OPTIMIZING REAL-TIME AUDIO SIGNALS OVER MOBILE NETWORKS

BY

JEFFREY P. MONKS

B.S., University of Illinois, 1996

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1999

Urbana, Illinois

ABSTRACT

This thesis explores various methods for optimizing real-time audio communications over wireless data networks. Currently, many methods for improving communications deal with optimizing the communications protocol. In this thesis the protocol is not changed, and any protocol is sufficient. The methods investigated in this work deal with optimizing communications through signal processing and data manipulation on top of a well-accepted protocol. The optimizations discussed deal with balancing the signal quality, bandwidth, and robustness.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Benjamin Wah, for his guidance and advice during my work on this thesis.

I would also like to acknowledge the aid of NSF grant MIP 96-32316.

TABLE OF CONTENTS

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

A variety of methods exists for optimizing the transmission of audio data, depending on the properties of the underlying network. In this work the mobile network is assumed to have bursty loss, variable and limited bandwidth, and nonstationary delay, attributes typical of most wireless networks. The methods available for handling these negative channel characteristics include equalization, interleaving, compression, rate-based control, and buffering. The purpose of this work is to combine different communication solutions to obtain the most desirable end-to-end system.

The underlying system is assumed to be an off-the-shelf wireless data network. The network is designed to send both real-time and non-real-time data. It is therefore advantageous to optimize the real-time data at a software level. This way the system will not be biased to handle any particular type of real-time and non-real-time data, with systems of inherent time constraints still achieving good performance. Many different data mobile phones and communication devices exhibit this paradigm. One example would include mobile Internet phones for business users at an airport. The airport could provide service to these users with a wireless LAN.

To properly demonstrate the effectiveness of different methods for overcoming negative channel effects and how these methods work together, a test bed was developed. In this test bed, voice signals were transmitted between two machines. The test bed consists of four Pentium Pro 200MHz computers, each with an onboard sound card, an Open Systems Software (OSS) sound driver, a microphone, a speaker, and four WaveLANs for networking the four computers (see Figure 1.1). The OSS driver sets up the sound card to retrieve 16-bit samples at 8 kHz from the microphone. The WaveLANs

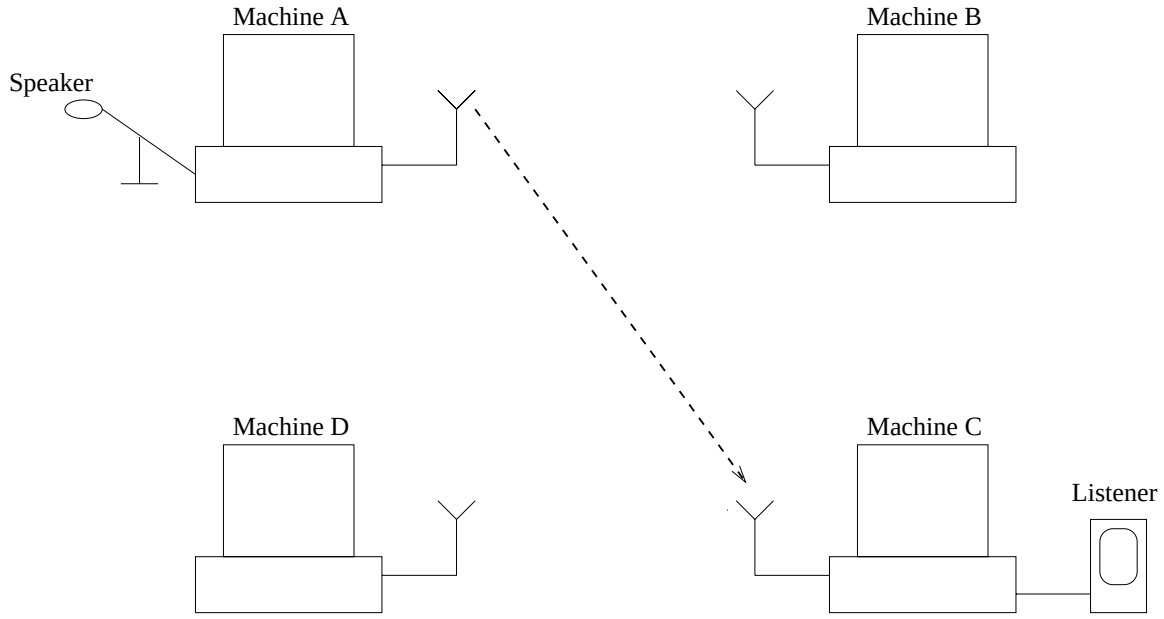

Figure 1.1 Test bed setup

were all set to the same channel to give the effect of mobiles contending for the same broadcast channel. This will also result in interference proportional to system workload. As the network load of each computer is increased, the probability of collisions increases.

To determine when collisions happen, it is important to note how the WaveLAN cards work. The WaveLAN implements the IEEE 802.11 standard [1] over a direct sequence spread spectrum (DSSS) communications channel. The IEEE 802.11 standard does the following: increases message reliability by incorporating carrier sense multiple access/collision avoidance (CSMA/CA) with acknowledgments, eliminates lost messages due to "hidden-node" collisions by using request-to-send (RTS) and clear-to-send (CTS) messages, increases performance reliability with automatic rate selection (ARS), and improves resilience to interference through message fragmentation (by breaking larger messages into smaller ones) [2].

The computers in the test bed were set up such that two of the machines (A and C in Figure 1.1) transmit and receive real-time voice signals. Machine A used the attached microphone to record input voice data, which was then packetized, coded, and sent over

the wireless LAN to machine B. The packets were then decoded, reassembled, and played on the speakers attached to machine B. All the machines in the test bed (A, B, C, and D) were used to run background processes in order to inject user interference or background noise into the system to test its robustness.

In this project, the user datagram protocol (UDP) was used to transmit packets between hosts. UDP was chosen because it is a well-established protocol and is more flexible than the transmission control protocol (TCP). Particularly, TCP implements retransmissions to guarantee reliable packet transmission, with no guarantees on when a packet will arrive at the destination. This also means that subsequent packets will wait an unknown amount of time for earlier packets to get through. For real-time data we must guarantee that packets arrive at the destination within a fixed period of time or move on to the next packet. UDP provides a best-effort service that is desirable for this scenario.

Chapter 2 discusses interleaving as a method for increasing robustness to network errors, packetization of data, and system latency issues. In Chapter 3, different voice compression techniques are investigated as a method of reducing bandwidth per voice signal and increasing the capacity of the system. Chapter 4 discusses different methods of reconstructing voice signals. Finally, Chapter 5 explores the effects of user interference on the performance of the prototype. Chapter 6 then makes conclusions, based on the results, on how to best choose the parameters for a real application.

CHAPTER 2

INTERLEAVING AND PACKETIZATION

In this chapter, interleaving is discussed as a method for overcoming bursty data loss. Interleaving is considered on two different levels, sample interleaving and packet interleaving. Also the packet structure is explored. To make interleaving robust, packet headers must contain information to notify the receiver when the all packets in an interleaved set have arrived. In this way, if packets are lost the receiver still knows when to process a data set.

2.1 Interleaving Methods

The first level of interleaving, which will be referred to as sample interleaving, distributes burst errors through a set of samples, making the errors less noticeable to the user. Short bursts are easier to conceal with interpolation and equalization because the distance between valid data is smaller. Chapter 4 will elaborate further on this subject. The number of samples separating sequential data after sample interleaving is called the *sample interleaving factor* (SIF). The samples recorded are grouped into sets, which are interleaved so each set contains SIF streams. Each stream is sent in a separate packet. Therefore, a single packet loss will result in every SIF sample being lost in the set.

The second level of interleaving will be referred to as packet interleaving since it interleaves packets instead of samples. Packet interleaving increases the ability to reconstruct data if there is a burst error affecting more than one interleaved stream. This method reduces the probability that more than one consecutive sample is lost. The reconstruction algorithms discussed in Chapter 4 perform better if consecutive samples are not corrupted. The number of packets separating sequentially interleaved streams is called

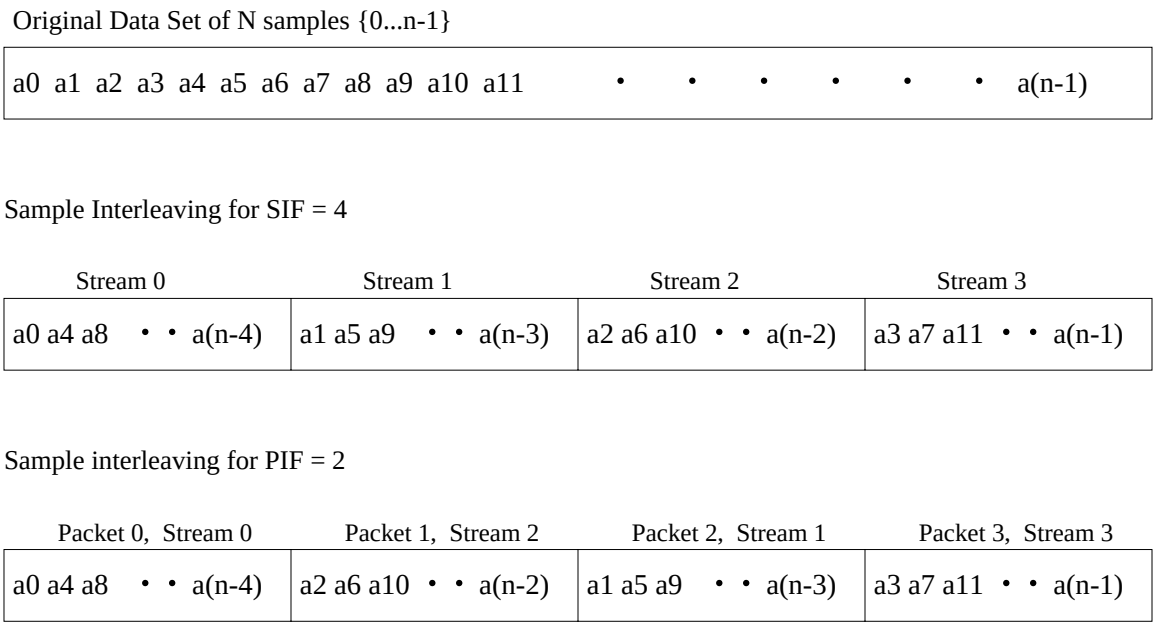

Figure 2.1 Sample and packet interleaving

the *packet interleaving factor* (PIF). Packets are sent from streams created by sample interleaving in a round-robin sense ordered by packet interleaving.

Figure 2.1 shows both sample and packet interleaving for SIF = 4 and PIF = 2. The first sequence represents a data set retrieved from the sound card, the second sequence is a result of sample interleaving, and the third set is a result of packet interleaving. Notice in the second stream consecutive samples in each stream are four samples apart as compared to those in the original sequence. Deinterleaving involves the same process but in the opposite direction (bottom to top).

