AUDIO STREAMING USING INTERLEAVED FORWARD ERROR CORRECTION

Submitted by

N.ARVIND-8907211
V.VIJAY BASKAR-8907245

Guided by

INTERNAL GUIDE

Mr. B. SIVASELVAN
Lecturer,
Department of Computer Science and Engineering,
Sri Venkateswara College of Engineering

In partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING
OF
UNIVERSITY OF MADRAS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRI VENKATESWARA COLLEGE OF ENGINEERING
PENNALUR 602105
MARCH 2003

BONAFIDE CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRI VENKATESWARA COLLEGE OF ENGINEERING

PENNALUR, SRIPERUMBUDUR – 602 105.

INDIA.

This is to certify that this project is a bonafide record of work done by

N.ARVIND-8907211
V.VIJAY BASKAR-8907245

during the academic year 2002-2003,

and submitted to the **UNIVERSITY OF MADRAS** *in partial fulfillment of the requirements for the award of the degree of*

BACHELOR OF ENGINEERING

ABSTRACT

During the last few years we have witnessed an explosive growth in the development and deployment of applications that transmit and receive audio content over the networks. The problem of providing a jitter free audio content over the network is an issue under extensive research. The time since the generation of packet at the source and its reception at the destination can fluctuate from one packet to another. This phenomenon is referred to as jitter. The effective utilization of network bandwidth and the rate at which data is transmitted are the important factors that decide the audio quality and minimization of jitters. These are achieved by the loss recovery schemes that attempt to preserve acceptable audio quality even in the presence of packet loss. Listening to music over Internet is aided by a system called Streaming server. Streaming servers are highly influenced by the existing network parameters like server load, congestion and the protocol in use. Our project unfolds an approach using Interleaved Forward Error Correction technique for contiguous listening pleasure.

Keywords: jitter, Streaming server, congestion, Interleaved Forward Error Correction

ACKNOWLEDGEMENTS

Our project would not have been a success but for the sincere and unstinting support received by us from many a quarter. We would like to avail this opportunity to thank some of them.

First of all we thank our respected Principal **Prof. S. Muthukkaruppan** to have provided us with all the necessary facilities and healthy atmosphere during the course of the project. We are grateful to our Head of Department **Prof. R. Ramachandran**, for timely help and advice.

We are also indebted to **Prof.D.K.Muthiah** for providing us with valuable insights during the course of the project.

We express our gratitude to **Mr. B. Sivaselvan** for his guidance, technical support and encouragement throughout the course of the project work.

We also thank our project coordinator **Ms. P. Devi**, other faculty members, staff and students of the Department of Computer Science and Engineering, Sri Venkateswara College of Engineering for their cooperation and support.

TABLE OF CONTENTS

LIST OF FIGURES

1.INTRODUCTION

It would be a real pleasure to hear music over Internet without any breaks in between. But we are in a congested world where everything that travels spatially has to face traffic jam and collision. Listening to music over Internet is aided by a system called Streaming Audio. Network data packets are no exceptions to this rule. Nowadays, even a dial up connection supports better reception of streaming data. But we have always developed a compromise on the quality of music that we hear because higher the quality, higher the data rate requirement. With the data rate that is possible in a dial up or an ISDN connection, we cannot afford to use a higher data rate quality audio service.

1.1 MOTIVATION AND OBJECTIVE

Streaming audio is highly influenced by the existing network parameters like server load, congestion and protocols. Server load depicts the efficiency of a server, which is to handle many clients simultaneously. This can be easily fixed by providing a good server configuration. After all, the sole purpose is to provide high quality service for the clients. But congestion is the key factor that is to be analyzed. This project unfolds an approach using the UDP transport layer for contiguous listening pleasure irrespective of congestion situations. The choice of UDP instead of TCP is dealt in the following section.

1.2 CHOICE OF UDP

Most of the network applications use the Transfer Control Protocol (TCP/IP) or connection oriented protocol. TCP affirms error-free, in-sequence data transfer for the user. TCP also handles congestion in its own way. Packet loss, duplicate packets, packet retransmission are handled implicitly by TCP. TCP is also called a state-full protocol. Since it handles all by itself, the application that is running on TCP can never take control over the transmission policy. Streaming systems come under the category of Fault tolerant Real time systems where error is tolerable. This is the major problem with the TCP. It does too much work to ensure error-free transmission and hence it compromises the timely data delivery. In streaming audio, timely packet delivery is more important than error-free delivery. The nature of audio being redundant allows errors to be tolerant in the audio data. Since our hearing mechanism is of Averaging type, minor error bursts are ignored or rather equalized by the human hearing system. Exploiting this fact unleashes the sensitivity towards errors.

TCP's latency in data delivery exponentially increases when congestion rate is high. TCP does the congestion control with mechanisms like slow start and exponential back-off which eventually gears up the latency factor. This latency factor is very crucial when timely delivery is the driving criterion.

UDP, on the other hand is a stateless protocol. Although it's an unreliable protocol, it's the best choice for streaming applications. The reason is all the work that is done by TCP can be done based on the requirements of the application. Since the main requisite is packet delivery rather than error-free delivery, the application is free to choose its optimizations with respect to the existing network conditions. The bottleneck with the UDP is that loss of packets is not known, as there is no acknowledgement from the client. Moreover in the receiving end one has to sequence the packets according to their time stamp or sequence number. Since a packet loss up to 5% in audio content is generally tolerable, UDP is most favored.

1.3 NATURE OF AUDIO

Audio data is basically binary data with a lot of flexibility. The only constraint on audio data is timing. Here, timing means sampling and playing rate. Audio can be erroneous to some extent. Based on the fact that hearing system falls under the category of Averaging sensors, minor pitfalls in the quality could be easily compromised. It is based on this fact that our project is modeled. Another wonderful advantage is the redundancy of audio data. If a part of audio is missing, the gap in playing could be regenerated from already played data and the available future data.

1.4 GENERAL DATA FLOW

This part explains about the general data exchange between the server and the client. The block diagram depicts the general operations involved in streaming audio.

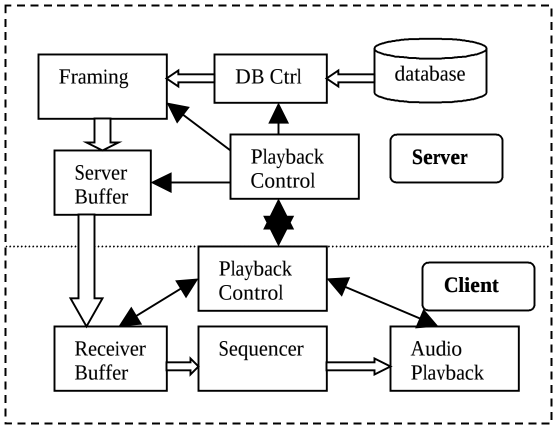

FIGURE 1.1 General Data Flow

From the client, a request for audio is posted to the server. The server searches for the required audio in its database. If the audio file is found, it splits the audio file into numerous data frames that are sequence numbered and put in the output buffer for delivery. On the other hand, data is received in the client's buffer, sequenced in the right order and placed in the audio playback system. But in this system, the possibility of jitter and breaks in playing are very high. Some of the methods of reducing jitter and break in playback are: Forward Error Correction method and Interleaving, dealt in the following chapters. Then our optimized solution is proposed which is essentially a combination of both the methodologies.

2.LITERATURE SURVEY

2.1 PCM SOUND WAVE FILE FORMAT

The WAVE file format is a subset of Microsoft's RIFF specification for the storage of multimedia files. A RIFF file starts out with a file header followed by a sequence of data chunks. A WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub-chunks -- a "fmt " chunk specifying the data format and a "data" chunk containing the actual sample data [5].

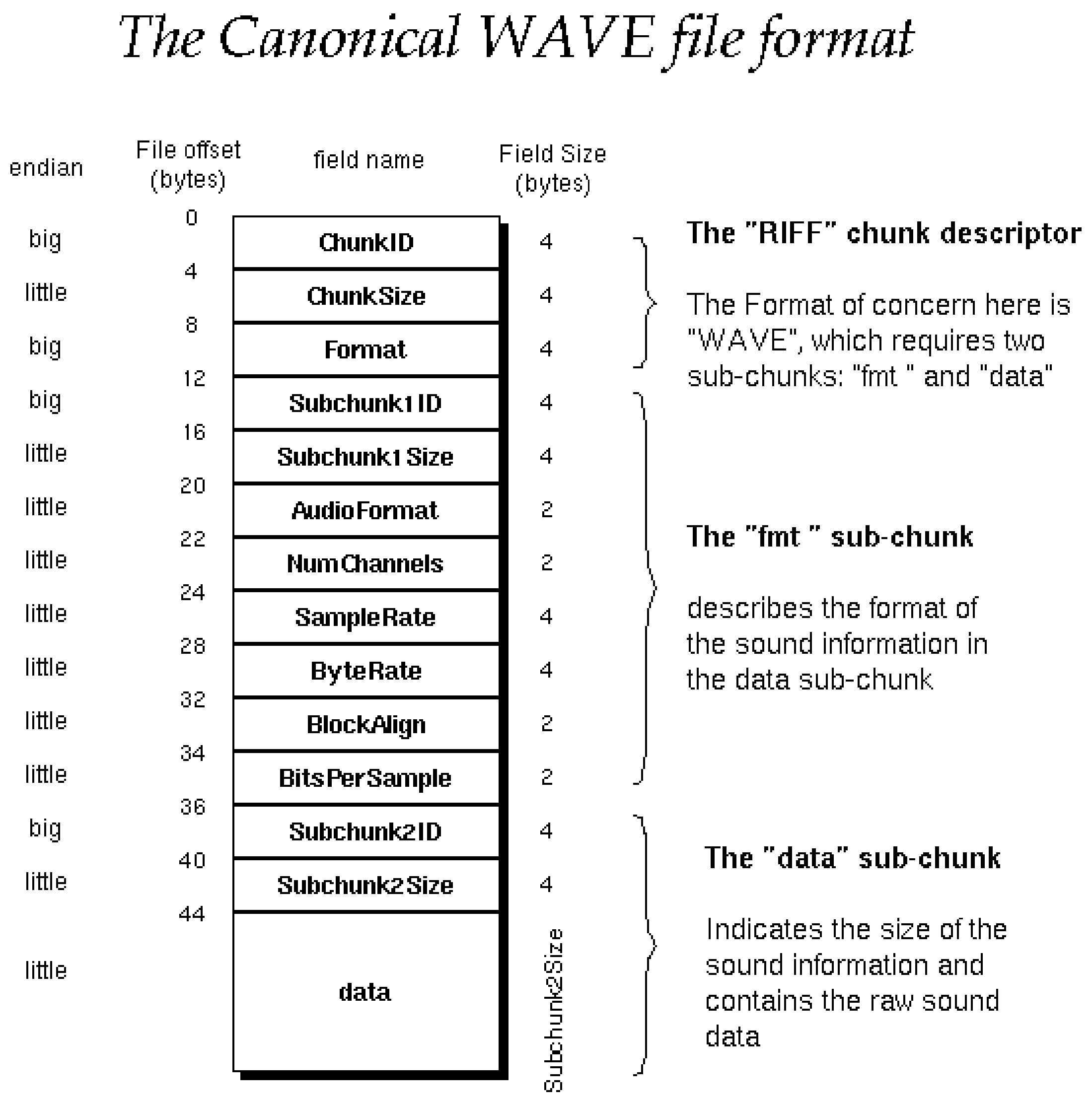

FIGURE 2.1 Canonical Wave Format

2.1.1 WAV FILE FORMAT DESCRIPTION

WAV files are probably the simplest of the common formats for storing audio samples. Unlike MPEG and other compressed formats, WAVs store samples "in the raw" format where no pre-processing is required other than formatting the data.

