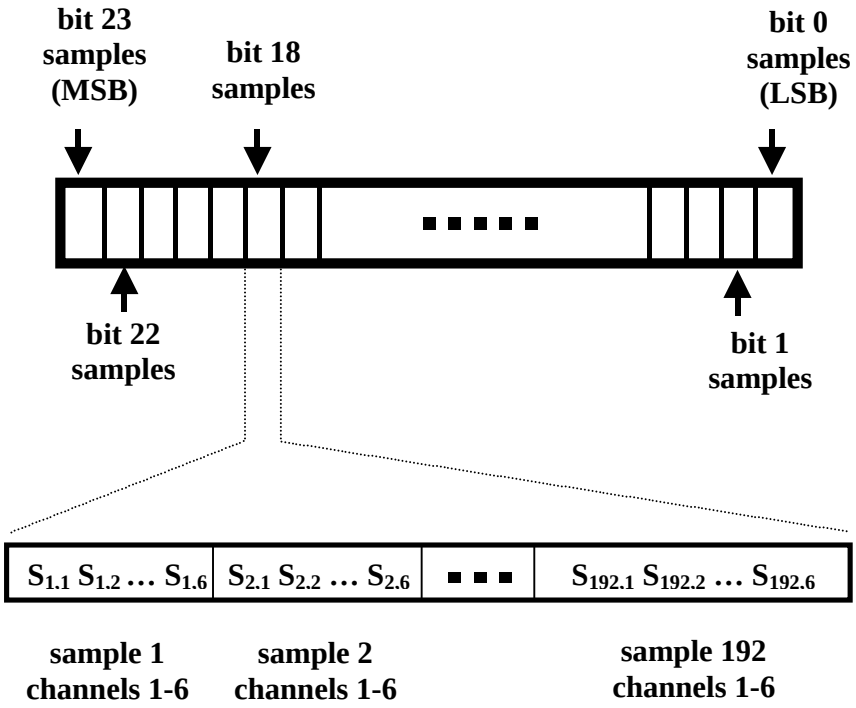

Figure 2. Format of an uncompressed audio data block. Every six bit segment contains the values of one particular bit position of one audio sample over all six channels. The data is ordered with the most significant bit of every sample appearing before the next significant bit. This permits signal reduction without complex manipulations through a simple truncation of the block at any chosen resolution.

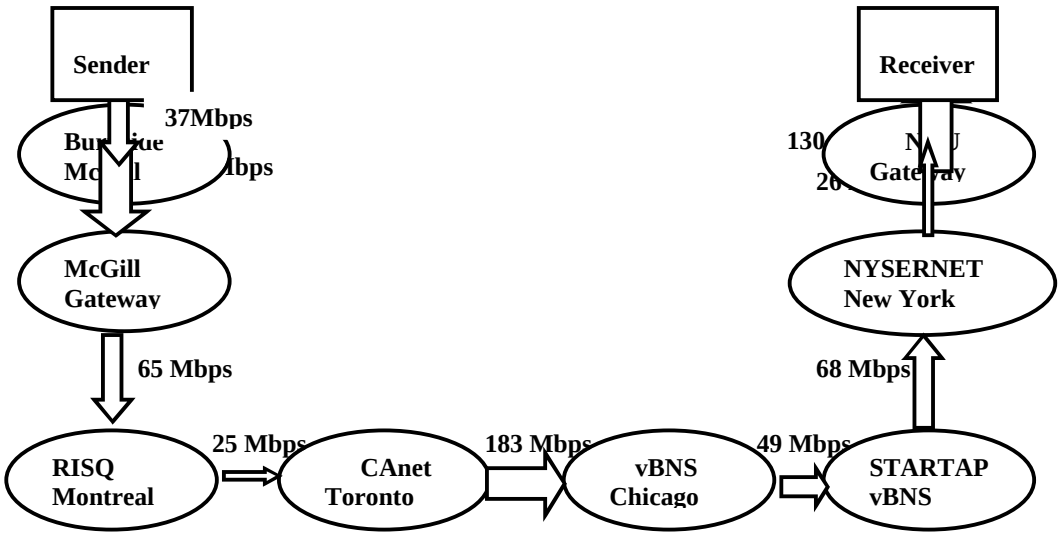

Figure 3. The network path between *sender* and *receiver* machines, illustrating the available capacity on each link, as measured by the *pathchar* utility (ftp.ee.lbl.gov/pathchar). Depending on the time of day and day of week, Internet traffic and the characteristics of the path can vary significantly.