2.2 Packet Structure

After packet interleaving, interleaved streams must be placed in separate packets. The packet structure is shown in Figure 2.2. The header size is four bytes and is kept as small as possible so as not to become a significant overhead. The components of the header are as follows: *Set_Num* determines the sampled set a packet is from, *Int_Num*

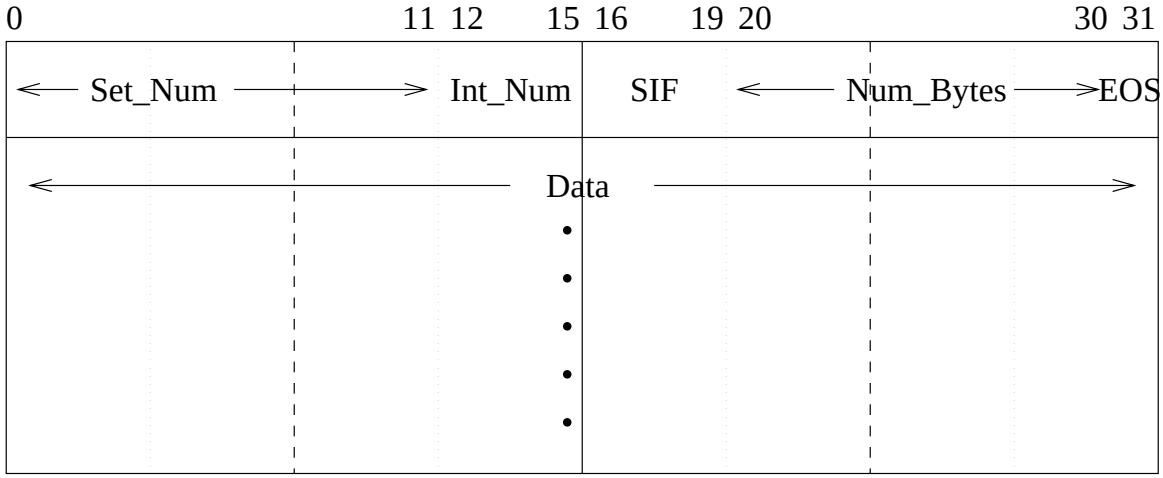

Figure 2.2 Packet structure

identifies each packets position within the data set, *SIF* specifies the interleaving factor so dynamic interleaving can later be employed, *Num_Bytes* specifies the total number of bytes the receiver can expect in the data set, and end of stream (*EOS*) notifies the receiver when the transmission is complete. *Set_Num* is necessary to identify when the next set has arrived. All packets in a set must be received (or declared lost) before deinterleaving can be performed. The elements of the packet header are used to help identify when all the packets in a set (not dropped by the network) have arrived. The current set is considered complete if a packet with the last *Int_Num* in the set arrives, the next *Set_Num* arrives, or the set times out (when the set is close to its deadline). In this method, if the last packet in the set is lost, a later set will push the data out. Likewise, the time-out will prevent an incomplete set from being wasted if subsequent data does not arrive for a long time.

In addition to correctly placing packets in the data set, a mask that identifies the lost data is created for use by the reconstruction procedure. The reconstruction routine uses the set mask to determine which samples require reconstruction.

2.3 System Latency

The transmitter and receiver are split into multiple processes to reduce their processing overheads. The transmitter consists of a sampling (recording) process and a process for interleaving, compressing, packetizing, and sending data. The receiver consists of a process for receiving, depacketizing, decompressing, deinterleaving, and reconstructing data and a process for playing data. In this manner the processing and networking delay is incorporated only into the initial latency.

The receiver also buffers data in the playing process to reduce the effects of network jitter. The buffer size must be increased as the background noise is increased. This topic will be more fully discussed in Chapter 5.

The timing of recording, processing, sending, receiving, buffering, and playing packets is shown in Figure 2.3. After the sample time of each set (labeled set sample time in Figure 2.3) the recorded data is sent to the sending process for processing and sending. The set is recorded in SIF number of separate segments, each taking read time (as labeled in Figure 2.3). After each set is sampled, the sending process is notified (shown as a dotted arrow between the reading and sending processes in Figure 2.3) to send a packet. This approach spreads packet transmissions throughout the time it takes to sample the data set, which reduces the probability that a burst error on the channel will corrupt more than one packet. For the first sampled set no signal is sent to the sending process, because no data is yet available to be sent. After the last sampled set is passed to the sending process, the recording process samples one more set (which will not be passed to the sending process) to maintain the proper timing in the sets sent.

The time between recording the voice signal at the client and playing it at the server is called the system latency. When SIF increases, the packet size decreases, while the overhead increases. As a result, it is desirable for robustness and efficiency reasons to have a large set, but for reasons of system latency it is desirable to have a smaller set. Therefore, there is a trade-off between latency, robustness, and efficiency, and the system

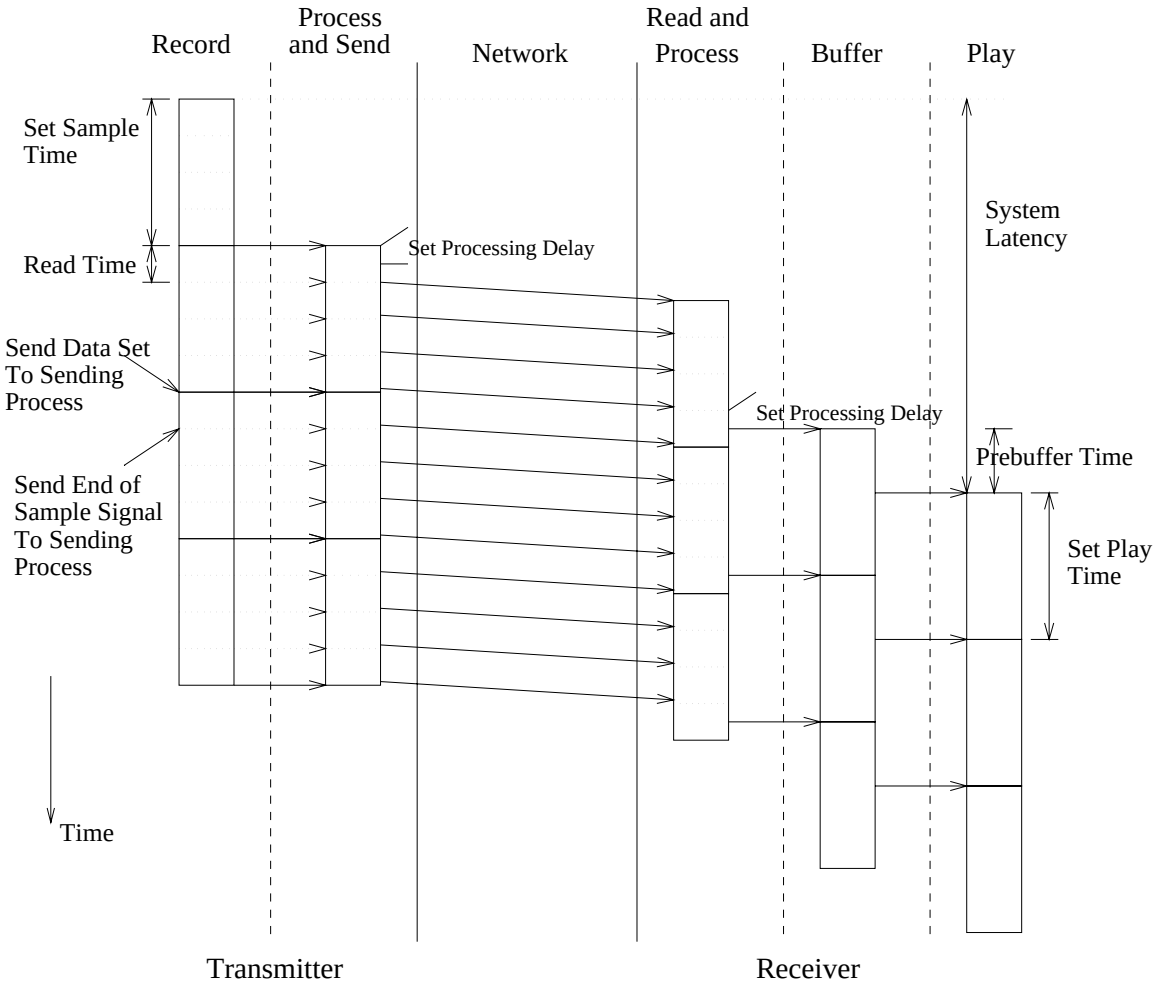

Figure 2.3 System actions versus time

must be designed such that all are in an acceptable range. This range will vary depending on the background interference. This will be evaluated further in Chapter 5.

To record the system latency, the transmitter and receiver are run on the same machine, and a second machine is used to echo the messages back to the transmitting machine. The transmitted message is sent through the WaveLAN network and returned through a fast Ethernet (100Base-T) network. This gives a very good estimate of the system latency, because the fast Ethernet has a small fraction of the WaveLAN network latency. Table 2.1 shows the relationship between latency versus set size. The reason for

Table 2.1 System latency

Set Size (bytes)	Measured System Latency (ms)	Twice Set Sample Time (ms)
80	12.7	9.77
160	23.3	19.5
320	43.5	39.0
640	84.5	78.1
1280	166	156

choosing the set size as multiples of 80 bytes is for it to be consistent with the requirements of the compression methods discussed in Chapter 3. The second column shows the latency measured with the method described above, and the third column shows the theoretical latency due to just twice the set sampling time (calculated from the sampling rate and set size). Here we see that the system latency is slightly more than twice the set sampling time. Note that it takes set sampling time to sample the data and set sampling time to send the data, since the packets are spread throughout the next set sampling period. The sampling time is, therefore, the most significant contributor to the delay, and the processing and network delays count for little of the total delay. The delay associated with a set size of 1280 bytes is slightly noticeable. If possible, we want to stay at 640 bytes or below (depending on the network traffic).

CHAPTER 3
COMPRESSION OF VOICE SIGNALS

In this chapter, compression will be discussed as a method for reducing the bandwidth used by a voice signal. The voice signals were recorded in 16-bit samples at 8 kHz in its natural format, pulse code modulation (PCM) [3], resulting in a bit rate of 128 kbit/s. This bandwidth is too high to be supported effectively in a mobile network.

In this work many compression methods were considered to reduce the voice bandwidth, including adaptive differential PCM (ADPCM) [4], linear predictive coding (LPC) [5], code excited linear prediction (CELP) [6], and low-delay CELP (LD-CELP) [7].

The LPC-10e and CELP coders result in a large compression ratio but at the cost of great delay; hence, they are not suitable for real-time implementations, as discussed by Spanias in [8].

The ADPCM (G.723) algorithm has very low delay and compresses voice signals to 16 kbit/s with an acceptable signal-to-noise ratio (SNR). Note that most literature [8] refers to G.721 as a 32 kbit/s ADPCM standard. For this work, because wireless networks were employed, it is desirable to use a lower bandwidth protocol, G.723. The ADPCM method implements scalar quantization to compress individual voice samples from 16 bits to 2 bits.

The other compression method implemented was LD-CELP, which also produces voices signals at 16 kbit/s with low delay at an acceptable SNR. LD-CELP compresses voice signals by combining a sequence (vector) of samples and by quantizing them together. LD-CELP incorporates the techniques used in CELP voice coding technologies except that it uses 5 sample-length vectors instead of 40. This reduces the delay of a typical CELP algorithm from 60 ms for 20 ms frames to approximately 5 ms [9], which is acceptable for real-time communications. The LD-CELP algorithm uses an 8-entry code-

Table 3.1 Encoding and decoding time

Time per 1kbyte (512 samples)		
Coder	Encoder (ms)	Decoder (ms)
ADPCM	5.97	4.93
LD-CELP	17.10	9.12

book for gain information and a 128-entry five-sample code book for shaping information in order to compress five samples (80 bits) to 10 bits (7 bits for shaping information and 3 bits for gain information). Notice that the input buffer size must then be chosen such that the total output bits are evenly divisible into bytes, since the coder outputs a 10-bit sample at a time. Therefore, the input buffer size must be a multiple of 80 bytes (16, 5 sample runs) resulting in an output buffer size of 10 bytes. This also explains the set sizes chosen in Table 2.1. See [8] for a more in-depth analysis of various speech coding techniques.

3.1 ADPCM versus LD-CELP

The ADPCM and LD-CELP encoder and decoder delays were recorded using the gprof Unix profiling tool on a 200-MHz Pentium Pro machine from the test bed described in Chapter 1. The computation delay shown in Table 3.1 demonstrates that the encoding time for LD-CELP takes more than three times that of ADPCM and more than twice as long for decoding. This is a large difference, but because the time for the data to be sampled and sent is about 125 ms for 1 kbyte of data (twice the sampling time), both coders increase the total system latency (described in Chapter 2) by only a small amount.