Each Chunk breaks down as follows:

RIFF Chunk (12 bytes in length total)

Byte Number	
0 – 3	"RIFF" (ASCII Characters)
4 – 7	Total Length Of Package To Follow (Binary, little endian)
8 – 11	"WAVE" (ASCII Characters)

FIGURE 2.2 RIFF Chunk

FORMAT Chunk (24 bytes in length total)

Byte Number	
0 – 3	"fmt_" (ASCII Characters)
4 – 7	Length Of FORMAT Chunk (Binary, always 0x10)
8 – 9	Always 0x01
10 – 11	Channel Numbers (Always 0x01=Mono, 0x02=Stereo)
12 – 15	Sample Rate (Binary, in Hz)
16 – 19	Bytes Per Second

20 – 21	Bytes Per Sample: 1=8 bit Mono, 2=8 bit Stereo or 16 bit Mono, 4=16 bit Stereo
22 – 23	Bits Per Sample

FIGURE 2.3 FORMAT Chunk

DATA Chunk

Byte Number	
0 – 3	"data" (ASCII Characters)
4 – 7	Length Of Data To Follow
8 – end	Data (Samples)

FIGURE 2.4 DATA Chunk

The easiest approach to this file format might be to look at an actual WAV file to see how data is stored. In this case, let us examine DING.WAV, which is standard with all Windows packages. DING.WAV is an 8-bit, mono, 22.050 KHz WAV file of 11,598 bytes in length. Lets begin by looking at the header of the file.

```
246E:0100  52 49 46 46 46 2D 00 00-57 41 56 45 66 6D 74 20   RIFFF-..WAVEfmt..
246E:0110  10 00 00 00 01 00 01 00-22 56 00 00 22 56 00 00   ......…"V.."V………..
246E:0120  01 00 08 00 64 61 74 61-22 2D 00 00 80 80 80 80   ..……data"-....………
246E:0130  80 80 80 80 80 80 80 80-80 80 80 80 80 80 80 80   ..............……………..
246E:0140  80 80 80 80 80 80 80 80-80 80 80 80 80 80 80 80   ..............…………….
```

As expected, the file begins with the ASCII characters "RIFF" identifying it as a WAV file. The next four bytes identifies the length, 0x2D46 bytes (11590 bytes in decimal), which is the length of the entire file minus the 8 bytes for the "RIFF" and length (11598 -

11590 = 8 bytes). The ASCII characters for "WAVE" and "fmt " follow. Next, we find the value 0x00000010 in the first 4 bytes (length of format chunk: always constant at 0x10). The next four bytes are 0x0001 (Always) and 0x0001 (A mono WAV, one channel used).

Since this is a 8-bit WAV, the sample rate and the bytes/second are the same at 0x00005622 or 22,050 in decimal. For a 16-bit stereo WAV the bytes/sec would be 4 times the sample rate. The next 2 bytes show the number of bytes per sample to be 0x0001 (8-bit mono) and the number of bits per sample to be 0x0008.

Finally, the ASCII characters for "data" appear followed by 0x00002D22 (11,554 decimal), which is the number of bytes of data to follow (actual samples). The data is a value from 0x00 to 0xFF. In the example above 0x80 would represent "0" or silence on the output since the DAC, which is used to playback samples is a bipolar device (i.e. a value of 0x00 would output a negative voltage and a value of 0xFF would output a positive voltage at the output of the DAC on the sound card).

RIFF stands for Resource Interchange File Format. The default byte ordering assumed for WAVE data files is little endian. The sample data must end on an even byte boundary. 8-bit samples are stored as unsigned bytes, ranging from 0 to 255. 16-bit samples are stored as 2's-complement signed integers, ranging from -32768 to 32767. There may be additional subchunks in a Wave data stream. If so, each will have a char[4] SubChunkID, and unsigned long SubChunkSize, and SubChunkSize amount of data.

2.2 ERROR CONCEALMENT

We consider a number of techniques for error concealment which may be initiated by the receiver of an audio stream and do not require assistance from the sender. These techniques are of use when sender-based recovery schemes fail to correct all loss, or when the sender of a stream is unable to participate in the recovery.

Error concealment schemes rely on producing a replacement for a lost packet, which is similar to the original. This is possible since audio signals exhibit large amounts of short-term self-similarity. As such, these techniques work for relatively small loss rates (~< 15 percent) and for small packets (4-40 ms). When the loss length approaches the length of a phoneme (5-100 ms) these techniques break down, since whole phonemes may be missed by the listener [4]. It is clear that error concealment schemes are not a substitute for sender-based repair, but rather work in tandem with it. A sender-based scheme is used to repair most losses, leaving a small number of isolated gaps to be repaired. Once the effective loss rate has been reduced in this way, error concealment forms a cheap and effective means of patching over the remaining loss.

Error concealment can be broadly classified into three categories:

1. Insertion-based schemes repair losses by inserting a fill-in packet. This fill-in is usually very simple: silence or noise is common, as is repetition of the previous packet. Such techniques are easy to implement but, with the exception of repetition, have poor performance.

2. Interpolation-based schemes use some form of pattern matching and interpolation to derive a replacement packet that is expected to be similar to the lost packet. These techniques are more difficult to implement and require more processing when compared with insertion-based schemes, which leads to improved performance.

3. Regeneration-based schemes derive the actual state from packets surrounding the loss and generate a replacement for the lost packet from it. This process though expensive to implement, yields good results.

The following section discusses each of these categories in detail.

2.2.1 INSERTION BASED REPAIR

Insertion-based repair schemes derive a replacement for a lost packet by inserting a simple fill-in. The simplest case is splicing [2], where a zero-length fill-in is used; an alternative is silence substitution, where a fill-in with the duration of the lost packet is substituted to maintain the timing of the stream. Better results are obtained by using noise or a repeat of the previous packet as the replacement. The distinguishing feature of insertion-based repair techniques is that the characteristics of the signal are not used to aid reconstruction. This makes these methods simple to implement, but results in generally poor performance.

2.2.1.1.SPLICING

Splicing together the audio on either side of the loss can conceal lost units so that no gaps are left due to a missing packet but the timing of the stream is disrupted. Low loss rates and short clipping lengths (4-16 ms) fair best, but the results were intolerable for losses above 3 percent [2]. The use of splicing can also interfere with the adaptive play out buffer required in a packet audio system, because it makes a step reduction in the amount of data available to buffer. The adaptive play out buffer is used to allow the reordering of misordered packets and removal of network timing jitter but poor performance of this buffer can adversely affect the quality of the entire system. It is therefore clear, that splicing together audio on either side of a lost unit is not an acceptable repair technique.

2.2.1.2.SILENCE SUBSTITUTION

Silence substitution fills the gap left by a lost packet with silence in order to maintain the timing relationship between the surrounding packets. It is only effective with short packet lengths (< 4 ms) and low loss rates (< 2 percent), making it suitable for interleaved audio over low-loss paths [2]. The performance of silence substitution degrades rapidly as packet sizes increase, and quality is unacceptably bad for the 40 ms packet size in common use in network audio conferencing tools. Despite this, the use of silence substitution is widespread, primarily because it is simple to implement.

2.2.1.3.NOISE SUBSTITUTION

Since silence substitution has been shown to perform poorly, an obvious next choice is noise substitution, where, instead of filling in the gap left by a lost packet with silence, background noise is inserted instead. A number of studies of the human perception of interrupted speech have shown that phonemic restoration, the ability of the human brain to subconsciously repair the missing segment of speech with the correct sound, occurs for speech repair using noise substitution but not for silence substitution [2]. In addition, when compared to silence, the use of white noise has been shown to yield both better quality as well as intelligibility. It is therefore recommended as a replacement for silence substitution.

2.2.1.4.REPETITION

Repetition replaces lost units with copies of the unit that arrived immediately before the loss. It has low computational complexity and performs reasonably well. The subjective quality of repetition can be improved by gradually adding repeated units. The use of repetition with fading is a good compromise between the other poorly performing insertion-based concealment techniques and the more complex interpolation-based and regenerative concealment methods.

2.2.2.INTERPOLATION BASED REPAIR

A number of error concealment techniques exist which attempt to interpolate from packets surrounding a loss to produce a replacement for that lost packet. The advantage of interpolation- based schemes over insertion-based techniques is that they account for the changing characteristics of a signal.

2.2.2.1.WAVEFORM SUBSTITUTION

Waveform substitution uses audio before and optionally after the loss to find a suitable signal to cover the loss. Audio engineers studied the use of waveform substitution in packet voice systems [2]. They examined both one- and two-sided techniques that use templates to locate suitable pitch patterns on either side of the loss. In the one-sided scheme, the pattern is repeated across the gap, but with the two-sided schemes interpolation occurs. The two-sided schemes generally performed better than one-sided schemes and both work better than silence substitution and packet repetition.

2.2.2.2.PITCH WAVEFORM REPLICATION

Audio engineers presented a refinement on waveform substitution by using a pitch detection algorithm on either side of the loss. Losses during unvoiced speech segments are repaired using packet repetition and voiced losses repeat a waveform of appropriate pitch length. The technique, known as pitch waveform replication, was found to work marginally better than waveform substitution [2].

