# Repair of Streaming Multimedia with Adaptive Forward Error Correction

Kenneth French*, Mark Claypool*
WPI Computer Science Department

**ABSTRACT**

Internet multimedia applications have timing constraints that are often not met by TCP, the de facto Internet transport protocol, hence, most multimedia applications use UDP. Since UDP does not guarantee data arrival, UDP flows often have high data loss rates. Network data loss can be ameliorated by the use of Forward Error Correction (FEC), where a server adds redundant data to the flow to help the client repair lost data. However, the effectiveness of FEC depends upon the network burst loss rates, and current FEC approaches are non-adaptive or adapt without effectively monitoring this rate. We propose a Forward Error Correction protocol that explicitly adapts the redundancy to the measured network burst loss rates. Through evaluation under a variety of network conditions, we find our adaptive FEC approach achieves minimal end-to-end delay and low loss rates after repair.

Keywords: Multimedia, FEC, Streaming, Audio, Video

## 1. Introduction

The increasing power and connectivity of today's computers have fueled the growth in Internet traffic, predominantly in the form of text-based Web traffic. Text-based applications have some characteristics and requirements in common. Most, such as telnet, ftp, and http, require guaranteed delivery, where every unit of data must be delivered without loss or error. As a result, these applications use the Transport Control Protocol (TCP), which provides guaranteed delivery by automatically retransmitting lost or corrupted data packets. Certain applications, such as TFTP or DNS, may not need a strictly reliable protocol, but rather a simple protocol with minimal delay and overhead. These applications commonly use the User Datagram Protocol (UDP). UDP does not provide any protection against loss; however, it does not have the overhead of retransmission allowing it to provide a fast, "best- effort" delivery.

Emerging new technologies in real-time operating systems and network protocols provide great opportunity for distributed multimedia applications. Multimedia applications have requirements different than text-based applications[1]. An audio stream, for example, requires that data is received in a timely fashion and is more forgiving of lost data[2]. If a data packet arrives at the player too late, it misses the time it needed to be played. This phenomenon, called jitter, causes gaps in the audio heard or unevenness in the video. In many cases, a late data packet in a multimedia application contributes nothing to the playback and is equivalent to a loss. Small losses in the playback stream can be replaced with substitute data or concealed so that the user does not notice.

Multimedia data transmission on the Internet often suffers from delay, jitter, and data loss. Data loss in particular can be extremely high on the Internet, often as high as 40%. Unlike traditional applications, multimedia applications can tolerate some data loss. While small gaps may not significantly impair media quality, too much data loss can result in unacceptable media quality. While TCP can be used to have any lost data retransmitted, the added delay and jitter of retransmissions and the window-based method of sending data, make TCP typically unsuitable for multimedia applications, especially when they are interactive[3]. UDP, conversely, provides a "best-effort" service that provides the multimedia application with greater control over timing. UDP does not, however, offer any guarantees on data loss. With UDP, potentially all data sent can be lost.

A multimedia stream can repair UDP data loss by the use of Forward Error Correction (FEC). The main idea behind FEC is for the server to add redundant data to a stream to help the client repair data loss. Media independent FEC seeks to repair data without knowledge of the data type, using a code or sequence to encode the data. One media independent FEC

* kfrench@wpi.edu; phone 1 508 831 5000; fax 1 508 831 5776; http://www.wpi.edu/~kfrench; claypool@cs.wpi.edu; phone 1 508 831 5409; fax 1 508 831 5776; http://www.cs.wpi.edu/~claypool; WPI Computer Science Department, 100 Institute Rd, Worcester, MA 01609

approach[4] for multicast uses a code created from Galois Field elements to construct a repair checksum that can be computed to recover a percentage of loss in a network stream. Media independent FEC systems have some shortcomings when they are used for an interactive audio session, the primary of which is the added end-to-end delay to repair information when a packet is lost on the network. The receiver needs to wait until it has received a number of packets to be able to reconstruct the missing information. This will add to the overall latency of the playback because the receiver is waiting for many packets to arrive before decoding and reconstructing the missing information.