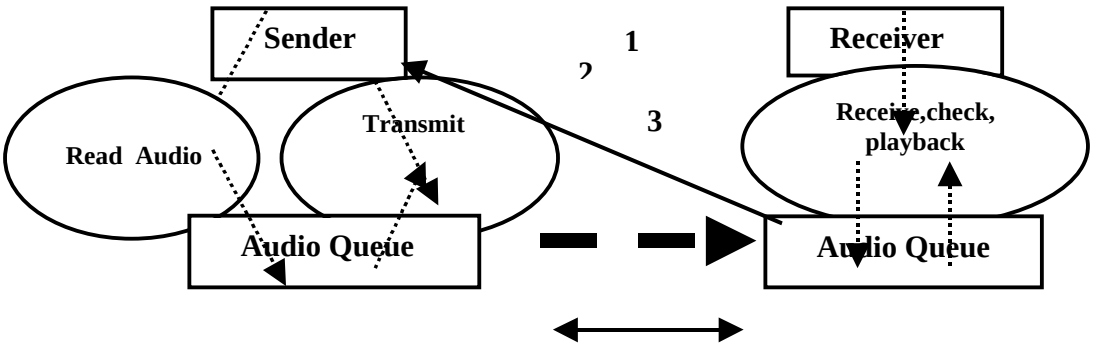

Figure 4. Communication Software Structure. 1 and 3 are TCP connections, 2 is UDP connection.

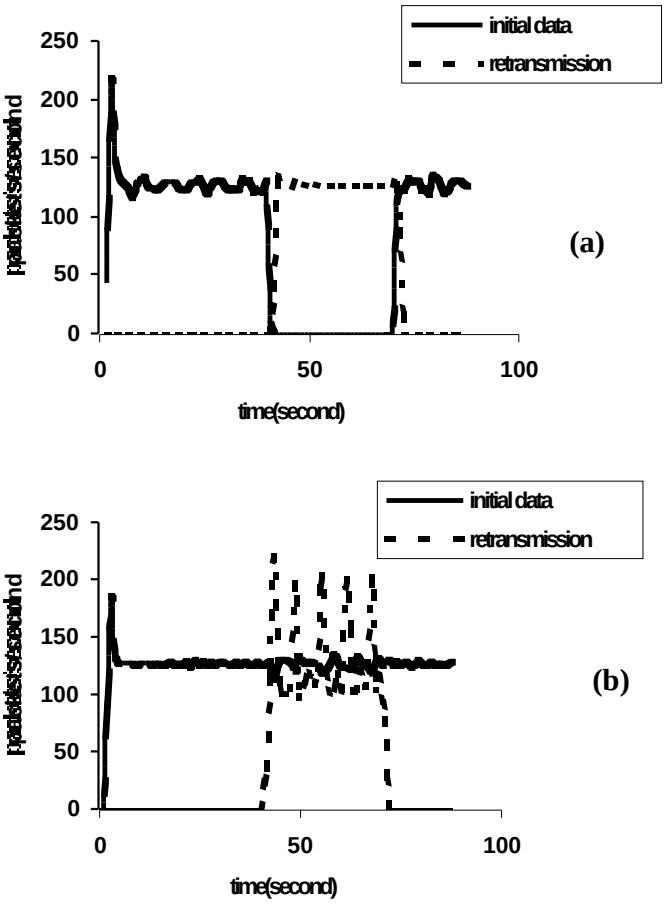

Figure 5. Demonstration of the loss recovery protocol, in which the receiver starts to drop packets at 40 seconds. (a) The sender begins transmitting packets normally, but in response to a resend request from the receiver, the retransmission protocol is activated at t=40s. (b) The receiver fails to receive any packets via the UDP socket so issues a retransmission request shortly after the onset of packet loss.