The SNR for the system was calculated according to

$$SNR = 10 \log_{10} \left(\frac{\sum_{i=0}^{N-1} S_i^2}{\sum_{i=0}^{N-1} (S_i - \hat{S}_i)^2} \right), \tag{3.1}$$

where S_i refers to sample number i in the input sequence (file) consisting of N total samples. Note the SNR calculation is the average SNR for the input sequence in decibel

Table 3.2 Encode and decode time

File	Size (kbyte)	File Duration (s)	Description
F1	500	31.25	Adult male voice
F2	1000	62.5	Multiple voices speaking
F3	5000	31.25	Adult female voice
F4	1000	62.5	Music (piano and singing)
F5	113	7.06	Prerecorded Sound file

(dB) format. For SNR calculations, several different input files were used for testing the compression methods (and later reconstruction methods). Table 3.2 gives a description of five different input files referred to as F1-F5. Files F1-F4 were recoded in PCM format with the microphone attached to one of the computers in the test bed, and F5 is a WAV file downloaded from the Internet.

To keep the bandwidth small and to incorporate robustness, voice compression and interleaving are combined. In order not to defeat the robustness of interleaving, each interleaved stream must be compressed separately. This way, even if a packet is lost the other packets in the data set can still be decompressed.

Figure 3.1 shows the SNR for different SIF values (1, 2, 4, and 8). In this experiment no packets were dropped. Nevertheless, we expect SNR to decrease as SIF increases because the distance between inputs samples to the encoder and decoder increases with SIF. Therefore, the variance increases with SIF, and the input data cannot be represented as accurately. From another view, as SIF is increased, consecutive samples become less similar. As a result, the error in representing the change between samples (in the case of ADPCM) or multiple samples together (for LD-CELP) increases.

Figure 3.1 shows that LD-CELP has a higher SNR for every input file tested than does ADPCM. Hence, LD-CELP is the preferred voice compression technique. For file F4, ADPCM performs particularly poor. This file is a music recording with a piano; when the keys on the piano were struck sharply, the difference between consecutive samples

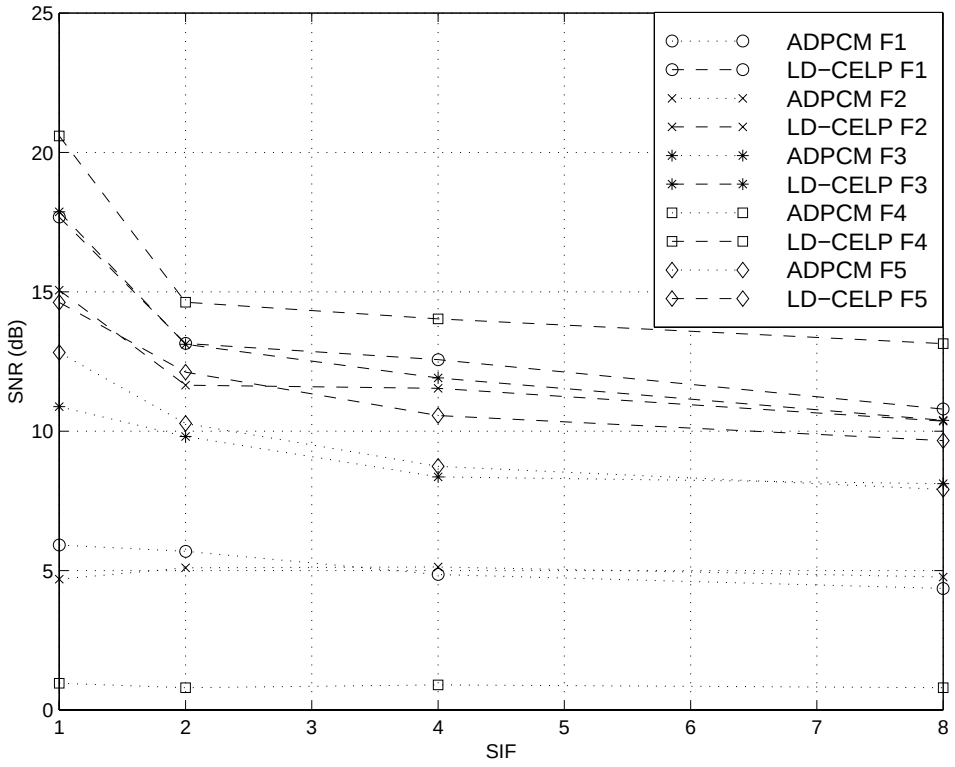

Figure 3.1 SNR versus SIF

was too great to be recorded in 2 bits. However, LD-CELP performs very well for this file because it encodes the general shape and gain of five consecutive samples, which provides more leverage in balancing the information in the coded bits. For ADPCM, there are only a fixed number of bits available for encoding each sample. This trend is also evident in file F2 where the performance of LD-CELP is also much better than that of ADPCM, because this voice sample has many different voices from distinct speakers tuning in and out. The performance of ADPCM will be low for any input sequence with large amplitude deviations between consecutive samples. File F5 yields an ADPCM performance closer to that of the LD-CELP than any of the other samples. This was a sound file clip from a movie downloaded from the Internet. File F5 is much smoother and of higher quality because it was professionally recorded. This allows ADPCM to perform much better. The F1-F4 samples recorded with the PC microphone have the breath from the speaker

added as noise when they speak into the microphone. They are of lower quality than the professionally sampled input sequence, F5. The professional recording process would be much more expensive than a simple PC microphone and, therefore, not realistic to combine with a typical real-time network communication structure.

Increasing SIF from 1 to 2 has a larger effect on LD-CELP than ADPCM. However, LD-CELP still performs better at all interleaving factors. This degradation in performance does not make a large difference in perception (when SNR is still above 10 dB for LD-CELP), and the benefits in robustness outweigh the performance degradation of increasing SIF above 1.

The above results give many motivations for choosing LD-CELP over ADPCM. The greatest advantage is that it provides greater voice quality than does ADPCM. The SNR was not shown relative to the PIF value because this factor adds no degradation to the performance of a coder. Packet interleaving takes place after coding and therefore, has no effect on the performance of the coder. The effects of packet loss on the ADPCM and LD-CELP compression methods will be explored in Chapters 4 and 5.

CHAPTER 4
RECONSTRUCTION

When dealing with lossy networks, data samples (packets) are periodically lost or corrupted. However, it is quite often possible to restore lost data samples to near their originally sampled values. This process is called *reconstruction* and involves either using neighboring sample values to interpolate the original values or estimating the effects of the channel and inverting those effects. Reconstruction requires some of the data in the set be present, so interleaving must be incorporated into the transmitted data.

4.1 Interpolation

Two algorithms that assist in correcting for lost data samples are interpolation and equalization. Interpolation uses received samples surrounding lost samples to better estimate the true values of lost samples. The values of lost samples are originally zero because UDP does not deliver corrupted packets (they are simply dropped if the checksum is not valid, see [10]). UDP does not allow packets with data corrupted by the network to be delivered (with probability $1 - \frac{1}{(2^{16}-1)}$); they are simple dropped. Therefore, we can assume all packets received by UDP contain the actually sampled values. After deinterleaving, a bit mask was then used to identify the lost sample entries in the data set (lost packets map to lost samples in the data set after deinterleaving). A "0" bit corresponded to a correctly received sample, and a "1" bit corresponded to a lost sample. This way, the interpolator can estimate the lost bits by a straight line between two consecutive "0" bits. Figure 4.1 shows how interpolation was implemented. The "O" values represent received samples and the "X" values correspond to estimated samples. The dotted lines between received ("O") values show the interpolation process for finding the estimated ("X") values from the lost data. When edge samples (corrupted bits at

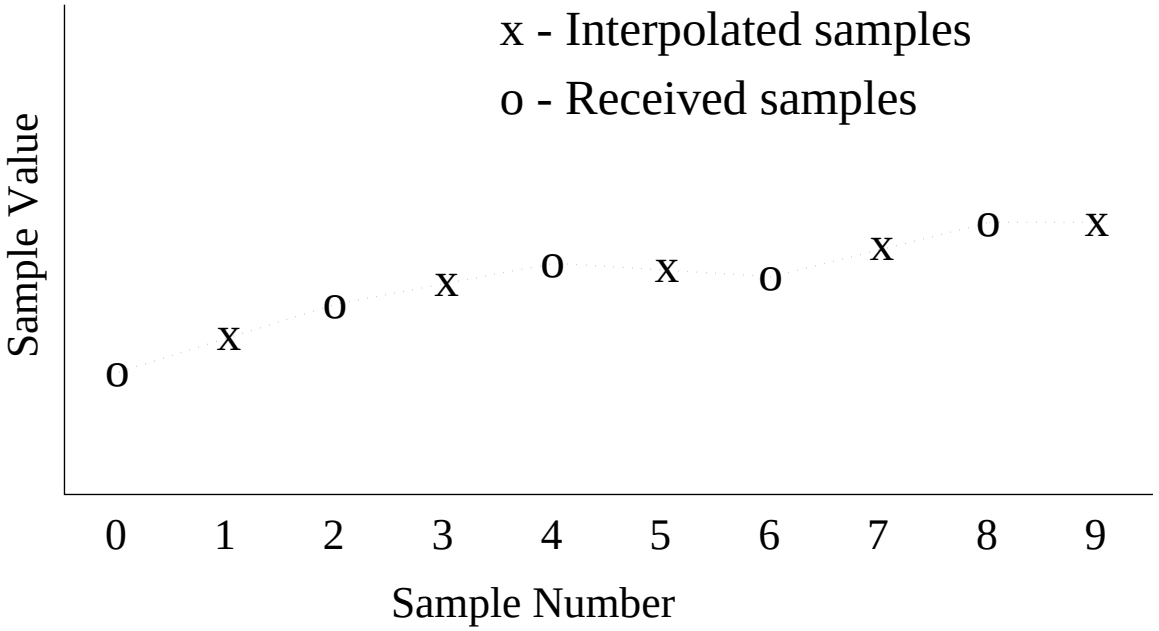

Figure 4.1 Interpolation method

the end of a set) were estimated, only one reference was available, so their value was set to the last received sample in the set.

4.2 Equalization

Equalization was performed on the data samples after interpolation. The interpolation smoothed the data and gave the equalizer a good starting point so that it would converge closer to the actual sample values. There are many types of equalizers used in different voice communication applications. For this project, linear mean squared (LMS) adaptive equalizers (filters) were implemented because they are flexible and have low complexity. In particular, a gradient descent LMS equalizer [11] was chosen due to its fast convergence and simplicity to implement.