2.2.2.3.TIME SCALE MODIFICATION

Time scale modification allows the audio on either side of the loss to be stretched across the loss. This scheme that finds overlapping vectors of pitch cycles on either side of the loss, offsets them to cover the loss, and averages them where they overlap. Although computationally demanding, the technique appears to work better than both waveform substitution and pitch waveform replication.

2.2.3.REGENERATION-BASED REPAIR

Regenerative repair techniques use knowledge of the audio compression algorithm to derive codec parameters, such that audio in a lost packet can be synthesized. These techniques are necessarily codec-dependent but perform well because of the large amount of state information used in the repair. Typically, they are also computationally intensive, to some extent.

2.2.3.1.INTERPOLATION OF TRANSMITTED STATE

For codecs based on transform coding or linear prediction, it is possible that the decoder can interpolate between states. For example, the ITU G.723.1 speech coder [2] interpolates the state of the linear predictor coefficients on either side of short losses and uses either a periodic excitation the same as the previous frame, or gain matched random number generator, depending on whether the signal was voiced or unvoiced. For longer losses, the reproduced signal is gradually faded. The advantages of codecs that can interpolate state rather than recoding the audio on either side of the loss is that there are no boundary effects due to changing codecs, and the computational load remains approximately constant. However, it should be noted that codecs where interpolation may be applied typically have high processing demands.

2.2.3.2.MODEL - BASED RECOVERY

In model-based recovery the speech on one, or both, sides of the loss is fitted to a model that is used to generate speech to cover the period loss. In the latest developments, interleaved encoded speech is repaired by combining the results of autoregressive analysis on the last received set of samples with an estimate of the excitation made for the loss period [2]. The technique works well for two reasons: the size of the interleaved blocks (8/16 ms) is short enough to ensure that the speech characteristics of the last received block have a high probability of being relevant. The majority of low-bit-rate speech codecs use an autoregressive model in conjunction with an excitation signal.

2.3 AUDIO INTERPOLATION – AN OVERVIEW

Audio interpolation is a method of making digital audio sound better than it really is. These days, almost every digital playback device (CD players, digital receivers, sound cards, etc) uses interpolation to improve our listening experience. And with computers ever increasing in speed, it's now possible to use software implementations as well without a serious decrease in performance. Let us first consider how sound is digitized. As a sound wave enters an ADC (analog to digital converter) samples are taken at timed intervals. Higher sample rates produce better recordings, but also use more memory. So, despite the drawbacks of lower quality, people sometimes use lower sample rates to conserve space. The following is a 1046Hz sine wave being sampled at 8kHz. The black dots in FIGURE 2.5 represent the samples.

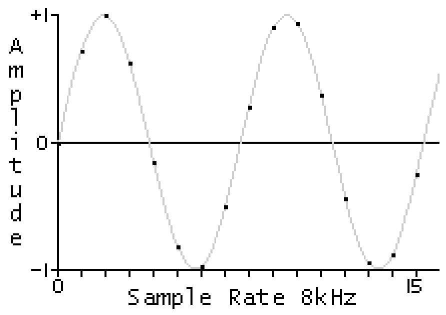

FIGURE 2.5 Sampled Sine Wave at 8kHz

For this example, the rate we want to output at is 32kHz. To accomplish this we just send

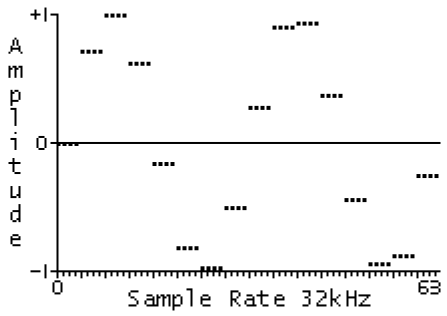

FIGURE 2.6 Sampled Sine Wave at 32 kHz

each sample four times, as indicated in the above figure .

This works effectively, but if we look at the actual sound wave being output, we see that there is a vague resemblance to the original. Also notice the hard edges after each sample. This gives lower sampled static sound.

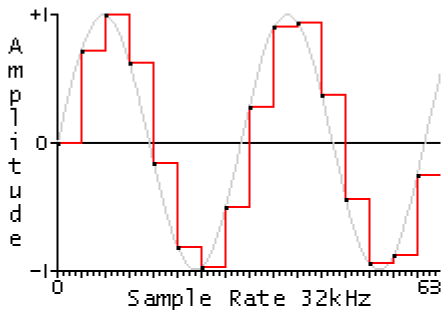

FIGURE 2.7 Comparison of Original Wave with Sampled Output

What interpolation does is instead of repeating each sample it creates new samples between the originals. Thus a smoother wave form is output. There are different methods that can be used, but by far the easiest, fastest, and most common is linear interpolation. This method compares two original samples then, based on this information, creates new samples that change at an even rate.

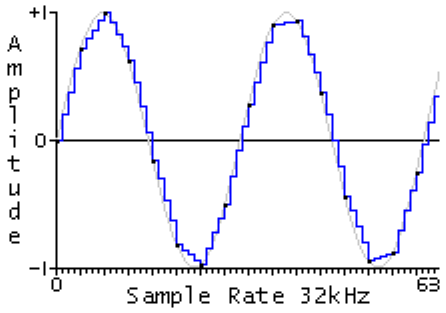

FIGURE 2.8 Linear Interpolated Wave

The problem with linear interpolation is that it makes straight lines between each sample and sound waves don't follow straight paths (with the exception of synthesized sounds like a saw tooth). Is it possible to recreate the natural curves of the sound wave? Yes, to an extent. To illustrate another method, let us consider the same situation (a 1046Hz sound wave sampled at 8kHz), but use a more complicated sound wave, i.e. tunes from a guitar. FIGURE 2.9 is the original sound with markers placed at the sampling points and FIGURE 2.10 is the playback with linear interpolation.

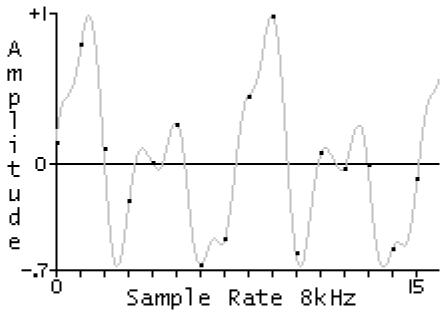

FIGURE 2.9 Original Wave Form With Sampled Points

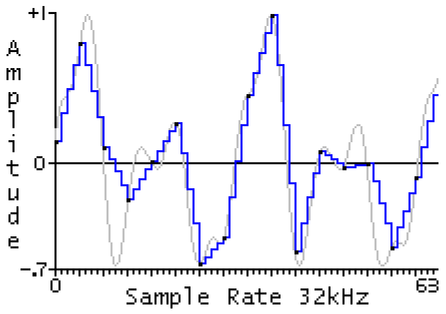

FIGURE 2.10 Linear Interpolated Wave

Cubic interpolation is basically an algorithm that tries to guess where the missing points should be, rather than plot them in a neat line. Almost all graphic and drawing programs have a spline tool where you lay down points that are then connected by means of a curving line. This same method can be applied to audio by using the samples as points on a plane.

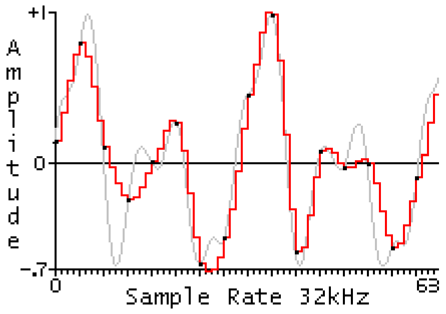

FIGURE 2.11 Cubic Spline Interpolated Wave

Though cubic interpolation is a better method than linear, it is also much slower. Either way, we notice that neither method completely reconstructs the wave. Peaks that are lost between samples during digitizing are lost permanently. Even CD's, with a sample rate of 44.1kHz, can't effectively reproduce frequencies above 10kHz. However, our ears have a hard enough time discerning those sounds anyway, so it probably doesn't matter.

2.4.LOSS MITIGATION SCHEMES

In the following sections, three loss mitigation schemes are discussed. These schemes have been discussed in the literature a number of times, and found to be of use in a number of scenarios. Each technique is briefly described, and its advantages and disadvantages are mentioned thereof.

2.4.1.RETRANSMISSION

Retransmission of lost packets is one of the means by which loss may be repaired. It is clearly of value in non-interactive applications, with relaxed delay bounds, but the delay imposed means that it does not typically perform well for interactive use. In addition to the possibly high latency, there is a potentially large bandwidth overhead associated with retransmission. Not only are units of data sent multiple times but also additional control traffic must flow to request retransmission. In our case the overhead of requesting retransmission for most packets may be such that the use of forward error correction is more acceptable. This leads to a natural synergy between the two mechanisms, with a forward error correction scheme being used to repair all single packet losses, and those receivers experiencing burst losses, and willing to accept the additional latency, using retransmission based repair as an additional recovery mechanism. In particular, protocols which use retransmission but bound the number of retransmission requests allowed for a given unit of data may be appropriate. Such retransmission-based schemes work best when loss rates are relatively small. As loss rates increase, the overhead due to retransmission requests increases.

In the worst case, for every multicast packet, at least one receiver does not receive the packet, which means that every packet needs to be transmitted to the whole group at least twice." In cases such as this, it is clear that the use of retransmission is probably only appropriate as a secondary technique to repair losses, which are not repaired by FEC. An alternative combination of FEC and retransmission takes the approach of using parity FEC packets to repair multiple losses with a single retransmission, achieving substantial savings relative to pure retransmission. Furthermore, the retransmission of a unit of audio does not need to be identical to the original transmission: the unit can be recoded to a lower bandwidth if the overhead of retransmission is thought to be problematic. There is a natural synchrony with redundant transmission, and a protocol may be derived in which both redundant and retransmitted units may be accommodated. This allows receivers that cannot participate in the retransmission process to benefit from retransmitted units if they are operating with a sufficiently large playout delay. The use of retransmission allows for an interesting trade-off between the desired playback quality and the desired degree of latency inherent in the stream. Within a large session, the amount of latency, which can be tolerated varies greatly for different participants: some users desire to participate closely in a session, and hence require very low latency, whereas others are content to observe and can tolerate much higher latency.