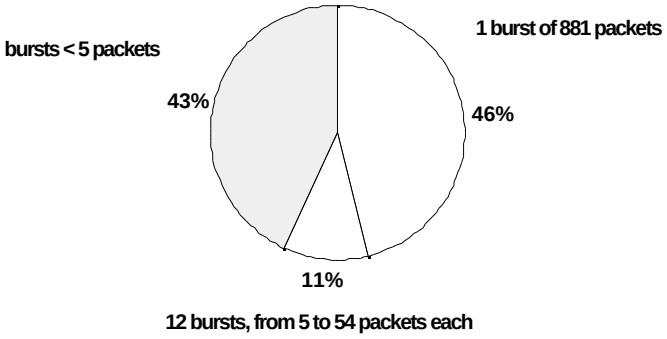

Figure 6. The bursty nature of packet loss. While there are only 13 bursts of packet loss that could not be recovered using an FEC scheme, these bursts accounted for 57% of the total packet loss. In particular, a single large burst, such as the one shown here of 881 packets, is not atypical for busy periods of the day.

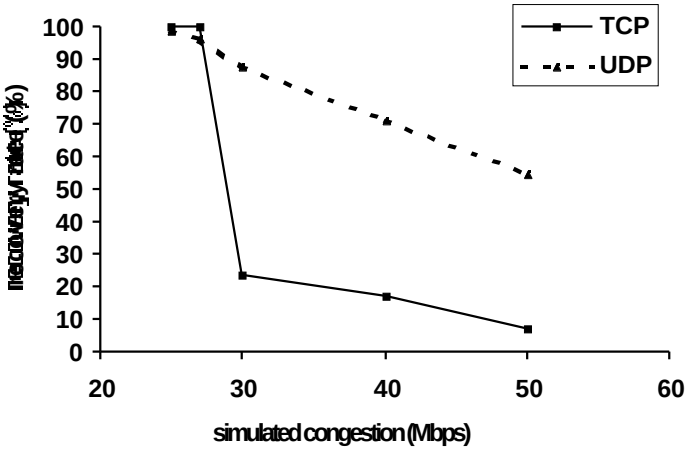

Figure 7. UDP- vs. TCP-based loss recovery protocols under network congestion. While competing throughput is under 30 Mbps, TCP is able to recover fully, but once congestion increases beyond this point, UDP is preferable for our application.

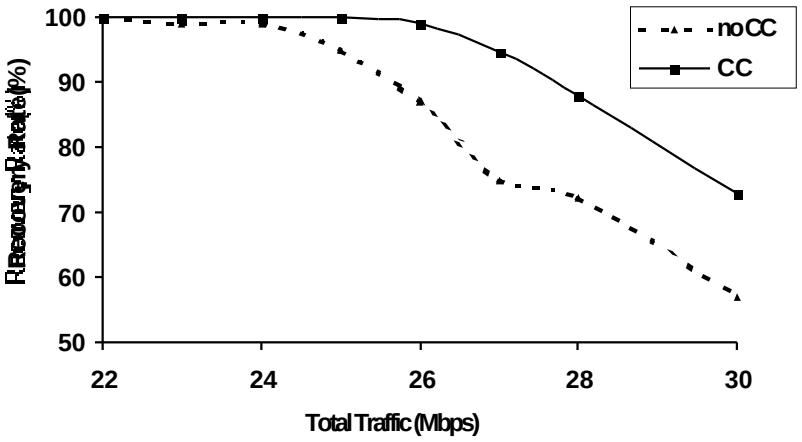

Figure 8.Demonstration of the effectiveness of lost packet recovery using layered coding congestion control (CC). The x-axis represents the sum of the bandwidth consumed by the simulated congestion stream and the uncompressed audio stream. The y-axis indicates the percentage of packets that are recovered by the protocol in time for playback.

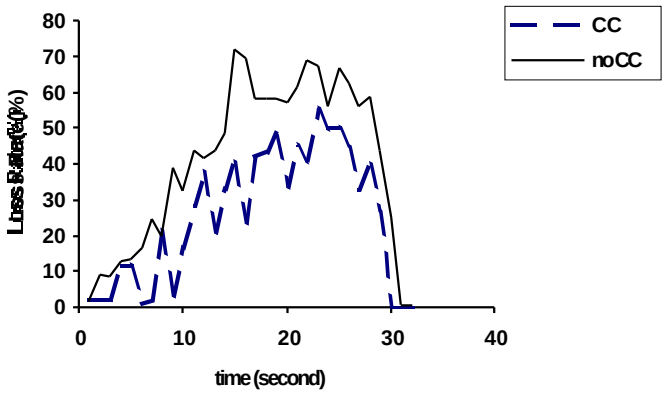

Figure 9. The effect of layered coding congestion control (CC) in relieving network load, under an artificially generated congestion stream of 30 seconds.