A typical LMS algorithm implementation relies on past samples to estimate present ones. However, in the algorithm implemented in this project, both past and future samples are used to estimate the present sample (placing the estimated sample in the

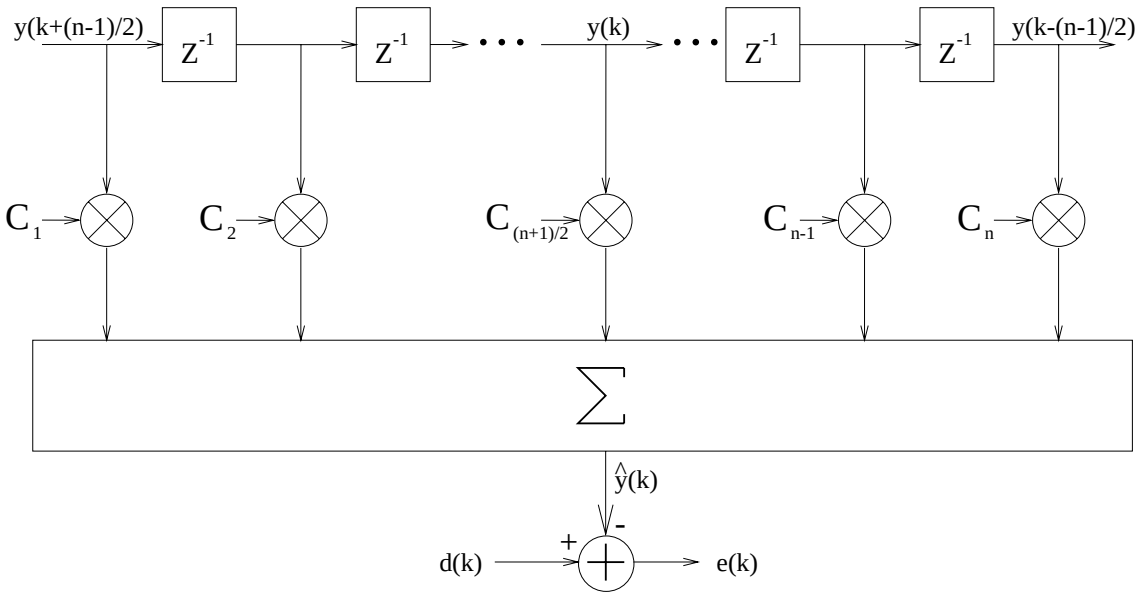

Figure 4.2 An LMS equalizer

middle of the equalizer window) as shown in Figure 4.2. Also, only the corrupted samples are equalized (using the lost sample bit mask). Using both past and future samples has an advantage in that the estimated sample is only half as far from the farthest samples in the equalizer. The farthest any sample is from the estimate is $(n-1)/2$ instead of $n-1$, where n is the equalizer size (number of samples used for estimating the current sample). As a result, we maintain the same filter order (length), but reduce the time variance between samples over the equalizer window. When the filter length increases the equalizer is able to more accurately represent higher-order effects. However, the time variance of the equalizer window also increases with filter length. In addition, there exists a diminishing rate of return as the number of higher-order effects modeled increase. Therefore, an optimal filter length exists that is the best balance between including the greatest filter length such that increasing it will cause increased time variance and degraded performance. In a real system, the optimal filter length will depend on the medium that carries the transmitted data.

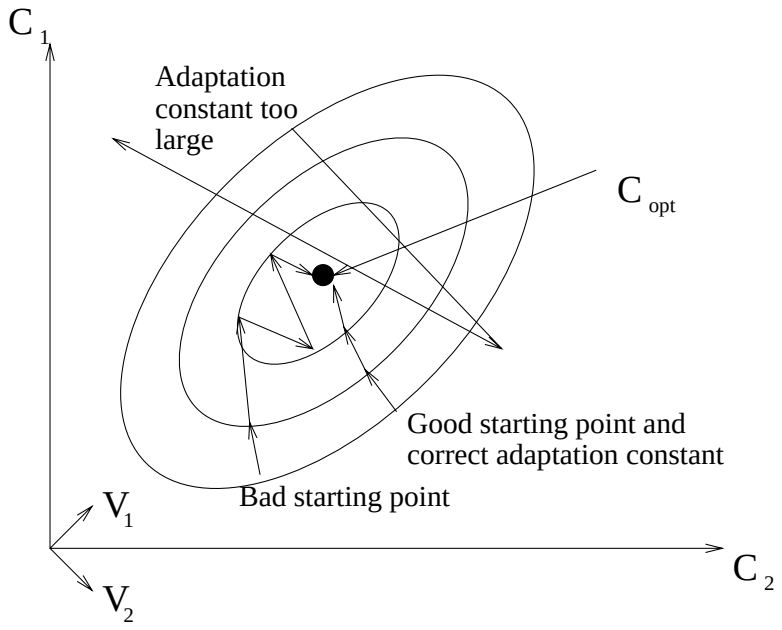

Figure 4.3 Gradient descent

The initial conditions (starting values of the multiplier coefficients, C_1 - C_n) also proved to be critical to the success of the equalizer. The reason for this is shown in Figure 4.3, where ovals represent ranges for which the eigenvalues are of equal value, V_1 and V_2 are the eigenvectors, C_1 and C_2 are the multiplier coefficients, and C_{opt} is the optimum choice for C_1 and C_2. Starting in an off-centered region of the oval may cause the algorithm to never converge or to take much longer to converge.

A good starting point was found to be the $C_{\frac{n+1}{2}-1} = 0.5, C_{\frac{n+1}{2}} = 0.01, C_{\frac{n+1}{2}+1} = 0.5$, and all other multiplier coefficients are zero. Note that this is almost the effect of the interpolator.

Care must also be taken in deciding the size of the gradient coefficient. The coefficient size is represented by the arrow length in Figure 4.3. If the coefficient is too large, then the algorithm will diverge. However, if it is too small, the algorithm will take a long time to converge. A gradient coefficient around 10^{-10} was generally chosen.

The operation of the equalizer was divided into training and nontraining periods. The training periods were the times when all the data in a set were correctly received. During

the training period the equalizer was given a sequence with the equivalent of one packet being dropped (every SIF sample interpolated from its neighboring samples). The error was then calculated as the difference between the estimated output from the equalizer and the actual data received. This error calculation is then used to adjust the multiplier coefficients as follows:

$$C(k+1) = C(k) - \frac{\beta}{2}\nabla_c[e_c^2(k)] \tag{4.1}$$

$$= C(k) - \beta e_c(k)y(k) \tag{4.2}$$

or for each multiplier coefficient:

$$C_j(k+1) = C_j(k) + \beta e(k) y\Big(k - \frac{n-1}{2} - (i-1)\Big), \tag{4.3}$$

where $C_j(k+1)$ is the jth multiplier coefficient for equalizer input $y(k+1)$, β is the gradient coefficient, $e(k)$ is the error in the estimate $\hat{y}(k)$, and $y(k - \frac{n-1}{2} - (j-1))$ is the input to multiplier j. During the training period, the data set has the correct data, so the estimated data is only used to adjust the coefficients.

When data is corrupted in the set, the equalizer is in a nontraining period. During this period, the coefficients are fixed (not adapted), and the output of the equalizer is used as the data into the set for the lost samples.

The filter was tested by dropping every other sample in every other set (that is dropping 25% at the transmitting host) so that training periods and use of the equalizer were alternated between sets. To keep the filter length balanced (equal number of samples on each side of the current input, $y(k)$), only odd filter lengths were implemented. Different filter lengths were tested for each of the input files (with no compression), and the results are shown in Figure 4.4. A filter length of 1 means that interpolation was implemented, but no equalization was involved. Figure 4.4 shows that SNR improves for increasing filter lengths, up to a filter length of 7; after which only some of the input data sequences (files) improve while others cause decreases in SNR. For the data tested, using a filter length of 7 would result in an improvement of at least .5 dB over just using interpolation.

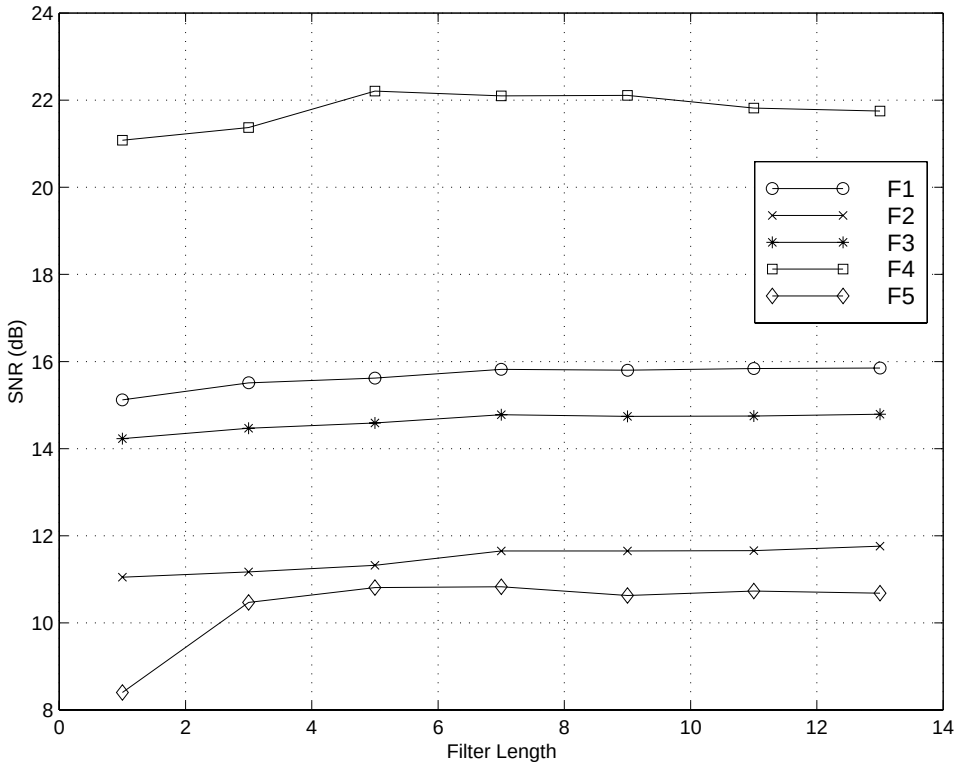

Figure 4.4 Filter length versus SNR with no compression and set size = 640 bytes

A 0.5 dB improvement, while not exceptional, is a 12% increase in SNR, which is enough to justify the cost of adding the equalizer.

4.3 Reconstruction With Compression

Now consider using compression with the same packet loss scenario as described in Section 4.2. One important factor to note about the voice compression methods is that both carry a state in order to adapt their encoder and decoder and to obtain the best possible SNR. However, when packets are lost, the state of the encoder and decoder become unsynchronized, and the SNR drops significantly. For this reason, the encoder and decoder state of each interleaved stream must be kept separate.

There are two possible techniques for separating the state of the encoder and decoder. The first technique is to track down all the variables used to hold the state of the coder

Table 4.1 SNR for different compression techniques, 25% loss, SIF = 2, for file F1

Method	Coder	SNR for $flen = 1$	SNR for $flen = 7$
Include all outputs	ADPCM	4.60	4.50
Include all outputs	LD-CELP	5.45	4.92
Drop unsynchronized	ADPCM	5.58	5.60
Drop unsynchronized	LD-CELP	10.12	10.19

and to change each variable into an array indexed by the stream number which is passed into the coder by the calling process. The second technique involves creating multiple processes that call the same coding procedure. When different processes call the same procedure the variables in the procedure reside in a separate memory space for each process. The first technique was used for the ADPCM coder since only one structure was necessary to track the state. However, the LD-CELP coder's state was much more complicated and was retained by many different variables in many different procedures and files. This made the first technique too complicated and tedious to implement, and as a result, the second technique was implemented instead. The second technique, in addition to being simpler to implement for LD-CELP, is more expensive in memory and processing time. Using multiple processes separated not only the state variables but also all variables and memory space associated with the process. Furthermore, additional interprocess communications was required to synchronize the encoding and decoding the streams in different coding processes. The additional overhead of adding these processes is observed in Chapter 5 when the system latency is measured in terms of background workload.

If the encoder and decoder states are separated for different streams and the packet-loss scenario used for the equalizer is included, the resulting SNR for data file F1 (when the output from all decoders is included) is shown in Table 4.1. The implications of choosing the filter length, labeled *flen* in the table, will be discussed later. The resulting SNR is still quite low for both compression techniques. The reason for this is that, even

though under this loss scenario, the packets in one stream are all received, packets are periodically dropped in the second stream, causing the corresponding decoder to be out of synchronization with the encoder and its output degraded. In such cases, where the contribution of an unsynchronized decoder only degrades the quality of the signal, it is beneficial to drop the packets in the corresponding stream.