Media-specific FEC[5] uses knowledge of the data type in adding encoding information. Low bandwidth redundancy is piggy-backed to the video stream at the sender, as shown in Figure 1. A lost packet is replaced by the redundancy transmitted within the next packet. When the redundancy fails to repair the lost packet, a repetition based error concealment technique is used to fill the gap. The stream at the top of Figure 1 is created with repair frames attached to primary data frames. The example shows that the second, third, fifth, and sixth packets are lost during the network transmission. The receiver is able to repair some of the lost data using the smaller, lower quality repair information when the stream is played out.

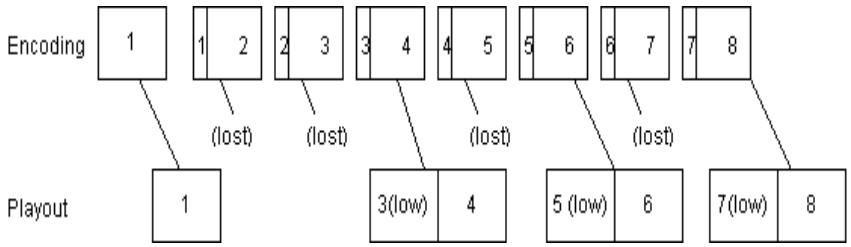


Figure 1. Repair mechanism used by new protocol.

Media-specific FEC is used with many audio applications because the audio format has various quality levels, and other research has shown that audio can be repaired using a lower quality sample of the lost information[6]. Media specific FEC is also a good choice for interactive applications where a large end-to-end delay is a concern because media independent FEC may add too much delay to the stream[7]. While FEC can be effective for repairing loss under some network conditions, under higher loss rates or bursty loss the repair information may be lost, too, making FEC ineffective.

One approach to adapting FEC to the specific network loss rates is the USF algorithm[8]. Based on a pre-set table, if network loss gets too high, the USF algorithm increases the amount of FEC and the spacing of the FEC information before sending the data across the network. The USF algorithm does not take into account the amount of delay added by the FEC nor does it record the level of network bursts, leaving it vulnerable to bursty loss or providing high end-to-end delays.

Our approach is to build an network protocol that explicitly adapts media-specific FEC to the burstiness of the current network.In order to provide a FEC mechanism that adapts to the current network conditions, we apply the following methodology: 1) design and implement a non-adaptive, media-specific FEC protocol (Section 2.0); 2) evaluate the effectiveness of the non-adaptive protocol under various network conditions (Sections 2.1 & 2.2); 3) build an adaptive FEC protocol based on results from the non-adaptive protocol (Section 3); 4) evaluate our adaptive FEC protocol (Section 3.).

## 2. FEC Protocol

We develop and evaluate our new protocol using the NS network simulator developed at the University of California at Berkley[9]. NS is an event driven simulator, where some examples of events are timers, packet arrival, packet transmission, and packet loss. The simulation engine that controls the system is written in C++, and uses a version of TCL that is object oriented called OTCL. These two languages allow for modules to be added and tested in the system with a minimal amount of overhead and change to existing simulator programs and routines. The NS simulator already has many modules or "plug-ins" that handle UDP, TCP, MPLS, steady state senders, variable rate senders, and some queue methods such as RED and FIFO, or DropTail. The NS system has an open source code policy that facilitates adding to add other network items into the system and to configure existing modules and protocols. NS also allows for packets to be examined at the IP level to check for lost, corrupted and delayed packets.

Our protocol is designed to incorporate some basic features of the media dependant FEC. We have initially designed our protocol for audio, but the techniques here should work for video as well. The protocol code uses a packet of audio data surrounded by a header and the repair information, with the total packet size being 504 bytes. The header takes up 24 bytes, the audio payload takes 320 bytes, and the repair information takes up 160 bytes. The program sends the audio data at a constant rate of 25 packets per second. Each packet contains 320 bytes of audio data, so at the rate given above it yields

a rate of 8000 bytes per second, which is a typical interactive voice audio sending rate. When all of the other data is considered it yields a rate of 12,600 bytes per second. The only real overhead in this system is the extra 160 bytes of repair information in each packet, so the system has an added overhead of 4000 bytes per second. The header information would be the same with or without the repair information. Figure 2 shows an example of the packet that is sent from the server to the client. The layout of the header and data information is shown along with the number of bytes for each part of the packet.