Those participants who require low latency must receive the media stream without the benefit of retransmission-based repair. Others gain the benefit of the repair, but at the expense of increased delay. In order to reduce the overhead of retransmission, the retransmitted units may be piggybacked onto the ongoing transmission. This also allows retransmission to be recoded in a different format, to reduce the bandwidth overhead further. As an alternative, Forward Error Correction (FEC) information may be sent in response to retransmission requests, allowing a single retransmission to potentially repair several losses. The choice of a transmission request algorithm, which is both timely and network friendly is an area of current study.

2.4.2 FORWARD ERROR CORRECTION

Forward error correction (FEC) is the means by which repair data is added to a media stream, such that packet loss can be repaired by the receiver of that stream with no further reference to the sender. There are two classes of repair data which may be added to a stream: those which are independent of the contents of the stream and those which use knowledge of the stream to improve the repair process.

2.4.2.1 MEDIA INDEPENDENT FEC

There has been much interest in the provision of media-independent FEC using block or algebraic, codes to produce additional packets for transmission to aid the correction of losses. A number of media-independent FEC schemes have been proposed for use with streamed media. These techniques add redundant data, which is transmitted in separate packets, to a media stream. Traditionally, FEC techniques are described as loss detecting and/or loss correcting. In the case of streamed media, loss detection is provided by the sequence numbers in packets.

The redundant FEC data is typically calculated using the mathematics of finite fields. The simplest of finite field is the eXclusive-OR operation. Basic FEC schemes transmit k data packets with n-k parity packets allowing the reconstruction of the original data from any k of the n transmitted packets. Budge [1] applied the XOR operation across different combinations of the media data with the redundant data transmitted separately as parity packets. These vary the pattern of packets over which the parity is calculated, and hence have different bandwidth, latency and loss repair characteristics.

Parity-based FEC techniques have a significant advantage in that they are media independent, and provide exact repair for lost packets. In addition, the processing requirements are relatively light, especially when compared with some media-specific FEC (redundancy) schemes, which use very low bandwidth, but high complexity encodings. The disadvantage of parity based FEC is that the codings have higher latency in comparison with the media-specific schemes, which are discussed in following section. A number of FEC schemes exist which are based on higher-order finite fields, for example Reed-Solomon (RS) codes [6], which are more sophisticated and computationally demanding. These are usually structured so that they have good burst loss protection, although this typically comes at the expense of increased latency. Dependent on the observed loss patterns, such codes may give improved performance, compared to parity-based FEC. This approach has been advocated for use with streaming audio [6].

There are several advantages of Media Independent FEC schemes.

1. The operation of the FEC does not depend on the contents of the packets, and the repair is an exact replacement for a lost packet.

2. In addition, the computation required to derive the error correction packets is relatively small and simple to implement.

The disadvantages of these schemes are the additional delay imposed, increased bandwidth and difficult decoder implementation.

2.4.2.2 MEDIA-SPECIFIC FEC

The basis of media-specific FEC is to employ knowledge of a media compression scheme to achieve more efficient repair of a stream than can otherwise be achieved. To repair a stream subject to packet loss, it is necessary to add redundancy to that stream: some information is added which is not required in the absence of packet loss, but which can be used to recover from that loss.

The nature of a media stream affects the means by which redundancy is added. If units of media data are packets, or if multiple units are included in a packet, it is logical to use the unit as the level of redundancy, and to send duplicate units. By recoding the redundant copy of a unit, significant bandwidth savings may be made, at the expense of additional computational complexity and approximate repair. If media units span multiple packets, for instance sensitive audio frames, it is sensible to include redundancy directly within the output of a codec. In another approach, media data packets include multiple copies of key portions of the stream, separated to avoid the problems of packet loss. The advantages of this second approach is efficiency: the codec designer knows exactly which portions of the stream are most important to protect and less complex since each unit is coded only once. An alternative approach is to apply media-dependent FEC techniques to the most significant bits of a codec’s output, rather than applying it over the entire packet. Several codec descriptions include bit sensitivities that make this feasible. This approach has low computational cost and can be tailored to represent an arbitrary fraction of the transmitted data.

The use of media-specific FEC has the advantage of low-latency, with only a single-packet delay being added. This makes it suitable for interactive applications, where large end-to-end delays cannot be tolerated. In a unidirectional non-interactive environment it is possible to delay sending the redundant data, achieving improved performance in the presence of burst losses, at the expense of additional latency.

2.4.3 INTERLEAVING

When the unit size is smaller than the packet size, and end-to-end delay is unimportant, interleaving is a useful technique for reducing the effects of loss. Units are re-sequenced before transmission, so that originally adjacent units are separated by a guaranteed distance in the transmitted stream, and returned to their original order at the receiver. Interleaving disperses the effect of packet losses. If, for example, units are 5ms in length and packets 20ms (ie: 4 units per packet), then the first packet could contain units 1, 5, 9, 13; the second packet would contain units 2, 6, 10, 14; and so on. It can be seen that the loss of a single packet from an interleaved stream results in multiple small gaps in the reconstructed stream, as opposed to the single large gap, which would occur in a non-interleaved stream. In many cases it is easier to reconstruct a stream with such loss patterns, although this is clearly media dependent. Note that the size of the gaps is dependent on the degree of interleaving used, and can be made arbitrarily small at the expense of additional latency.

One of the main disadvantages of interleaving is that it increases latency. This limits the use of this technique for interactive applications, although it performs well for non-interactive use. The major advantage of interleaving is that it does not increase the bandwidth requirements of a stream.

2.5 SUMMARY

2.5.1.NONINTERACTIVE APPLICATIONS:

For one-to-many transmissions in the style of radio broadcasts, latency is of considerably less importance than quality. In addition, bandwidth efficiency is a concern since the receiver set is likely to be diverse and the group may include members behind low-speed links. The use of interleaving is compatible with both these requirements and is strongly recommended. Although interleaving drastically reduces the audible effects of lost packets, some form of error concealment will still be needed to compensate. In this case the use of a simple repair scheme, such as repetition with fading, is acceptable and will give good quality.

Retransmission-based repair is not appropriate for a multicast session, since the receiver set is likely to be heterogeneous. This leads to many retransmission requests for different packets and a large bandwidth overhead due to control traffic. For unicast sessions retransmission is more acceptable, particularly in low-loss scenarios.

A media-independent FEC scheme will perform better than a retransmission-based repair scheme, since a single FEC packet can correct many different losses and there is no control traffic overhead. The overhead due to the FEC data itself still persists, although this may be acceptable. In particular, FEC-protected streams allow for exact repair, while repair of interleaved streams is only approximate.

2.5.2.INTERACTIVE APPLICATIONS:

For interactive applications, such as IP telephony, the principal concern is minimizing end-to-end delay. It is acceptable to sacrifice some quality to meet delay requirements, provided that the result is intelligible. The delay imposed by the use of interleaving, retransmission, and media-independent FEC is not acceptable for these applications. While media-independent FEC schemes do exist that satisfy the delay requirements, these typically have high bandwidth overhead and are likely to be inappropriate for this reason. Our recommendation for interactive conferencing applications is that media-specific FEC be employed, since this has low latency and tunable bandwidth overhead. Repair is approximate due to the use of low-rate secondary encodings, but this is acceptable for this class of applications when used in conjunction with receiver-based error concealment.

2.5.3.ERROR CONCEALMENT:

Receivers must be prepared to accept some loss in an audio stream. The overhead involved in ensuring that all packets are received correctly, in both time and bandwidth, is such that some loss is unavoidable. Once this is accepted, the need for error concealment becomes apparent. Many current conferencing applications use silence substitution to fill the gaps left by packet loss, but it has been shown that this does not provide acceptable quality. A significant improvement is achieved by the use of packet repetition, which also has the advantages of being simple to implement and having low computational overhead. The other error concealment schemes discussed provide incremental improvements, with significantly greater complexity.

3. IMPLELMENTATION METHODOLOGY

3.1 INTRODUCTION:

A number of applications have emerged which use User Datagram Protocol (UDP) to deliver continuous media streams. Due to the unreliable nature of UDP packet delivery, the quality of the received stream will be adversely affected by packet loss. A number of techniques exist by which the effects of packet loss may be repaired. These techniques have a wide range of applicability and require varying degrees of protocol support. In our report, a number of such techniques were discussed, and recommendations for their applicability were made.

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 from text-based applications. An audio stream, for example, requires that data be received in a timely fashion and with not much emphasis on data loss. Data packets arriving beyond the threshold wait leads to a certain amount of breaks in the audio listened to .If a data packet arrives at the client too late, it misses the time slot during which it had 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. 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 approach for multicast uses a code created from Galois Field [8] 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, 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 certain 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 uses knowledge of the data type in adding encoding information. A lost packet is replaced by the redundant data transmitted with the next packet. When redundancy fails to repair the lost packet, a repetition based error concealment technique is used to fill the gap. The receiver is able to repair some of the lost data using the smaller, lower quality repair information when the stream is played out. Media-specific FEC is used with many audio applications because the audio format has various quality levels, and researchers [9] have shown that audio can be repaired using a lower quality sample of the lost information. 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. While FEC can be effective for repairing loss under some network conditions, under higher loss rates or burst loss the repair information may also be lost, making FEC ineffective.

3.2 FORWARD ERROR CORRECTION

The basic idea of FEC is to add redundant information to the original packet stream. One can outline two FEC mechanisms. The first mechanism sends a redundant encoded chunk after every *n* chunk. The redundant chunk is obtained by exclusive OR-ing the *n* original chunks. In this manner if any one packet of group of *n+1 packet* is lost, the receiver can fully reconstruct the lost packet. But if two or more packets are lost, the receiver cannot fully reconstruct the lost packet. In our case the sender can create a nominal audio stream and a corresponding low-resolution, low-bit rate audio stream. The low-bit-rate stream is referred to as the redundant stream. The sender constructs the n^{th} packet or chunk from the nominal stream and appending to it the $(n\text{-}1)^{th}$ chunk from the redundant stream. In this manner, whenever there is a nonconsecutive packet loss, the receiver can conceal the loss by playing out the low-bit-rate encoded chunk that arrives with the subsequent packet.