When output packets from the unsynchronized decoder are dropped, Table 4.1 shows a modest improvement for ADPCM and a significant improvement for LD-CELP. The reason for the large increase in quality for LD-CELP is that this coder has many dependencies among the vector sets; therefore, the quality of one stream can greatly degrade, causing the overall quality of the system to drop. We can actually do better than this if we determine the time for which a decoder remains unsynchronized (time for which including the output of a particular decoder does not benefit the quality of the output signal). This time is represented as the number of sets to pass after a packet was lost on a particular stream. Once a decoder is unsynchronized, the time before synchronization will be referred to as the coder synchronization time (CST) and is used in our stream-drop algorithm. Therefore, when a packet loss occurs in a stream, the corresponding decoder is considered unsynchronized until CST consecutive packets are passed through the decoder (without contributing to the output). The stream-drop algorithm drops a stream when the stream had a packet loss less then CST sets before the current set. In addition, the best stream (the one with the greatest number of sets since its last packet loss) was always included because data from an unsynchronized decoder is better than no data in a set. The value of CST is chosen experimentally and depends on the size of the data set and network loss rate. The CST is chosen by measuring the SNR for different choices of CST and choosing the one which corresponds to the maximum SNR. While the best choice will vary slightly for different voice samples, the results provide a good approximation of CST.

The SNR was explored for various values of CST with different types of errors. To test the stream-drop algorithm one packet every 8 sets (each set with two interleaved streams) was dropped (a packet loss of 6.25%). Figure 4.5 shows the results for ADPCM

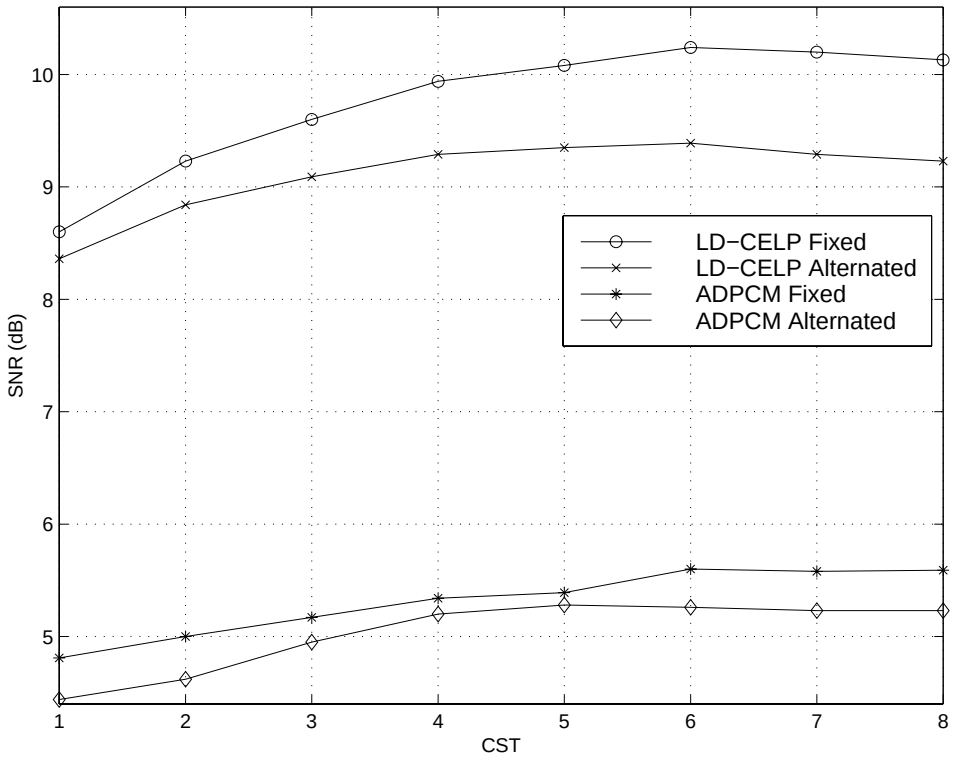

Figure 4.5 CST versus SNR, SIF = 2, set size = 640 bytes, packet loss every 8 sets

and LD-CELP. The figure shows one fixed stream corrupted (packets always lost in one interleaved stream) and alternating streams corrupted (packets dropped from alternating streams). Figure 4.5 shows that ADPCM has the highest SNR when CST is between 5 and 6, whereas the SNR for LD-CELP is the greatest when CST is about 5.

Next, consider an error scheme that corrupts packets every 16 sets. The new results for ADPCM and LD-CELP are displayed in Figure 4.6. The best CST is now between 5 and 8 for ADPCM and about 5 for LD-CELP. The figure demonstrates the sensitivity of the LD-CELP decoder to packet loss. Note that CST = 1 is equivalent to always including the contribution from the output of a decoder, and CST = 16 results in every output packet from an unsynchronized decoder being dropped (since an input stream will always have a packet dropped every 16 data sets). The best CST value is the point

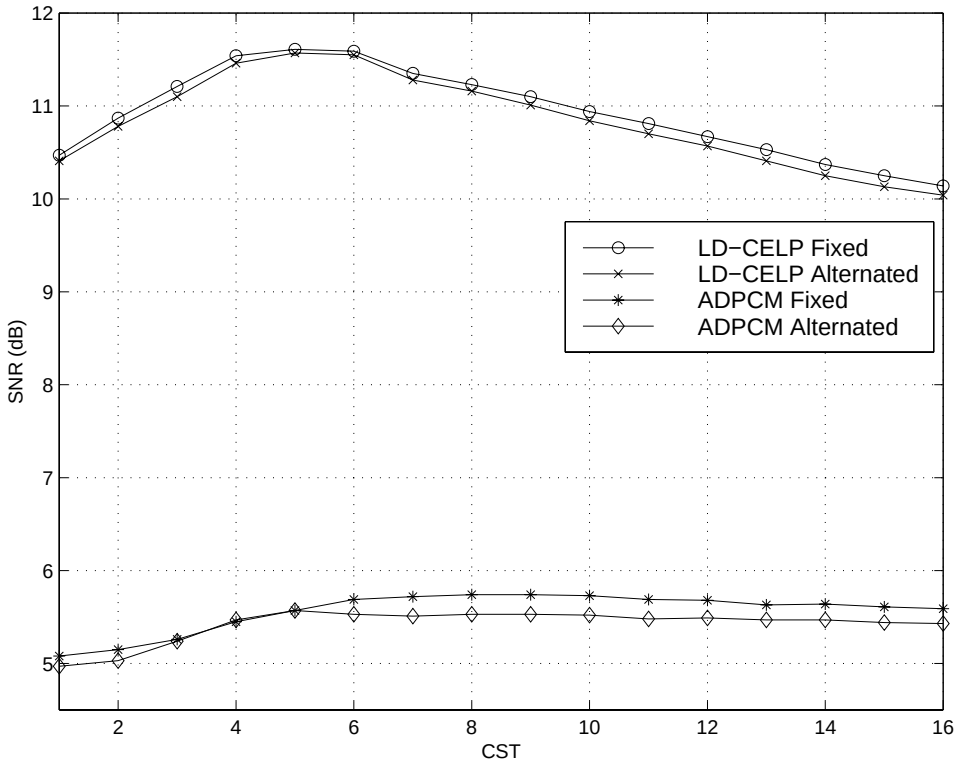

Figure 4.6 CST versus SNR for ADPCM, SIF = 2, set size = 640, packet loss every 16 sets

after which dropping more packets from a decoder is no longer beneficial, and prior to this point, the unsynchronized decoder may still degrade the quality of the output.

Figure 4.7 shows the overall system layout. Each stream has a separate encoder and decoder. At the receiver, all streams go into a stream-drop module that drops packets from unsynchronized decoders.

In Table 4.1 a filter length (labeled *flen* in the table) of 7 was compared to that of 1 for the different methods. A *flen* of 7 was chosen since it was the best choice from Section 4.2, and an *flen* of 1 specifies no equalization just interpolation. The equalizer with optimal parameters ($flen = 7$) was compared to the case when no equalizer was used to determine the utility of the equalizer when implemented in conjunction with compression. The equalizer showed no benefits after adding compression. In the first method (including all

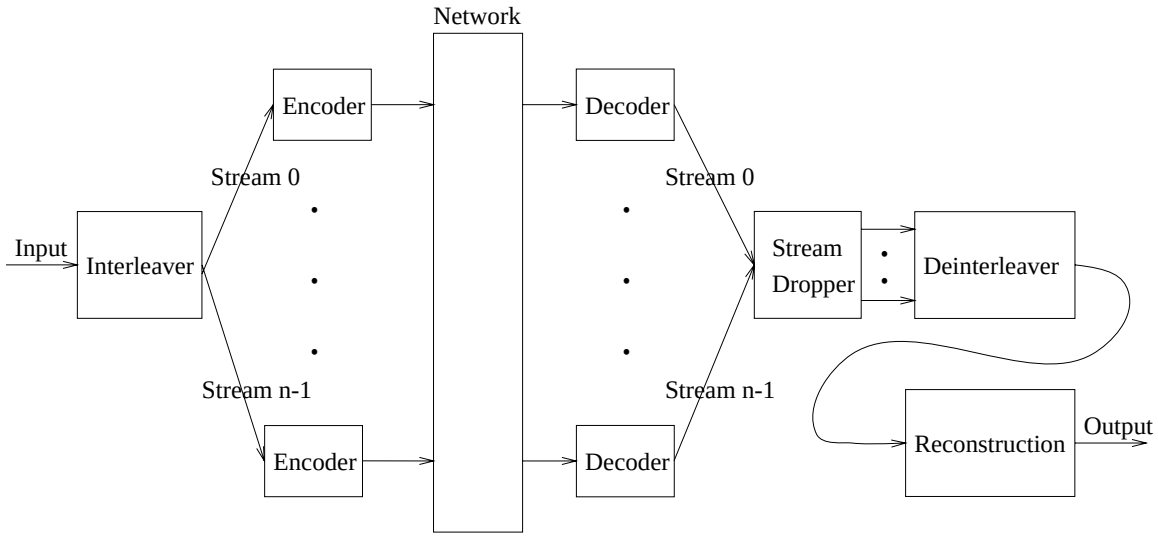

Figure 4.7 System design

streams in the output) the training period was corrupted by the unsynchronized decoder that mislead the equalizer to poor choices of multiplier coefficients; whereas in the second method (where the packets were dropped when the decoder was unsynchronized) only the first data set was used for training; as a result, the multiplier coefficient would be accurate for only the first few sets. Overall, the unsynchronized decoder prevented the equalizer from improving the quality of the output signal. Therefore, when compression is present in a lossy network, equalization has no appreciable benefit, and only interpolation should be used for reconstruction.

CHAPTER 5
USER INTERFERENCE

This chapter examines how interference from other users affects the quality of a particular voice signal transmitted over the network. The user interference or background workload will also allow us to test the robustness of the transmitted signals in the system. As the number of users increases, the available system bandwidth and quality degrade. However, the quality does not degrade linearly. The WaveLAN manages bandwidth among multiple users by sending RTS and CTS packets and by performing ARS (as described in the Chapter 1). In addition, large packets are fragmented into smaller ones in order to reduce the number of bytes lost. This last point may not add much value because an entire UDP packet is considered lost even if a small fraction is lost. The network bandwidth used by a single user will vary depending on the coding techniques implemented. For this reason, all background workload will be specified in total bandwidth demanded by all other users on the network.

5.1 Injecting Interference into Transmissions

The interference was generated by each machine sending interfering packets (of fixed size) to other machines in Figure 5.1, where each dashed line represents an interfering path, and the dark dashed line shows the path of the voice data. In this fashion, the interference was distributed among all the machines, with more load on the machines not processing the voice signal in order to give the appearance that the interference is coming from many machines on the local network. Each machine in the system sends a number of interfering packets per time unit proportional to the desired background workload.