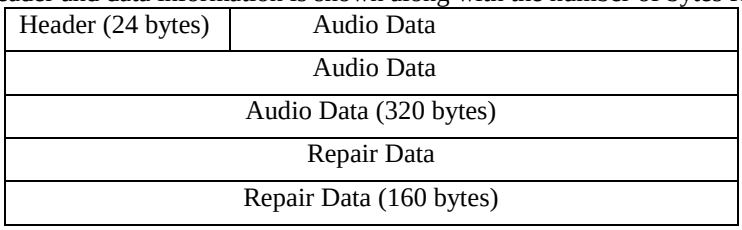



Figure 2 Sample Packet used with protocol

The program uses a variable called *repair_depth* to send and store its repair data. This spaces out the repair information from the packet being sent, with larger values intended to avoid loss during burst loss events on the network. The repair depth can be set to any number required (testing was done with a repair depth up to 10). If the repair depth is larger than the burst loss rate on the network, the protocol is able to repair a larger number of lost packets. If the number is set much smaller, the original audio data and its repair information are lost and increase the overall stream loss numbers.

The protocol includes a *FEC_class* that is used when a program wants to use FEC to encode and decode information either sent or received. The functions and defines are encapsulated into the class to facilitate conversion into and out of the FEC algorithm. The FEC_class is used as the program starting routine where it creates the media specific redundant information and inserts it into the packets at the correct repair_depth. It also handles receiving the data from the network and keeping track of loss and repair statistics for the current session.

Another class that was created for the NS simulator was the *FEC_Send_Timer* class that contains the methods called from the event driver in the main simulator. This allows the NS subsystem to interact with the new FEC objects. This new class is an extension of the *TimerHandler* class that is defined in NS, allowing the FEC objects to be able to "register" its own function to handle events that the NS simulator may generate. The extended FEC timer handler facilitates scheduling events such as sending packets at a constant intervals and scheduling handlers that receive packets off the network and process them for any needed repairs.

### 2.1 Experimental Setup

To evaluate our protocol, we devised a test environment where we could control the loss in the network by controlling the number of active flows. The test environment we devised is shown in Figure 3. Since most traffic on the Internet consists of TCP data flows, the main test environment used competing TCP flows in different amounts to increase the loss seen end to end. We ran our new FEC-based protocol (depicted by the nodes with the label "FEC") against from 10 up to 100 competing TCP flows running FTP. The flows compete for scarce router resources in a typical "dumbbell" topology, with one FIFO DropTail router at node N1. The link between the source nodes to the network nodes has a bandwidth of 10Mbps and a 5ms delay, and the two network nodes are connected together with a 6Mbps, 20ms delay link. This configuration causes congestion and loss at the N1 router in order to evaluate the effectiveness of the FEC-based protocol. The DropTail FIFO router has a packet queue equal to 10 packets to increase the chance of dropping many packets from the system when congestion occurred. Similar to other work[10], experimental runs are for 120 seconds each, with the first 20 seconds of each test discarded to account for any unstability caused by startup.

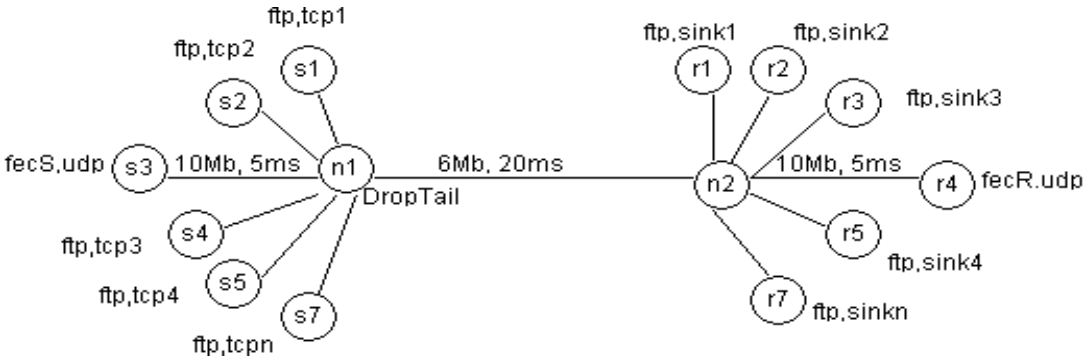



Figure 3 Network topology and Flows

The main performance statistic for the FEC protocol is the loss in the network (the *raw* loss) and the loss after repair (the *perceived* loss). The FEC receiver keeps track of the number of packets received, the current latency, the average latency, number of packets repaired and the total number of packets that could not be recovered. Based on preliminary testing, 10, 15, 20, 35, 50, and 100 competing TCP flows were run causing the network loss to range from 13% to 32%.