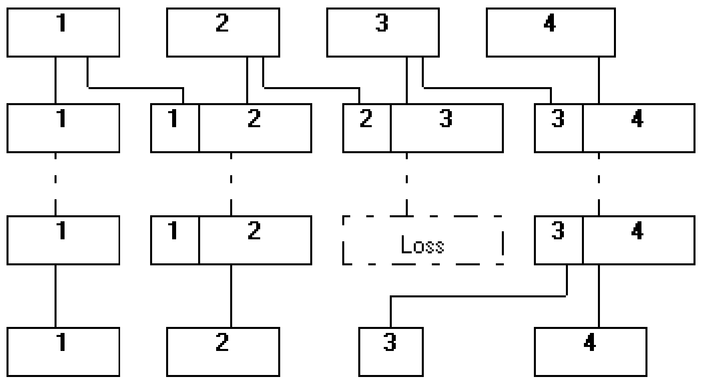

FIGURE 3.1 Forward Error Correction

In the above illustration, one can witness the loss of the 3rd frame. Since frames are constructed using forward error correction methodology, the 4th frame contains the low quality 3rd frame. So, it becomes very easy for the client to reconstruct frame 3 from the frame 4. Even if multiple frames are missed, filling can be done by this way. But if continuously many frames are lost, one cannot afford to choose this method. In this case interleaving method (discussed below) shall help.

3.3 INTERLEAVING

Interleaving is another alternative available where the sender re-sequences units of audio data before transmission, so that originally adjacent units are separated by a certain distance in the transmitted stream. Interleaving can mitigate the effect of packet losses. The idea of interleaving is to ensure playback continuously with possible vacuoles. That is, the methodology confirms delivery of uniformly separated frames. There is an illustration about the working furnished below.

If for example, audio data units are 5 milliseconds in length and a single data frame is 20 milliseconds (that is, 4 units per chunk) in length, then the first chunk could contain units 1, 5, 9, 13; the second chunk could contain 2, 6, 10 and 14 and so on.

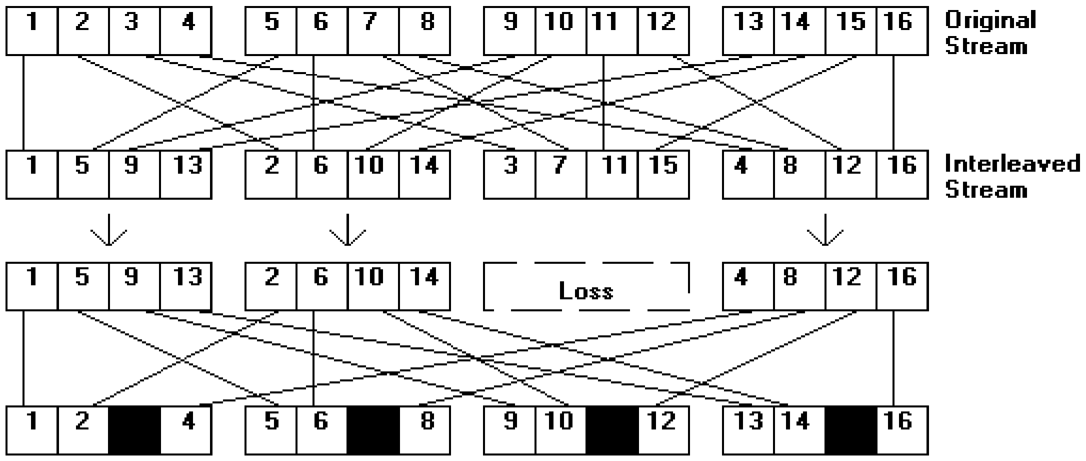

FIGURE 3.2 Interleaving

The method depicted above delivers uniformly separated data units. One can see that frame 3 (4 data units) is lost. Even if it is lost, the sequence of audio data is more or less filled up. As a worst case, if frame 2& 3 are lost, we might construct the following sequence: 1 - - 4 5 - - 8 9 - - 12 13 - - 16. A still worse case is to have lost 2, 3 & 4. Even in that case the client could receive frames: 1 - - - 5 - - - 9 - - - 13 - - - . If the audio data that is being played is not very sensitive, this gap would not be a big hearing ailment.

3.4 INTERLEAVED FORWARD ERROR CORRECTION

We propose to have Interleaved FEC, which essentially combines the best from both the above methodologies. With the interleaved data frame transfer, one can be guaranteed with uniformly distributed frames even in the worst case. What if the received frames are appended with low quality next frame? This would guarantee more packed uniformly distributed frames of data. The illustration shown below depicts the operation of interleaved FEC.

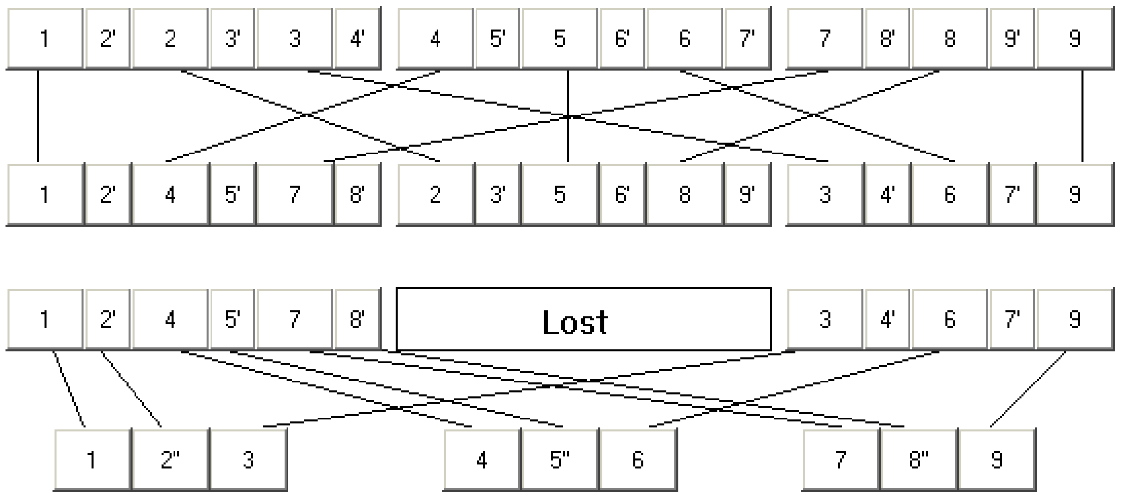

FIGURE 3.3 Interleaved Forward Error Correction

From the figure, it is very clear that even in worse conditions, the playing of the data is not getting affected to a larger extent. Although, we tend to fill the gaps with low quality frames, it's always a better option to promise contiguous playback.

The overhead on the system can be relatively compromised against the output quality that is being achieved. Using interpolation techniques, the low quality small frame is reconstructed to form the frame of size equal to the size of a normal data frame (*a constant*). Frame size in the audio playback input buffer being a constant determines the frame reconstructed.

3.5.NETWORK LOSS CHARACTERISTICS

If it is desired to repair a media stream subject to packet loss, it is useful to have some knowledge of the loss characteristics, which are likely to be encountered. A number of studies have been conducted on the loss characteristics of the audio packets and although the results vary somewhat, the broad conclusion is clear: in a large conference it is inevitable that some receivers will experience packet loss [7]. Packet traces show a session in which most receivers experience loss in the range 2-5%, with a somewhat smaller number seeing significantly higher loss rates. Other studies have presented broadly similar results .It has also been shown that the vast majority of losses are of single packets [9]. Burst losses of two or more packets are around an order of magnitude less frequent than single packet loss, although they do occur more often than would be expected from a purely random process. Longer burst losses (of the order of tens of packets) occur infrequently. These results are consistent with a network, here small amounts of transient congestion cause the majority of packet loss. In few cases, a network link is found to be severely overloaded, and large amount of loss results. The primary focus of a repair scheme must, therefore be to correct single packet loss, since this is by far the most frequent occurrence and our proposed methodology focuses on this issue. It is desirable that losses of a relatively small number of consecutive packets may also be repaired, since such losses represent a small but noticeable fraction of observed losses. The correction of large bursts of loss is of considerably less importance.

3.6.SERVER SIDE DESIGN

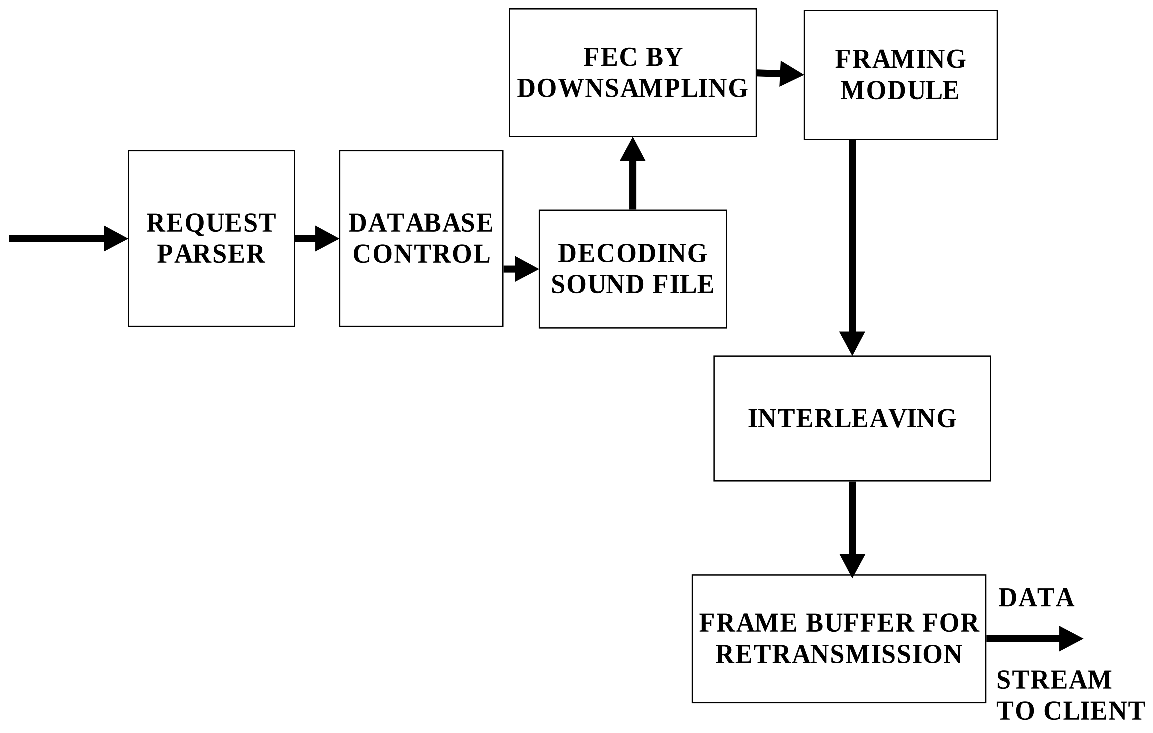

FIGURE 3.4 Server Side Design

3.6.1.REQUEST PARSER

The Server waits for a request from the client. The server on request then identifies the request as whether it should play an audio file or it should retransmit a frame. Accordingly the request is sent for processing.