Through experimentation it was determined that the interference scales more evenly and consistently if multiple processes on multiple machines are used to inject the inter-

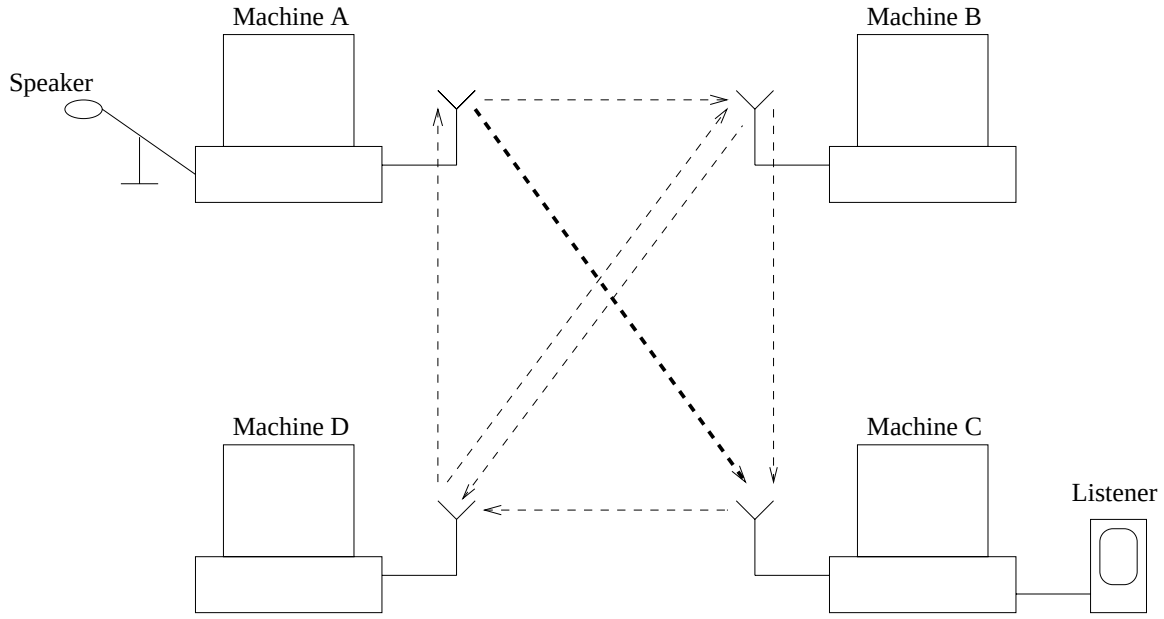

Figure 5.1 Testbed with voice signal and background workload

ference. This is due to the WaveLAN protocol that schedules packets in such a way that processes with large bandwidth do not stop processes with small bandwidth from arriving at their destination. To calculate the total background workload, the number of injected packets by each interfering process is summed to get a total background workload.

The F1 file was used for all tests related to user interference because it is desirable to observe various parameters with respect to different amounts of background workload. Each data point was the average of 10 runs of the F1 file (a total of 312.5 s). The background workload will be referred to in kilobits per second of injected traffic from other users. In the next few sections, the word *sequence* will be used to refer to the input voice data.

The first relationship observed is the limit of the network to provide a given data rate, with many processes on many nodes transmitting and collecting data at the same time. The relationship between the input data rate and the output data rate (all the transmitting and receiving processes combined) is shown in Figure 5.2. The limit of the WaveLAN network's data rate is approximately 1000 kbit/s. The difference between the

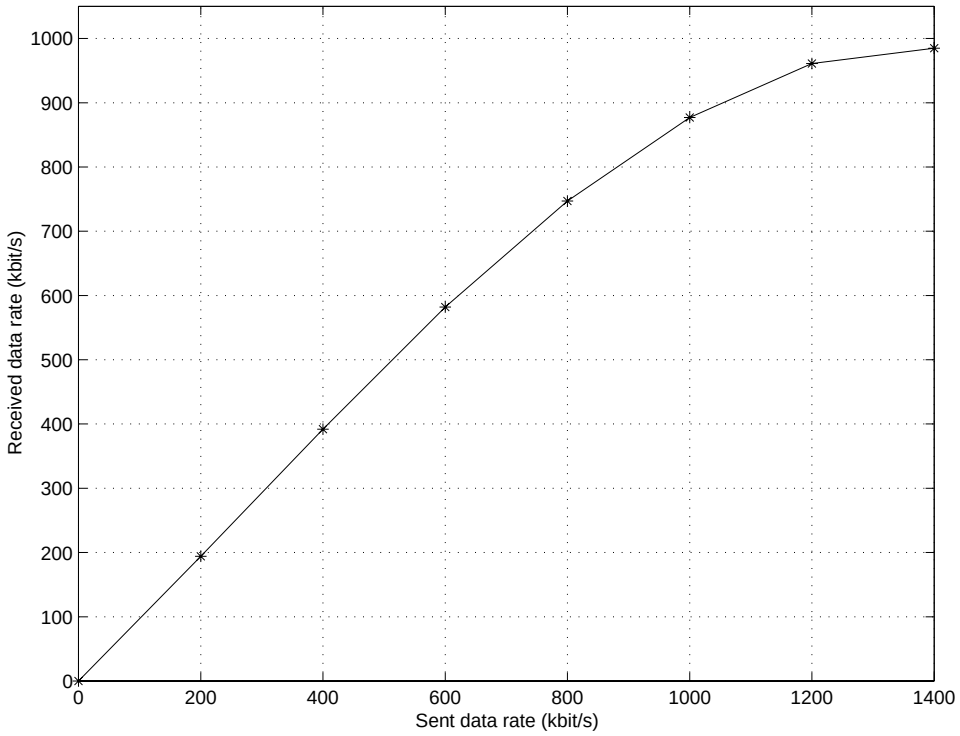

Figure 5.2 Received data rate for different sending rates

sent-data rate and the received-data rate is accounted for by the collisions of packets and congestion in the network. The graph is almost linear up to approximately 600 kbit/s, showing that the WaveLAN card conceals most of the channel interference (with the techniques described in Chapter 1) for moderate levels of injected traffic, after which the received rate degrades more significantly. In general, the figure shows the bandwidth does not scale (after a certain amount of system load) with the number of senders.

The curve in Figure 5.2 models the system's throughput response for different amounts of injected traffic. This data was gathered with four machines for both sending and receiving. If a different number of machines were present on the local area network, the system may behave somewhat differently. However, because all subsequent tests used the same topological configuration, these results give an intuitive understanding of the system's behavior under various workloads.

5.2 Performance of Compression Methods under Interference

In the prototype developed for this work, packets were dropped by the network when collisions occurred. The number of packets dropped by the network increased with the system workload. However, the latency also increased with the workload, causing packets to periodically miss their time frame. When this occurred the packets were dropped by the receiver. As a result, packets could be dropped by both the network (when collisions occurred) and the receiver (when they missed their deadline).

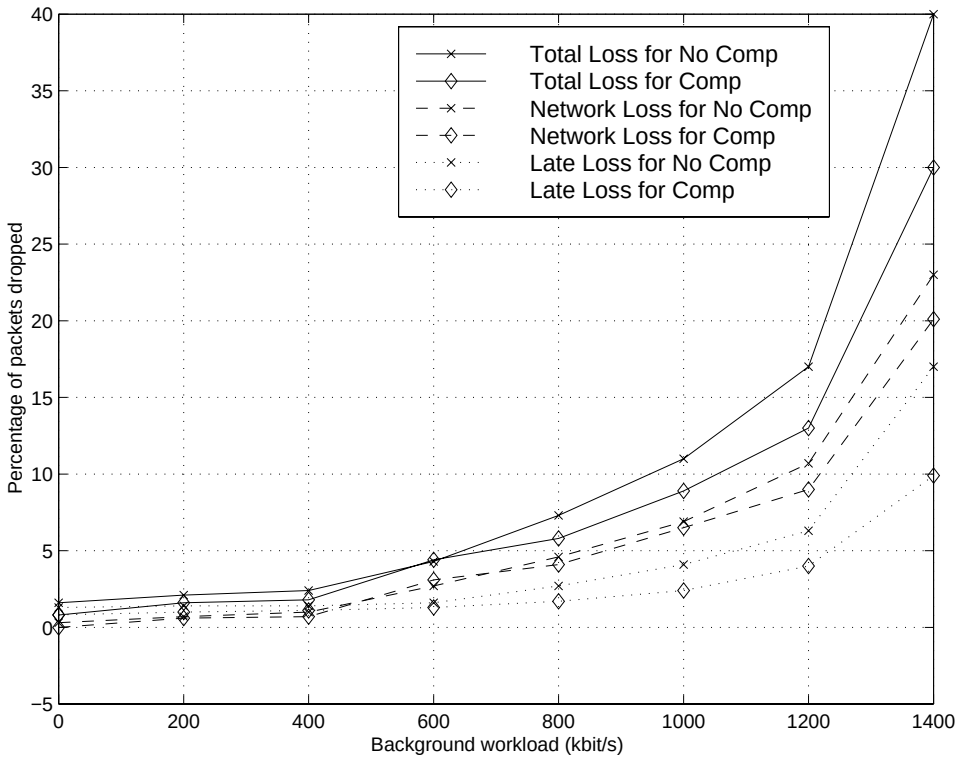

Figure 5.3 Percentage of packets dropped by network for compressed and uncompressed streams with respect to background workload for SIF = 2, set size = 640 bytes, and CST = 5

Figure 5.3 shows the percentage of packets dropped by the network and receiver for various levels of background workload. The uncompressed streams have a packet size of 160 bytes, and the compressed streams have packets of 20 bytes (due to the 8 to

1 compression). Note, that only one type of compressed stream is shown, instead of distinguishing between ADPCM and LD-CELP, because both of these methods result in 20-byte packets, and we are only concerned with the number of packets lost. The dashed lines in Figure 5.3 show the percentage of packets lost by the network; the dotted lines show the percentage of late packets dropped by the receiver because the packets missed their deadline; and solid lines represent the total number of packets dropped (by network and receiver). The number of packets dropped by the network increases with background workload because the probability of packets colliding increases as the number of packets injected per time unit increases. Furthermore, the number of late packets increases with workload because the larger the number of packets injected into the network, the longer it takes to schedule each packet.

The receiver contains a prebuffer that stores a specified number of sets to reduce the effects of network jitter on the output signal. For these results a one set buffer is chosen because it reduces the effects of jitter without increasing the latency above reasonable limits. Packets are declared late if they arrived later than

$$prev\_set\_time + (pbuf\_size + 1) * set\_sample\_time, \tag{5.1}$$

where $prev\_set\_time$ is the amount of time the previous set was received, $pbuf\_size$ is the number of sets prebuffered, and set_sample_time is the time at which it takes to sample a set and, therefore, the time between set transmissions.

In the remainder of this section, the SNR and system latency will be used as performance metrics for comparing the ADPCM and LD-CELP methods and the case when no compression is added. When the background workload is increased, the number of packets dropped increases. As the number of dropped packets increases, SNR decreases because the reconstruction process has less information, leading to lower quality signal. Similarly, latency increases because each set takes a greater amount of time before it can be declared complete (and passed to the playing process). The receiver waits to pass packets to the playing process until the last packet is received, a packet from a later set is received, or the packets in the set are close to missing their time schedule.