**2.2 Results**

We first examine the raw network loss (before repair) as the number of flows increases. Figure 4 shows the percentage of lost packets seen at the receiver with no FEC repair. The different network flow tests had an average loss ranging from 13 to 32 percent. Although these numbers are high for most network conditions, the FEC protocol is able to nearly completely repair data for network loss rates below 13 percent. If a network has a very high loss, the protocol may not be able to repair the packets, because the repair information is lost along with the original data.

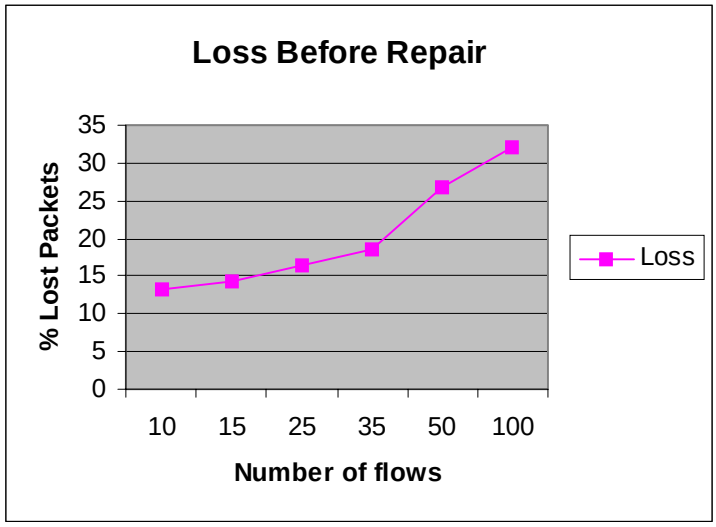



Figure 4 Percentage of Loss Before Repair.

We next present data with the FEC flow using a repair depth of 5 to show the general benefits of FEC. Table 1 presents the data. "# of Flows" is the number of competing TCP flows. "Total Recvd" is the total number of packets received by the FEC flow. "Repair Depth" is the repair depth run at (all were run at depth 5 in this table). "Total Recov" is the number of packets that were repaired by the FEC. "Percv Lost" is the number of packets that were then perceived lost by the application. "Ave Latency" is the average latency in the application, in seconds. "Loss %" is the unrepaired network loss percentage. And "Repair %" is the number of network packets that could not be repaired by the FEC.

| # of Flows | Total Recvd | Repair Depth | Total Loss | Total Recov | Percv Lost | Ave Latency | Loss % | Repair % |
|---|---|---|---|---|---|---|---|---|
| 10 | 2932 | 5 | 390 | 388 | 2 | 0.077 | 13.30 | 99.49 |
| 15 | 2949 | 5 | 417 | 414 | 3 | 0.078 | 14.14 | 99.28 |
| 25 | 2940 | 5 | 485 | 485 | 0 | 0.080 | 16.50 | 100.00 |
| 35 | 2936 | 5 | 546 | 535 | 11 | 0.083 | 18.60 | 97.99 |
| 50 | 2956 | 5 | 789 | 721 | 68 | 0.088 | 26.70 | 91.38 |
| 100 | 2966 | 5 | 954 | 832 | 122 | 0.090 | 32.16 | 87.21 |

Table 1 Sample data collected from test run

Figure 6 shows the percentage of packets that could not be repaired at the receiver at each repair depth. These are packets lost by the network that the FEC could not recover or fix. With higher repair depths, the experimental runs with few flows had most of the data recovered because of the low amount of bursty loss encountered; while the experimental runs with a high number flows saw a decrease in the number of lost packets, but did not recover all of the lost data. Notice that as the repair depth is increased above 5, the lines start to curve upward, and more loss is perceived by the application. This increase is caused by the repair depth being larger than the current burst but perhaps part of a following burst, causing both the primary and secondary data packets to be lost.