3.6.2.DATABASE CONTROL

The server searches the requested data file in the audio database. The client side is acknowledged about the availability of the file. If the audio file is not found, then error messages are sent to the client side or else the audio file is sent for decoding and transmission.

3.6.3.DECODING SOUND FILE

. The audio file is then decoded and its header is sent separately as a frame to the client without any modification. The header contains all the required information such as Sampling rate, Mono or Stereo, size of data samples, etc. Then the data samples alone are read and they are sent for conversion into frames.

3.6.4.FORWARD ERROR CORRECTION BY DOWNSAMPLING

In order to implement FEC, the data samples are also downsampled so that they can be appended with the high quality original data samples. This low quality downsampled parts are appended with the high quality parts of the previous frames.

3.6.5.FRAMING MODULE

Here the Forward error corrected frames are converted into complete packets by adding relevant header information to the frames. Some of the header information includes sequence number, end of file flags, frame type, etc.

3.6.6.INTERLEAVING

The frame sequence of the frames are completely altered using the method of interleaving. Copies of the frames are saved in a list and while doing so they are interleaved and positioned so that sequential loss of packets can be avoided while transmitting the packets to the client.

3.6.7.FRAME BUFFER

The linked list of the frames so created is maintained as a buffer, so that in case of retransmission, corresponding packets are retransmitted accordingly. The frames stored here will be in the interleaved format. The frame buffer is refreshed after waiting for a certain time interval.

3.7.CLIENT SIDE DESIGN

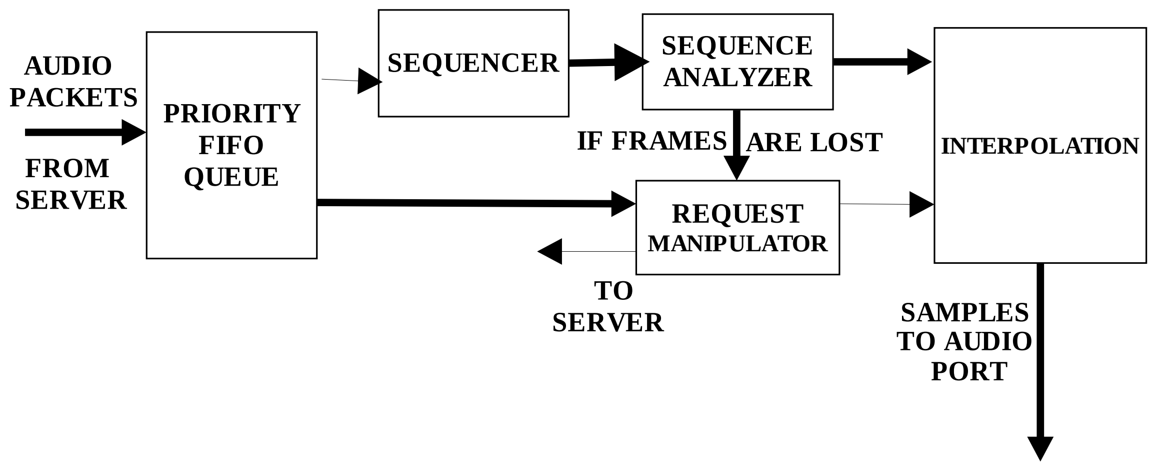

FIGURE 3.5 Client Side Design

3.7.1.PRIORITY FIFO QUEUE

The audio packets that are received from the server are maintained in a queue wherein the retransmitted frames are given more priority to be sent to the audio device so that the audio file can be played without any breaks. The queue consists of packets in the order in which the server had sent to them.

3.7.2.SEQUENCER

The audio packets are received from the server and maintained in a queue will be in disarray as they are interpolated. So they need to be sequenced so that the audio file can be played out in proper sequence. The sequenced frames are then sent to the Sequence analyzer.

3.7.3.SEQUENCE ANALYZER

The sequenced audio packets are then analyzed for any missing frames. If the missing frames are less in number and if they can be interpolated then there is no request for retransmission and the control is passed to the interpolation module. In case too many frames are missing, the control is passed onto the request manipulator.

3.7.4.REQUEST MANIPULATOR

The missing frames are analyzed and only the key frames are requested for retransmission. By key frames, we mean the minimum frames that will be required to reproduce the audio sequence by making full use of both the high and low quality parts of the frames.

3.7.5.INTERPOLATION

Here the frames are separated from their low quality appendages. If the situation requires the low quality frames then, they are interpolated and used as a high quality frame. The information on which frames to interpolate is provided by the sequence analyzer and the request manipulator in case of retransmissions. The data samples are then played sequentially using the audio interface.

4.CODING & TESTING

4.1 CODING REFERENCES

SERVER SIDE

1. **HEAD.H**

It includes the definitions of all the different structures used in the source code and the required header files.

2. **MSERVER.C**

It waits for a client to request an audio file and then forwards the audio file to other modules, so that it can be processed and transmitted to the client.

3. **PF.C**

Here the audio file is decoded so that the sampling rate, number of channels, number of samples and other required information can be known and these header information are sent to the client for they are required for playing the audio file. Also the data samples are read and sent as small frames to **SENDFRAME** module. They are appended with the low quality data samples got by downsampling the audio file so as to implement Forward Error Correction.

4. **SENDF.C**

Here, necessary header information is added to the frames and they are transmitted to the client. Here, only the header information of the WAV files is transmitted.

5. **SENDFRAME.C**

Here the data samples that are read from the WAV files are sent to the client side as numerous frames after adding proper header information.

6. FRAMEBUFFER.C

Here, the Forward Error corrected frames are interleaved and stored as a group of fifty frames and then transmitted to the client by setting a proper end of file tag to the frame so that it can be easily identified at the client side.

CLIENT SIDE

7. SCL.C

In this module the client sends the request to the server for an audio file. Network parameters are initialized here and acknowledgements are sent to the server.

8. DLIST.C

Here the frames from the server are received in the order they were sent. Later they are sent for reordering and analyzing the sequence.

9. SEQUENCER.C

The received frames are reordered in this module .The lost frames and the key frames are reported to the server.

10. PLAY.C

This module plays the samples at a desired rate. Here the sound device is opened using necessary parameters such as number of channels, sampling rate and format (little/big endian). Then the sound device is opened for writing the data in binary mode.

4.2 EXPERIMENTAL EVALUATION

We have used the proposed Interleaved Forward Error Correction technique and obtained the following output waveforms. The output waveforms are illustrated in the figures (next page) along with their input waveforms (WAV Files).

SCALE: X-AXIS –FREQUENCY (KHz)
Y-AXIS-AMPLITUDE

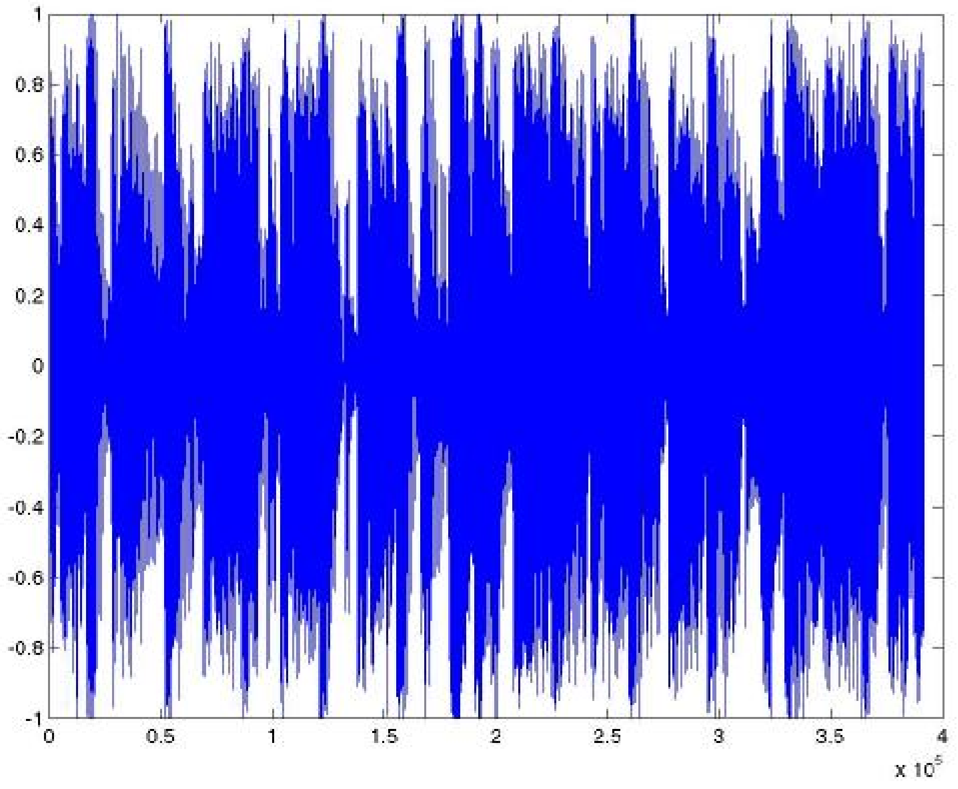

FIGURE 4.1 Waveform Analysis of Original Wave

The above figure illustrates the original waveform. FIGURE 4.2(next page) indicates the waveform obtained as a result of our proposed technique (Interleaved Forward Error Correction). Analyzing the two waveforms, we notice the similarity in them.