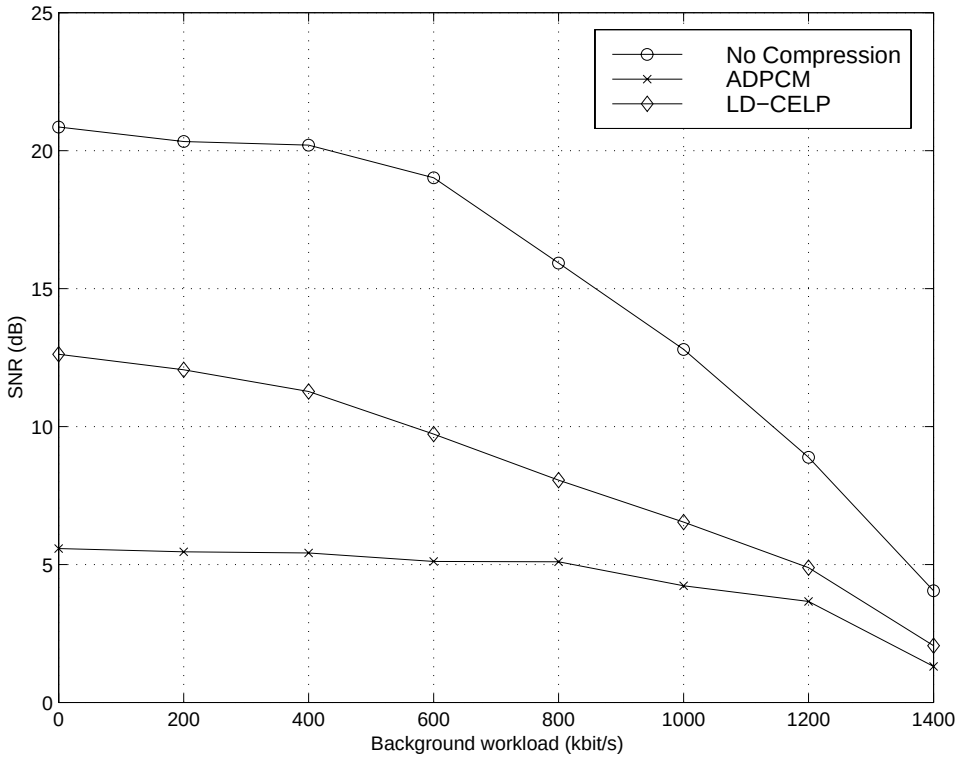

Figure 5.4 SNR for different types of compression with respect to background workload for SIF = 2, set size = 640 bytes, and CST = 5

In Figure 5.4 the SNR is shown when using no compression, ADPCM, and LD-CELP. The set size was 640 bytes, which is the maximum size at which the system delay found in Chapter 2 was within tolerable limits. Figure 5.4 shows that the SNR for sequences without compression remains significantly better than compressed data sequences until the background workload is extremely large (greater than 1200 kbit/s). Note that even under no background workload the system has some packet loss due to thermal noise present in the median. If this were not the case the uncompressed stream would be infinite when there is no background noise. For low background workload, LD-CELP performs well up to about 600 kbit/s, after which it degrades quite a bit below 10 dB, causing a noticeable drop in voice quality. The ADPCM coder starts off low and changes little until the background workload becomes larger than 800 kbit/s. With a set size of 640

bytes and an SIF of 2, the packet size will be 320 bytes and after compression, 40 bytes. For large workload levels (greater than 600 kbit/s in Figure 5.4) the uncompressed stream degrades much quicker than the compressed stream. This is due to a greater probability of losing larger packets. These results are also confirmed in Figure 5.3, where we see significantly greater packet losses for larger packets. For this experiment, the 160-byte uncompressed stream has a larger percentage of packets lost as compared to the 20-byte compressed stream for large background workloads.

When the number of users (background workload) on the network increases, the time to schedule (transmit) a packet also increases. As a result, the system latency (delay) will also vary with background workload. The delay was measured for different levels of interference and compared with respect to the two compression methods and no compression. Figure 5.5 reveals that LD-CELP degrades the system latency the most, followed by ADPCM. The figure shows the resulting latency with and without a prebuffer. The prebuffer was chosen to be (as above) a one-set buffer. Because the average time between sets is 40 ms, this is the time difference between methods using a prebuffer and those without. All methods (those with and without compression) result in similar system delays, until the background workload becomes very large, and the larger packets of the uncompressed sequence take longer to send. The reason for this result is that it takes longer to schedule larger packets than smaller ones under large system load.

5.3 Robustness of LD-CELP

LD-CELP has been shown to provide large SNR with only small penalties in latency. Therefore, it alone will be further analyzed for different interleaving factors.

Figure 5.6 shows the packet loss percentages with respect to background workload for SIF =2, 4, and 8. For these values of SIF and a set size of 640 bytes, the packet sizes (after compression) will be, respectively, 40, 20, and 10 bytes. The number of packets sent per set is equal to the SIF value so the total number of packets sent also increases directly with SIF. When SIF = 2 the network loss is larger than the other choices, whereas for SIF = 8 there is a greater percentage of late packets with large background workload.

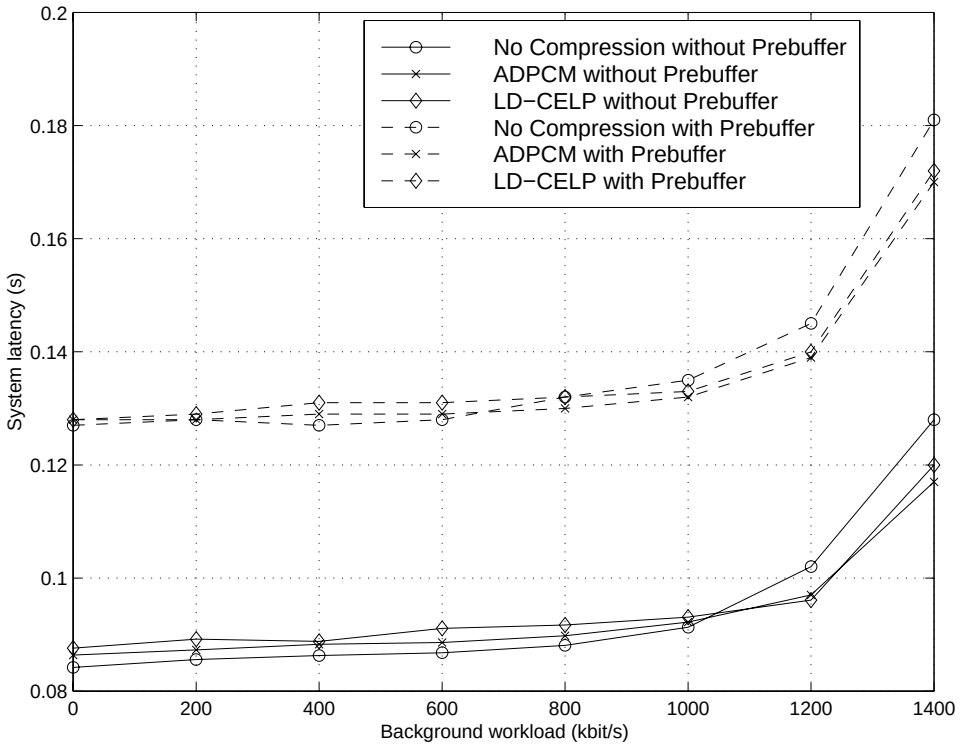

Figure 5.5 System latency for different types of compression with respect to background workload, SIF = 2, set size = 640 bytes, CST = 5

Larger packets (those associated with SIF = 2) have a higher percentage of loss because there is a greater chance of collision. However, smaller packets in greater numbers (true for SIF = 8) need more coding processes and are more likely to be late because a greater percentage of them arrive. The general trend is similar for all SIF values with each type of loss. This is particularly true for the total loss because the network and late packet loss have opposite loss statistics with respect to SIF. Any differences in the packet loss that corresponds to the SIF value are insignificant.

Figure 5.7 shows that the smaller SIF values perform better with low background workload (this also supports the results in Figure 3.1), though at larger workload levels the larger SIF values become the preferred choice. Particularly, the sequence with SIF = 2 performs the best between 0 and about 650 kbit/s of background workload, and

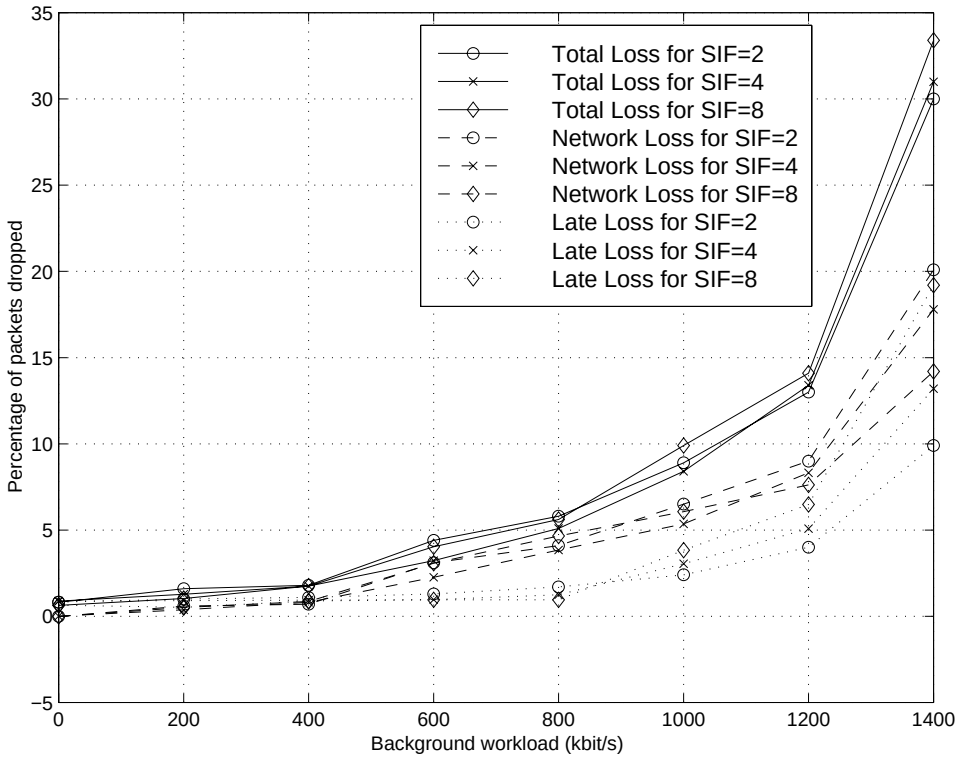

Figure 5.6 Packet loss for different sample interleaving levels with respect to background workload, set size = 640 bytes, CST = 5

the sequence with SIF = 4 performs the best between 650 and 1200 kbit/s. Then at approximately 1200 kbit/s both sequences with SIF values of 2 and 4 become equally insignificant. The sequence with SIF = 8 never becomes the best sequence because the system reaches its maximum load before the corruption rate is large enough to make up for its lower starting SNR (shown in Figure 3.1). The larger the SIF value, the lower the probability that all the packets in a set are dropped by the network because there are more packets per set. However, this does not mean that packets will arrive before the deadline. This is demonstrated in Figure 5.6, which shows that when more packets are sent, more will miss their deadline, and more sets will be dropped. In Figure 5.8, the interleaved sequence with SIF = 2 has the greatest percentage of sets lost, and the one with SIF = 4 has the best. This also explains why the SIF = 4 sequence had the greatest