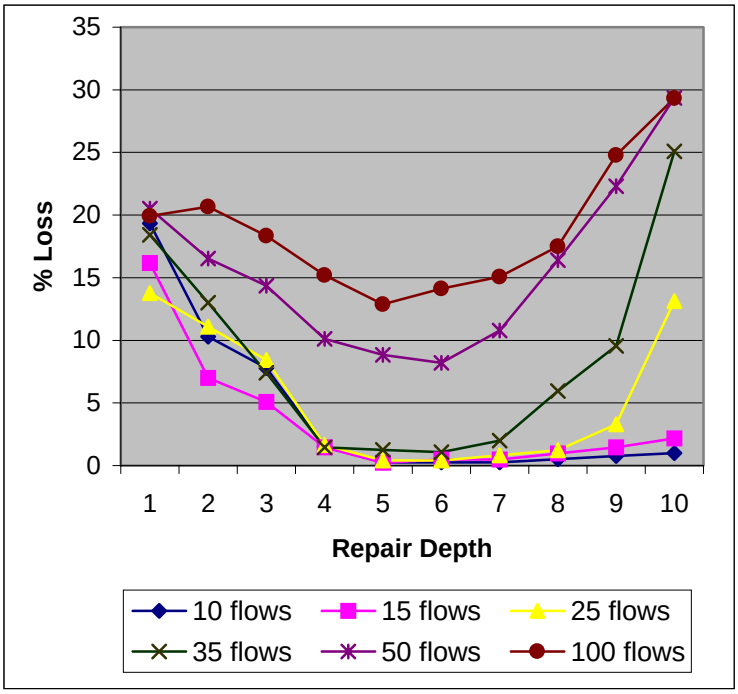



Figure 6 Percentage of Loss After Repair vs. Repair Depth

From the data in Figure 6, it is clear that there is an optimal repair depth where the loss after repair is at a minimum. When there are several minimum points (the curve is flat at the bottom), the smallest repair depth should be chosen as this will have the least end-to-end delay. For example, for 35 flows the best repair depth is 4 while for 50 flows the best repair depth is 6.

## 3.Adaptive FEC

Our approach is to develop a new protocol that uses media specific FEC that adapts to the current network burst rate. We first examine the loss patterns in the network to see they can help develop a protocol to automatically determine these optimal repair-depth values.

Figure 7 shows the loss pattern for the 10-flow experimental run. Most of the loss events cause losses of between 2 and 3 packets at a time; there are some spikes of greater loss with amounts as great as 5 packets in a row being lost in one network burst. A repair depth of 3 would repair most of the data, except a few burst spikes, and achieve a low average delay.

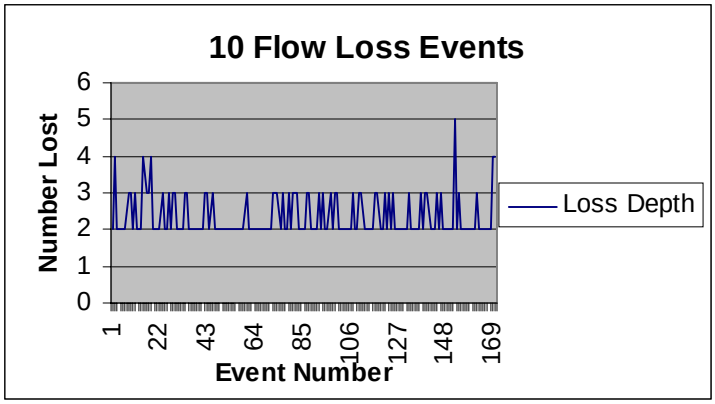



Figure 7. Loss events with the 10 flow test

Figure 8 shows the loss pattern for the 25-flow experimental run. Most of the loss events cause losses of between 2 and 4 packets at a time, while there are some burst spikes of greater loss with amounts as great as 5 packets in a row being lost in one network burst. A repair depth of 4 would repair most of the data, except a few burst spikes, and achieve a low average delay.