SCALE: X-AXIS –FREQUENCY (KHz)
Y-AXIS-AMPLITUDE

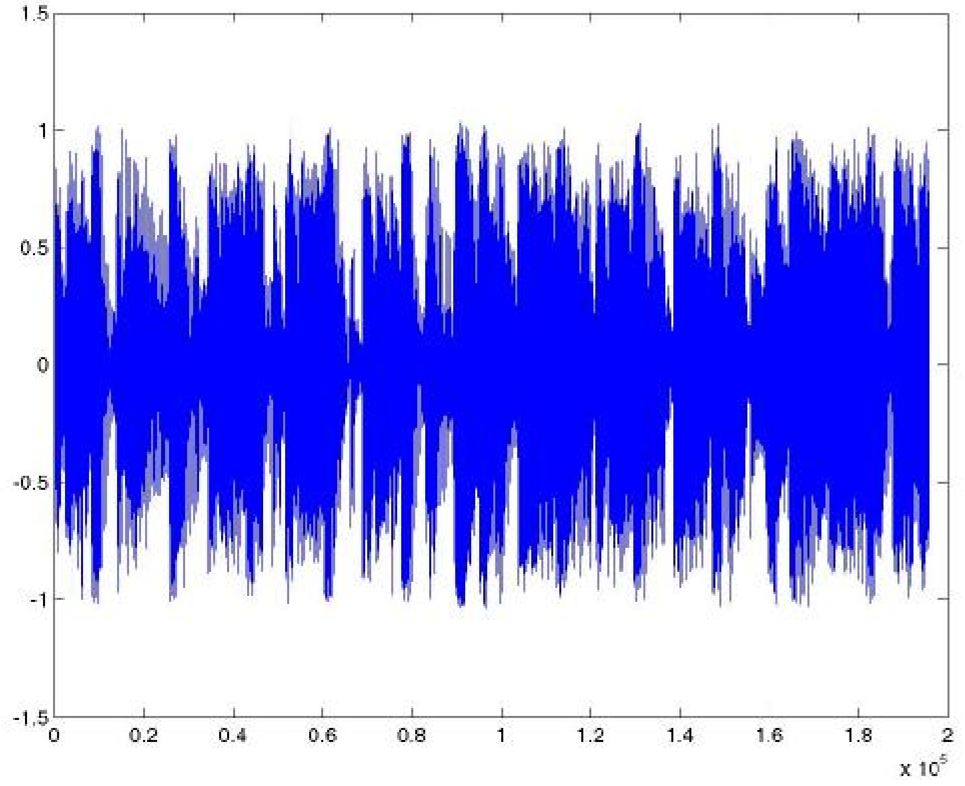

FIGURE 4.2 Waveform Analysis of Output Wave after Interpolation

5.CONCLUSION AND FUTURE ENHANCEMENTS

5.1 CONCLUSION

The growth in power of today's computers and networks present the opportunity for high-quality audio across the Internet to the desktop. Internet audio and video typically does not use TCP, the de facto 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 our project, we have proposed an Interleaved FEC protocol that combines the advantages of Forward Error Correction and Interleaving. Evaluation under network conditions with a variety of loss rates demonstrates our protocol provides 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 design of our proposed methodology and
2) An implementation of our Interleaved Forward Error Correction technique in a network environment.

5.2 FUTURE ENHANCEMENTS

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 play out when one packet is lost. After some waiting, the secondary frame will be extracted, decoded and played after the one frame halt, causing the play out to be uneven,

even if there is no loss. Past works have shown that this jitter can be as detrimental to perceived quality as is data loss [8].

1. Analysis on how much jitter, FEC introduces to the audio play out and how to solve this problem can be an interesting area of future research.

2. It is also possible to combine FEC with selective retransmission.

3.The addition of FEC repair data to a media stream is an effective means by which that stream may be protected against packet loss. However, application designers should be aware that the addition of large amounts of repair data when loss is detected will increase network congestion and hence packet loss, leading to a worsening of the problem, which FEC intended to solve. This is particularly important when sending to large multicast groups, since network heterogeneity causes different sets of receivers to observe widely varying loss rates: low-capacity regions of the network suffer congestion, while high-capacity regions are underutilized. Existing techniques typically use some form of layered encoding of data sent at different rates over multiple multicast groups, with receivers joining and leaving groups in response to long-term congestion and with FEC employed to overcome short-term transient congestion. Such a scheme pushes the burden of adaptation from the server to the clients with the client choosing the number of layers (groups) based on the packet loss rate they observe. Since the different layers contain data sent at different rates, clients will receive different quality of service depending on the number of layers they are able to join. Layered encoding schemes are expected to provide a congestion control solution suitable for streaming audio applications. However, this work is not yet complete and in the long term, effective congestion control must be incorporated with our proposed methodology

6.REFERENCES

[1] Kurose&Ross, *Computer Networking - A top down approach towards the Internet*, Prentice Hall Publication

[2] C Perkins, O Hardson and V Hardmon ,” A Survey of Packet Loss Recovery Techniques for Streaming Audio” *IEEE Network Magazine* Sept/Oct 1998 pp 40-47

[3] ITU-T Recommendation H324 Version 4 Annex A “Packet based Multimedia communication”, *ITU* Nov 2000

[4] G.Carle and E. W. Biersack, “Survey of error recovery techniques for Ipbased audio-visual multicast applications,” IEEE Network, vol. 11, no. 6, Nov./Dec. 1997, pp. 24–36.

[5] http://netghost.narod.ru/gff/graphics/summary/micriff.htm -Summary of RIFF Format

[6] J.C. Bolot and A. Vega-Garcia, “The case for FEC based error control for packet audio in the Internet,” to appear, ACMultimedia Sys.

[7] J.-C. Bolot and A. Vega-Garcia, “Control mechanisms for packet audio in the Internet,” Proc. IEEE INFOCOM ’96, 1996.

[8] C. S. Perkins and O. Hodson, “Options for repair of streaming media,” IETF Audio/Video Transport WG, RFC2354, June 1998.

[9] J.C. Bolot, “End-to-end packet delay and loss behavior in the internet, ”Proc. ACM SIGCOMM ’93, San Francisco, Sept. 1993, pp. 289–98.

APPENDIX –I: SOURCE CODE

HEAD.H

```
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<alloc.h>
#include<netinet/in.h>
#include<string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <vector>
#include <fcntl.h>
```

**//	Support from sound output**

```
#if defined(__linux__)
#  define dsp_device "/dev/dsp"
#  include <linux/soundcard.h>
#endif

struct header
{
int sequence_number;
int frame_type;
int check_sum;
int size_of_frame;
int compression_index;
int end_of_file;
};
struct frame
{
struct header h;
unsigned int data[2000];
unsigned char data1[100];
int crc_check;
struct frame *next;
};
```

MSERVER.C

```
#include "head.h"
#include "pf.c"
int main(int argc,char **argv)
{
int sockfd,n=0;
char str[]="Specify the file name : \0",p[30];
socklen_t len;
struct displist d;
struct sockaddr_in servaddr,cliaddr;
```

**//	initializes network parameters**

```
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(9877);
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
bind(sockfd,(const struct sockaddr *) &servaddr,sizeof(servaddr));
```

```
        len=sizeof(cliaddr);
        n=0;
//      server is made to wait for a request from a client
        while(n==0)
        n=recvfrom(sockfd,p,sizeof(p),0,(struct sockaddr *)&cliaddr,&len);
        if(strcmp(p,"start\0")==0)
        {
        sendto(sockfd,str,sizeof(str),0,(const struct sockaddr *)&cliaddr,len);
        n=0;
//      server receives the audio file to be played
        while(n==0)
        n=recvfrom(sockfd,p,sizeof(p),0,(struct sockaddr *)&cliaddr,&len);
        partf(p,sockfd,(const struct sockaddr *) &cliaddr,len);
        }
        return 0;
}
```

PF.C

```
        #include "sendframe.c"
        #include "sendf.c"
        void partf(char fn[100],int sckid,const struct sockaddr *pservaddr,socklen_t len)
        {
        FILE  *fp1,*fp2;
        int l=0,seq=-1,m=0;
        unsigned char s[1500];
        unsigned char ch[4],ch1[2];
        unsigned int c,d,e,i,j,s1[300],s2[200],k;
//      two pointers are made to point to the requested audio file
        fp1=fopen(fn,"rb");
        fp2=fopen(fn,"rb");
        sendto(sckid,"File found\0",10,0,pservaddr,len);
```

```
//          decoding of header information of the audio file
            for (l=0;l<10;l++){
            fread((void *)ch,sizeof(ch),1,fp1);
            s[m++]=ch[0];
            s[m++]=ch[1];
            s[m++]=ch[2];
            s[m++]=ch[3];
            }
            fread((void *)ch,sizeof(ch),1,fp2);
            printf("%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
            fread((void *)&c,sizeof(c),1,fp2);
            printf("\nSize %u",c);
            fread((void *)ch,sizeof(ch),1,fp2);
            printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
            fread((void *)ch,sizeof(ch),1,fp2);
            printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
            fread((void *)&c,sizeof(c),1,fp2);
            printf("\nsize %u",c);
            fread((void *)&c,sizeof(c),1,fp2);
            e = c;
            d = c & 0xffff;
            printf("\nfmt %u",d);
            if (d!=1) {
            printf("\nthis is not PCM wav file\n");
            return;
            }
            d=c>>16;
            printf("\ncha %u",d);
            fread((void *)&c,sizeof(c),1,fp2);
            printf("\nNSPS %u",c);
            fread((void *)&c,sizeof(c),1,fp2);
```

```
printf("\nNABPS %u",c);
fread((void *)&c,sizeof(c),1,fp2);
e = c;
d = c & 0xffff;
printf("\nblk %u",d);
d = e >>16;
printf("\nbps %u",d);
fread((void *)ch,sizeof(ch),1,fp1);
fread((void *)ch,sizeof(ch),1,fp2);
if (ch[0]!='d'){
for (j=0;j<2;j++){
fread((void *)ch,sizeof(ch),1,fp1);
fread((void *)ch,sizeof(ch),1,fp2);
s[m++]=ch[0];
s[m++]=ch[1];
s[m++]=ch[2];
s[m++]=ch[3];
}
fread((void *)ch1,sizeof(ch1),1,fp1);
fread((void *)ch1,sizeof(ch1),1,fp2);
s[m++]=ch1[0];
s[m++]=ch1[1];
fread((void *)ch,sizeof(ch),1,fp1);
fread((void *)ch,sizeof(ch),1,fp2);
s[m++]=ch[0];
s[m++]=ch[1];
s[m++]=ch[2];
s[m++]=ch[3];
printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
}
else printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
```

```
        fread((void *)&c,sizeof(c),1,fp2);
        printf("\nSize %u\n",c);
        k=c;
        fread((void *)ch,sizeof(ch),1,fp1);
        s[m++]=ch[0];
        s[m++]=ch[1];
        s[m++]=ch[2];
        s[m++]=ch[3];
//      despatching the header information alone
        sendf(0,m,seq++,s,sckid,pservaddr,len);
        fread((void *)s,sizeof(unsigned char),1000,fp2);
        m=0;
        j=0;

/*      samples are read and Forward Error Correction is implemented by
        downsampling
        and reading samples and then appending them with the unchanged samples
        of previous frames  */
        while(j<k){
        fread((void *)&c,sizeof(unsigned int),1,fp1);
        s1[m++]=c;
        j+=4;
        if (m==250){
//      low quality appendage by downsampling
        while(m<375){
        fread((void *)&c,sizeof(unsigned int),1,fp2);
        s1[m++]=c;
        fread((void *)&c,sizeof(unsigned int),1,fp2);
        }
        sendframe(0,1500,seq++,s1,sckid,pservaddr,len);
        m=0;
        }
        }
        sendframe(1,m*4,seq++,s1,sckid,pservaddr,len);
        fclose(fp2);
        fclose(fp1);
        return;
}
```