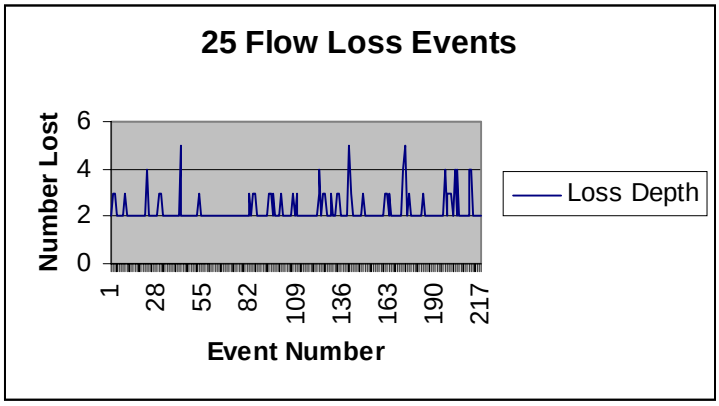



Figure 8. Loss events with the 25 flow test

Figure 9 shows the loss pattern for the 100-flow experimental run. Most of the loss events cause losses of between 2 and 6 packets at a time, while there are some spikes of greater loss with amounts as great as 7 packets in a row being lost in one network burst. A repair depth of 4 would repair most of the data, except a few burst spikes, and achieve a low average delay.

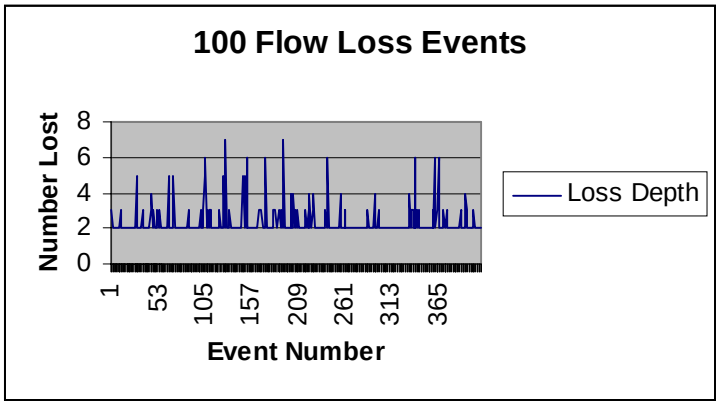



Figure 9. Loss events with the 100 flow test

Using these results, our adaptation is done by changing the distance in units of packets between the original data and the redundant repair data. Our protocol monitors the current burst loss rate and adapts the repair depth dynamically to most effectively repair lost data given this rate. Thus, when network burst rates are high, our protocol increases the repair depth to better repair lost data while when network burst rates are low, our protocol decreases the repair depth to reduce the end-to-end delay from the repair spacing.

Unlike the previous experiments that had a fixed number of flows for the entire run, we developed a scenario that could effectively test our adaptive FEC system as the flows increased from 10 to 100 over time. Figure 10 shows the network loss rates over time for this new simulation, where loss rates range from 13% to 33%.

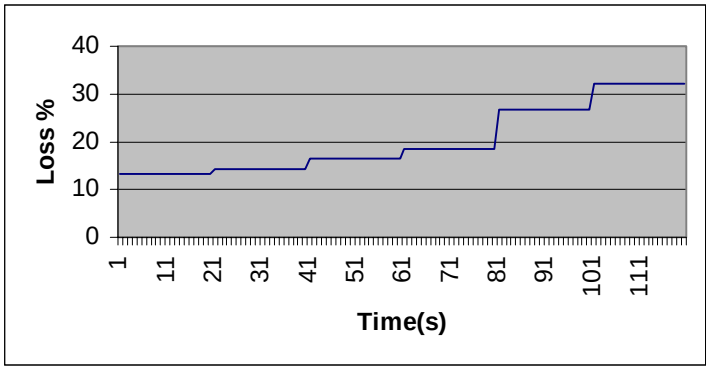
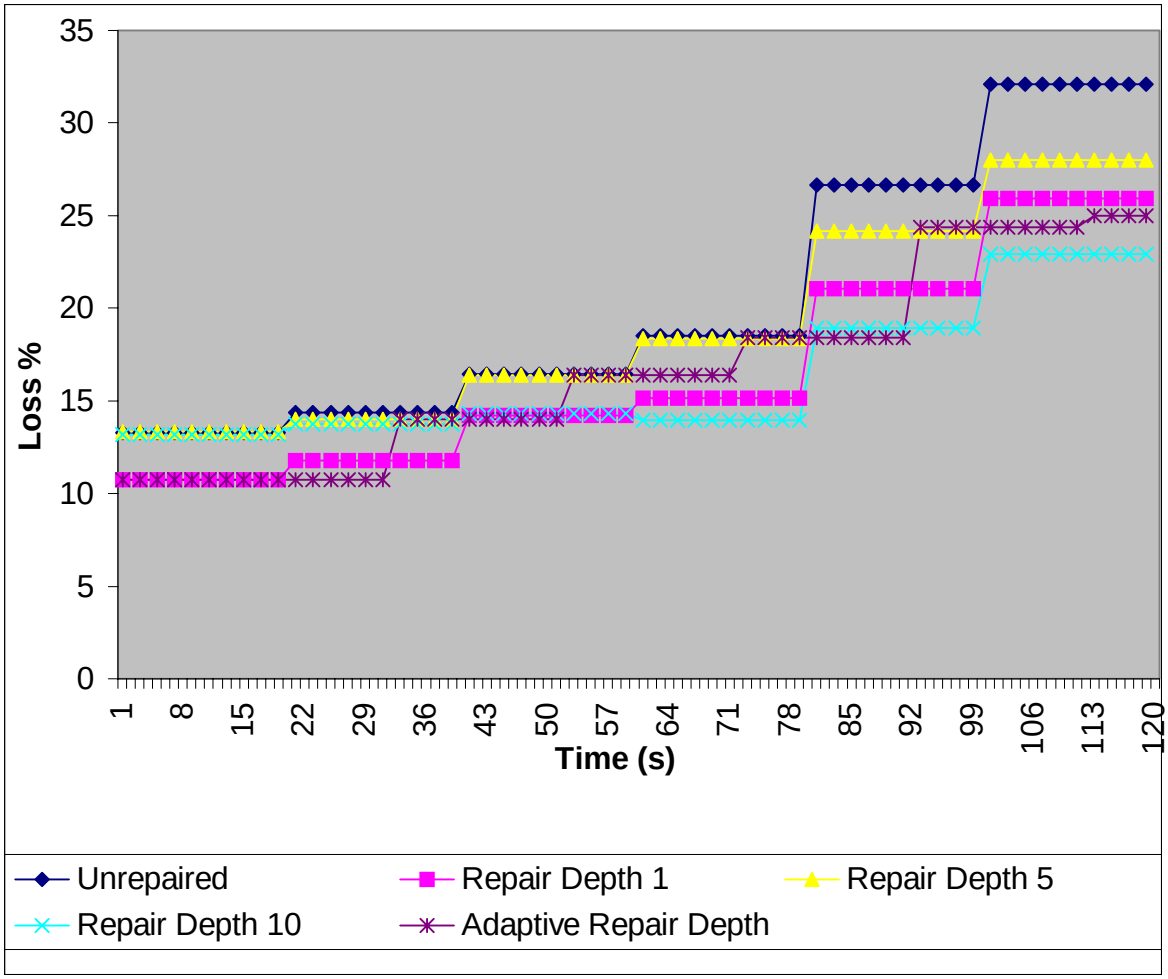



Figure 10. Network loss during sample testrun.

Figure 11 depicts a comparison of various FEC flows with static repair depths with our adaptive FEC protocol for the same network simulation. The repair depth of 1 has the best repair rates when the network loss rate is low for few flows, but later suffers from higher loss rates. In contrast, the repair depth of 10 has better repair rates for high network loss rates but suffers from a larger end-to-end delay for the whole session. Our adaptive protocol is able to achieve repair rates that are as low as the best repair depths in most cases and has a minimal end-to-end delay in nearly every case.