SENDF.C

```
//      constructs a complete frame only for the header information of the audio file
        void sendf(int endof,int leng,int num,unsigned char s[100],int sockfd,const struct
        sockaddr *claddr,socklen_t len)
        {
        struct frame f;
        f.h.sequence_number=num;
        f.h.frame_type=2;
        f.h.size_of_frame=leng;
        f.h.compression_index=3;
        f.h.end_of_file=endof;
        f.h.check_sum=f.h.sequence_number^f.h.frame_type^f.h.size_of_frame^f.h.comp
        ression_index;
        for(int i=0;i<leng;i++)
        f.data1[i]=s[i];
        printf("seqno : %d  \t size : %d\n",num,leng);
//      sends header information to the client
        sendto(sockfd,(const void *)&f,sizeof(f),0,claddr,len);
        return;
}
```

SENDFRAME.C

/*   makes a complete frame of data samples by adding header information for a set of data samples */

```
#include "framebuffer.c"
        void sendframe(int endof,int leng,int num,unsigned int s[500],int sockfd,const
        struct sockaddr *claddr,socklen_t len)
        {
        struct frame f;
        f.h.sequence_number=num;
        f.h.frame_type=2;
        f.h.size_of_frame=leng;
        f.h.compression_index=3;
        f.h.end_of_file=endof;
        f.h.check_sum=f.h.sequence_number^f.h.frame_type^f.h.size_of_frame^f.h.comp
        ression_index;
        for(int i=0;i<leng;i++)
        f.data[i]=s[i];
```

//   sends the created frame for buffering and transmitting to clients

```
        framebuffer(f,sockfd,claddr,len);
        sendto(sockfd,(const void *)&f,sizeof(f),0,claddr,len);
        return;
        }
```

FRAMEBUFFER.C

//   here a buffer is maintained and it is frequently refreshed

```
        vector<frame> v;
        void framebuffer(struct frame f1,int sockfd,const struct sockaddr
        *claddr,socklen_t len)
        {
        int i;
        vector<frame> :: iterator itr = v.begin();
```

//   Interpolation is done my rearranging the list with their sequence numbers

```
            if (f1.h.sequence_number%50==0) {
            v.clear();
            v.push_back(f1);
            }
            else{
//          inserts the frame in its correct interleaved position
            if (f1.h.sequence_number%50<5) v.push_back(f1);
            else{
            for (i=0;i<v.size();i++){
            if (f1.h.sequence_number%5<v[i].h.sequence_number%5) break;
            }
            itr = v.begin();
            itr=itr+i;
            v.insert(itr,f1);
            }
            }
//          sends the frames to the client in an interleaved manner
            if (v.size()%50==0 || f1.h.end_of_file==1){
            if (f1.h.end_of_file==1){
            v[v.size()-1].h.end_of_file=3;
            }
            for (i=0;i<v.size();i++){
            printf("%d %d\n",v[i].h.sequence_number,v[i].h.size_of_frame);
            sendto(sockfd,(const void *)&v[i],sizeof(f1),0,claddr,len);
            }
            printf("\n");
            }
            return;
            }
```

SCL.C

// sends the request to the server for an audio file

```
        #include "head.h"
        #include "dlist.c"
        int main(int argc,char **argv)
         {
        int sockfd,k=0;
        char ch,mesg[100],fln[30];
        socklen_t l;
        struct frame f;
        struct sockaddr_in sckaddr;
//      initializing network parameters
        bzero(&sckaddr,sizeof(sckaddr));
        sckaddr.sin_family=AF_INET;
        sckaddr.sin_port=htons(9877);
        inet_pton(AF_INET,argv[1],&sckaddr.sin_addr);
        sockfd=socket(AF_INET,SOCK_DGRAM,0);
        l=sizeof(sckaddr);
//      requesting and acknowledging the server
        printf("\nEnter y to continue n to stop : ");
        scanf("%c",&ch);
        if(ch=='y')
        {
        sendto(sockfd,"start\0",6,0,(const struct sockaddr *)&sckaddr,sizeof(sckaddr));
        while(k==0)
        k=recvfrom(sockfd,mesg,sizeof(mesg),0,(struct sockaddr *)&sckaddr,&l);
        printf("%s",mesg);
        scanf("%s",fln);
        sendto(sockfd,fln,sizeof(fln),0,(struct sockaddr *)&sckaddr,l);
        displst(sockfd,(struct sockaddr *)&sckaddr,sizeof(sckaddr));
        }
        else
        sendto(sockfd,"stop\0",5,0,(const struct sockaddr *)&sckaddr,sizeof(sckaddr));
        return 0;
}
```

DLIST.C

```
        #include "sequencer.c"
        struct  frame b;
        void displst(int sckid,struct sockaddr *servaddr,socklen_t len)
        {
        int n=0,i;
        unsigned char str[5000];
        char s[120];
        recvfrom(sckid,s,sizeof(s),0,servaddr,&len);
        if(strcmp(s,"No Such File\0")==0)
        {
        printf("%s\n",s);
        return;
        }
        else
        {
/*      receiving frames from the server in the order they were sent and sending
        them for reordering  */
        fp=fopen("out.dat","wb");
        while((n=recvfrom(sckid,(void *)&b,sizeof(b),0,servaddr,&len))!=0)
        {
        for(int i=0;i<b.h.size_of_frame;i++)
        str[i]=b.data[i];
        fwrite((void *)str,b.h.size_of_frame,1,fp);
        printf("%d\t",b.h.sequence_number);
        sequencer(b);
        if(b.h.end_of_file==3) break;
        }
        }
        fclose(fp);
        return;
```

SEQUENCER.C

// reordering of the received frames

```
unsigned char ch[4],ch1[2];
unsigned int c,d,e,i,s;
FILE *fp;
vector<frame> v1;
void sequencer(struct frame f)
{
vector<frame> :: iterator itr1 = v1.begin();
int k;
if (f.h.sequence_number<1) v1.push_back(f);
if (f.h.sequence_number%50==0 && f.h.sequence_number!=0){
v1.clear();v1.push_back(f);}
else{
for (k=0;k<v1.size();k++){
if (f.h.sequence_number<v1[k].h.sequence_number) break;
}
itr1 = itr1+k;
v1.insert(itr1,f);
}
for (k=0;k<v1.size();k++){
printf("%d \t",v1[i].h.sequence_number);
}
return;
}
```

PLAY.C

// plays the samples at a desired rate

```
unsigned char ch[4],ch1[2];
unsigned int c,d,e,i,s;
short int *data;
int sampling_rate,channels,number_of_samples,device,stereo,
format,bytes_per_second;
char fn[50];
FILE *fp,*fp1;
int main(){
printf("enter file name\n");
scanf("%s",fn);
fp=fopen(fn,"rb");
fread((void *)ch,sizeof(ch),1,fp);
printf("%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
fread((void *)&c,sizeof(c),1,fp);
printf("\n%u",c);
fread((void *)ch,sizeof(ch),1,fp);
printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
fread((void *)ch,sizeof(ch),1,fp);
printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
fread((void *)&c,sizeof(c),1,fp);
printf("\nsize %u",c);
fread((void *)&c,sizeof(c),1,fp);
e = c;
d = c & 0xffff;
printf("\nfmt %u",d);
```

```
        if (d!=1) {
        printf("\nthis is not PCM wav file\n");return 0;}
        d=c>>16;
        printf("\ncha %u",d);
        channels=d;
        fread((void *)&c,sizeof(c),1,fp);
        printf("\nNSPS %u",c);
        sampling_rate=c;
        fread((void *)&c,sizeof(c),1,fp);
        printf("\nNABPS %u",c);
        fread((void *)&c,sizeof(c),1,fp);
        e = c;
        d = c & 0xffff;
        printf("\nblk %u",d);
        d = e >>16;
        printf("\nbps %u",d);
        bytes_per_second = d/8;
        fread((void *)ch,sizeof(ch),1,fp);
        if (ch[0]!='d'){
        fread((void *)ch,sizeof(ch),1,fp);
        fread((void *)&c,sizeof(c),1,fp);
        fread((void *)ch1,sizeof(ch1),1,fp);
        fread((void *)ch,sizeof(ch),1,fp);
        printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
        }
        else printf("\n%c%c%c%c",ch[0],ch[1],ch[2],ch[3]);
        fread((void *)&c,sizeof(c),1,fp);
        printf("\nSize %u\n",c);
        number_of_samples=c/2;
//      opening sound device using necessary parameters
        device = open(dsp_device,O_WRONLY);
        if (device < 0){
        printf("Couldn't open sound device\n");}
        format = AFMT_S16_NE;
        if (ioctl(device,SNDCTL_DSP_SETFMT,&format) < 0){
        printf("Couldn't open sound device\n");}
        if (channels>1)
        stereo = 1;
        else stereo = 0;
        if (ioctl(device,SNDCTL_DSP_STEREO,&stereo) < 0){
        printf("Couldn't open sound device\n");}
        if (ioctl(device,SNDCTL_DSP_SPEED,&sampling_rate) < 0){
        printf("Couldn't open sound device\n");}
//      open sound device for writing data in binary mode
        fp1 = fdopen(device,"wb");
        printf("enter\n");
        fread((void *)data,bytes_per_second,number_of_samples,fp);
        fwrite((void *)data,bytes_per_second,number_of_samples,fp1);
        fflush(fp1);
        fclose(fp);
        fclose(fp1);
        return 0;
        }
```

APPENDIX –II: WORKING ENVIRONMENT

OPERATING SYSTEM: LINUX

PROGRAMMING LANGUAGE: C