## 4. Conclusions

The growth in power of today's computers and networks present the opportunity for high-quality audio and video across the Internet to the desktop. Internet audio and video typically does not use TCP, the de facto Internet protocol, since TCP has too strong a guarantee on lost data and TCP does not respect the timing constraints multimedia applications require. Instead, many streaming multimedia applications use UDP, making them susceptible to high data loss rates.

Forward Error Correction (FEC) is a promising means by which multimedia UDP flows can recover from lost data without using retransmissions. FEC has been used in many Internet audio applications and has been proposed for many other applications. Unfortunately, today's FEC implementations do not adapt to the specific loss bursts seen by the receiver, resulting in possibly high loss rates when the FEC repair depth is set too low or high average delay when the FEC repair depth is set too high.

In this paper, we present a FEC protocol that dynamically adjusts to the current burst loss conditions. Evaluation under network conditions with a variety of loss rates demonstrates our protocol can adjust the FEC repair depth to achieve high repair rates for high loss network conditions and low average delays for low loss network conditions. The contributions of this work include: 1) a detailed examination of the effect of media specific FEC repair depth on loss rates; 2) design and evaluation of an adaptive FEC protocol for multimedia flows on the Internet; and 3) implementation of the adaptive protocol in a popular network simulator.

# 5. Future Work

There are many areas for possible future work. With media-specific FEC, if each frame is to be decoded and played as soon as it arrives, there will be added delay during the display when one packet is lost. After some waiting, the secondary frame will be extracted, decoded and played after the one frame halt, causing the playout to be uneven even if there is no loss. Past work has shown that this jitter can be as detrimental to perceived quality as is data loss[11]. Analysis on how much jitter FEC introduces to the display and how to solve this problem can be an interesting area of future research. The adaptive repair depth algorithm should also consider the delay that is tolerable by the application. Trading off the delay and loss as in our previous work[12] may prove a beneficial approach. It is also possible to combine FEC with other repair techniques, such as interleaving or selective retransmission.

## REFERENCES


1. X. Guo and C. Pattinson, "Quality of Service Requirements for Multimedia Communications", *In Proceedings of Time and the Web Workshop*, Staffordshire, UK, June 1997.

2. M. Claypool and J. Riedl, "End-to-End Quality in Multimedia Applications", Chapter 40 in *Handbook on Multimedia Computing*, CRC Press, Boca Raton, Florida, 1999.

3. S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications", *In Proceedings of ACM SIGCOMM*, Stockholm, Sweden, May 2000.

4. Rizzo, L Vicisano, L "RMDP: an FEC-based Reliable Multicast protocol for wireless environments" http://www.iet.unipi.it/~luigi/mccr6.ps

5. C. Perkins, O. Hodson and V. Hardman. "A Survey of Packet-Loss Recovery Techniques for Streaming Audio", *IEEE Network Magazine*, Sep/Oct, 1998.

6. Hardman, V., Sasse, M. A., Handley, M. and Watson, A. Reliable Audio for Use over the Internet, In *Proceedings of Internet Society's International Networking Conference (INET)*, Ohahu, Hawaii, USA, 1995.

7. R. Steinmetz, "Human Perception of Jitter and Media Synchronization", *IEEE Journal on Selected Areas in Communications*, Vol. 14, No. 1, January 1996.

8. C. Padhye, K. Christensen and W. Moreno. "A New Adaptive FEC Loss Control Algorithm for Voice Over IP Applications", *In Proceedings of IEEE International Performance*, Computing and Communication Conference, February 2000.

9. VNT, "Virtual InterNetwork Testbed, A Collaboration among USC/ISI, Xerox PARC, LBNL, and UCB", URL: http://netweb.usc.edu/vint/

10. R. Rejaie, M. Handely and D. Estrin. "RAP:An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet", In *Proceedings of IEEE Infocomm*, 1999.

11. M. Claypool and J. Tanner. The Effects of Jitter on the Perceptual Quality of Video, In *Proceedings of the ACM Multimedia Conference*, Volume 2, Orlando, Florida, USA, November 5, 1999.

12. M. Piecuch, K. French, G. Oprica and M. Claypool. A Selective Retransmission Protocol for Multimedia on the Internet, In *Proceedings of SPIE Multimedia Systems and Applications*, Boston, MA, USA, November 5-8, 2000.