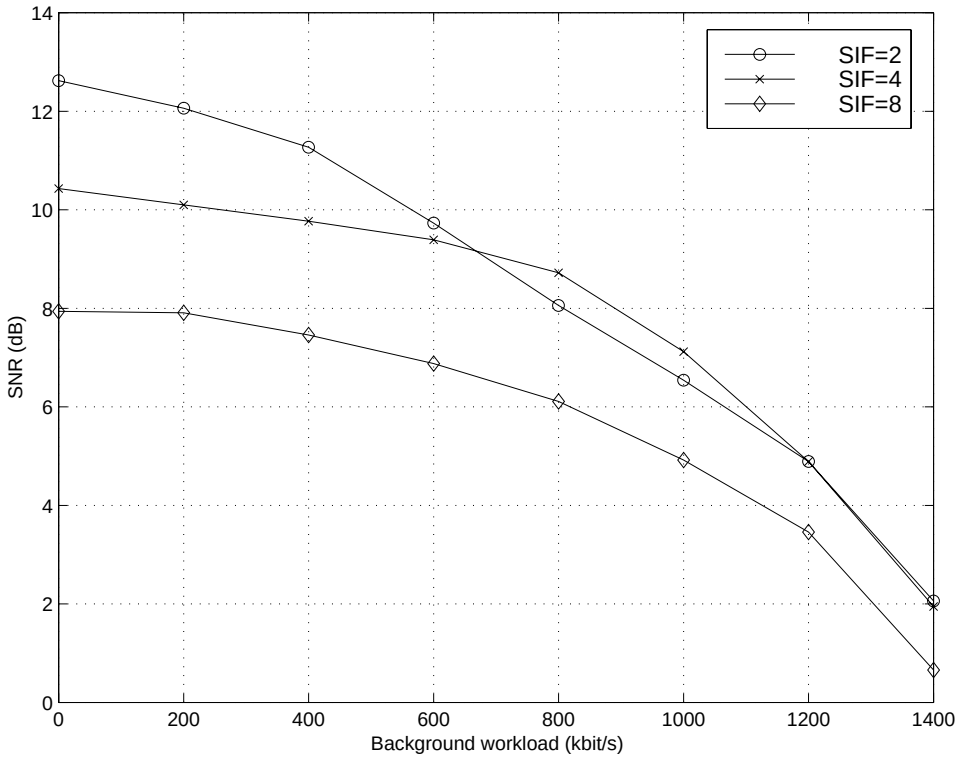

Figure 5.7 SNR for different sample interleaving levels with respect to background workload, set size = 640 bytes, CST = 5

SNR for high background workload. The increased loss of sets eventually degrades the performance of the sequence with the SIF values of 2 until it eventually loses any initial advantages it had over sequences with larger SIF values. When a set is lost there can be no reconstruction, leading to much lower SNR.

The system latency or delay for larger SIF values is greater due to the multiple encoders and decoders used for each stream. Therefore, using more streams requires more encoders and decoders. Also more packets are sent for greater SIF values. As mentioned earlier, the set size is fixed at 640 bytes; therefore, as SIF is increased, the packet size proportionally decreases, and the number of packets proportionally increases per set. This means that the receiver will have to wait for more packets to fill a set before moving on. If the network is heavily congested, the probability of a packet arriving late

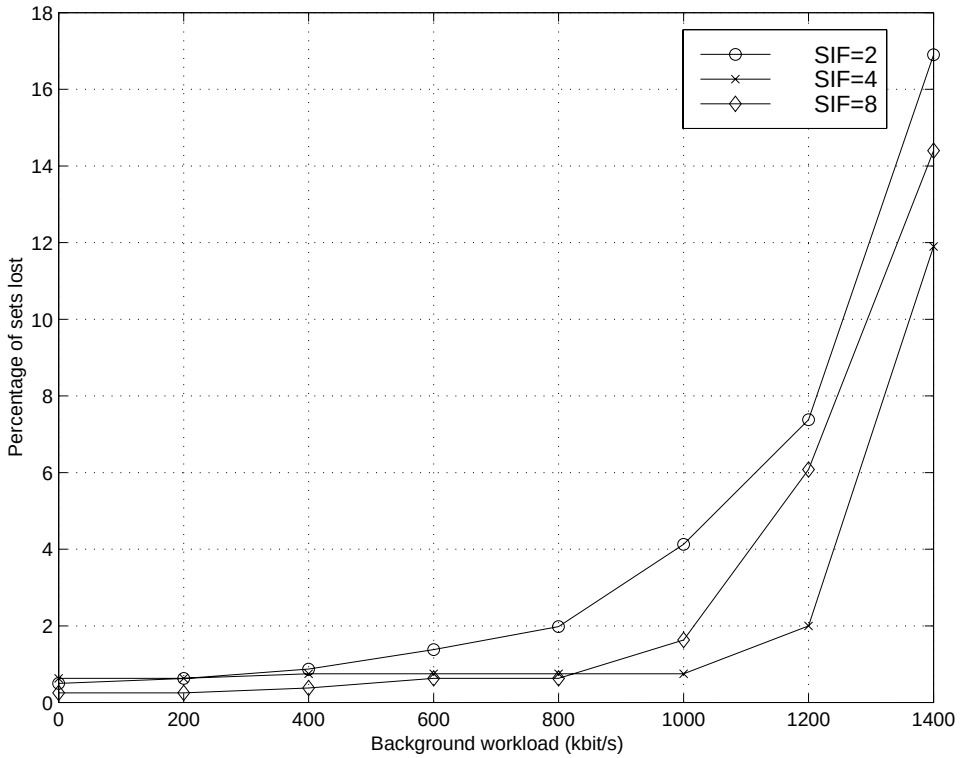

Figure 5.8 Percentage of sets lost for different sample interleaving levels with respect to background workload, set size = 640 bytes, CST = 5

increases as the number of packets increases (despite their smaller size). Figure 5.9 shows that the difference in delay between various SIF values becomes significant after the system is loaded above 1000 kbit/s. The results without and with a prebuffer are again shown. The difference in latency is also about 40 ms because the prebuffer delays one set before initially playing any voice data.

The jitter is a measure of the amount of variation of system latency and is defined by the following equation:

$$Jitter = 2 * (max\_delay - avg\_delay), \tag{5.2}$$

where max_delay is the maximum system latency recorded for all the transmitted sets, and avg_delay is the average system latency recorded for all the transmitted sets. In Figure 5.10, the jitter increases as the background workload increases. As the system

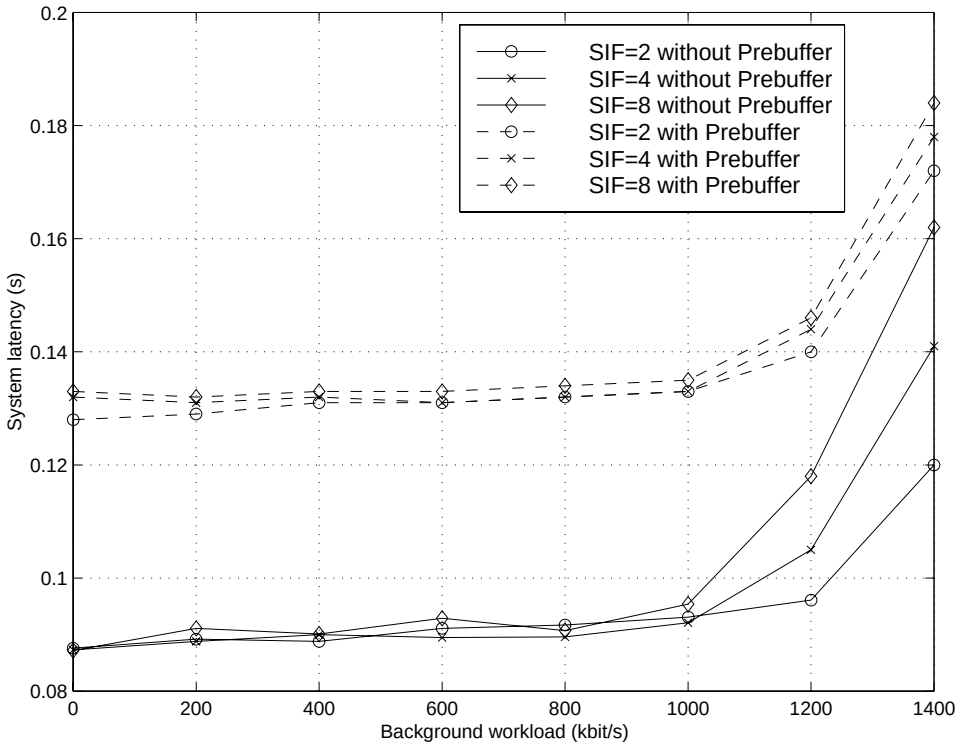

Figure 5.9 System latency for different sample interleaving levels with respect to background workload, set size = 640 bytes, CST = 5

becomes heavily loaded, the packet arrival rates varies greatly. This figure illustrates that the jitter of the sequences with larger SIF values grow faster than that with smaller SIF values as the system load increases. The true purpose of a prebuffer is to reduce the jitter seen at the output. In Figure 5.10 we see the jitter is reduced after the prebuffer is added. There is still some jitter though because when packets are lost, the prebuffer is temporarily emptied. When the next set arrives the prebuffering procedure attempts to delay playing the set until another set arrives so the buffer may be refilled. However, if the time before another packet arrives is too great, then the prebuffer procedure will play the packet anyway and attempt to refill on the next couple of sets. Therefore, some of the network jitter is seen at the output for a short period of time. The time that the prebuffer is depleted will increase with background noise, causing the jitter to increase.

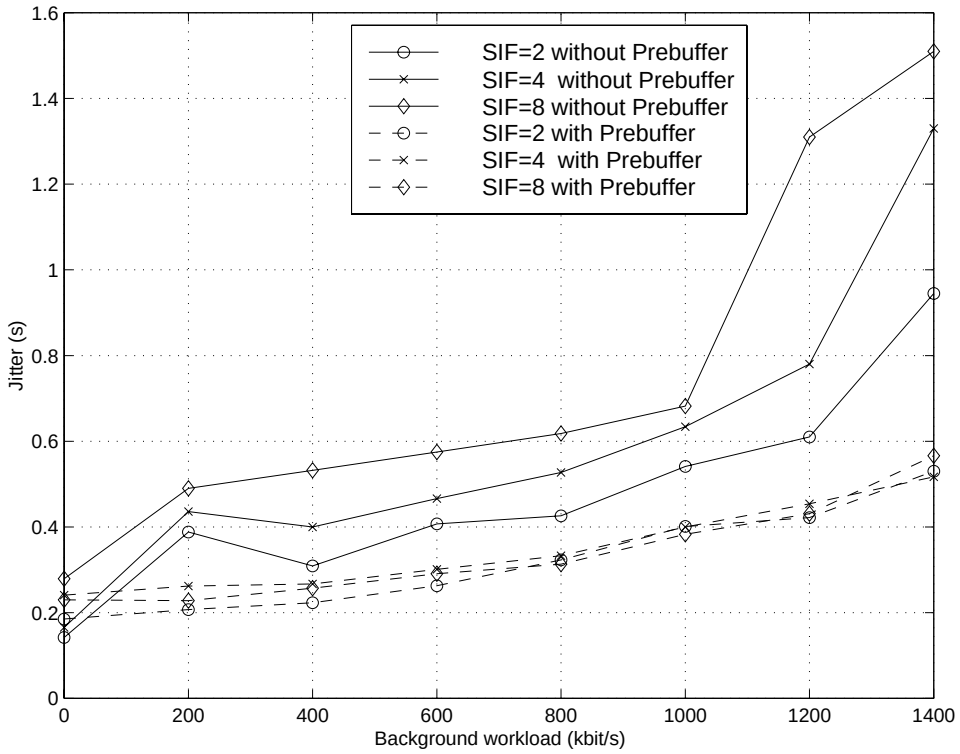

Figure 5.10 System jitter for different sample interleaving levels with respect to background workload, set size = 640 bytes, CST = 5

The PIF value is another factor for overcoming interference by spreading errors throughout the set and by allowing the reconstruction method to provide reconstructed samples to achieve a higher SNR. However, Figure 5.11 indicates that packet interleaving made an insignificant difference in improving SNR. The packet interleaving factor results in an increase in SNR by balancing the error bits through the set. For example we prefer to send a sequence such as "XOXOXOXO" instead of "XXOOXXOO" (where "X" represents a lost sample and "O" a received sample) to the reconstruction process because a higher SNR will result. However, for packet interleaving to make an improvement, a burst error must be long enough to corrupt more than one continuous packet. The probability of this occurring is small because packet transmissions are already spaced evenly throughout the time it takes to sample the next set (as described in Chapter 2). Also the

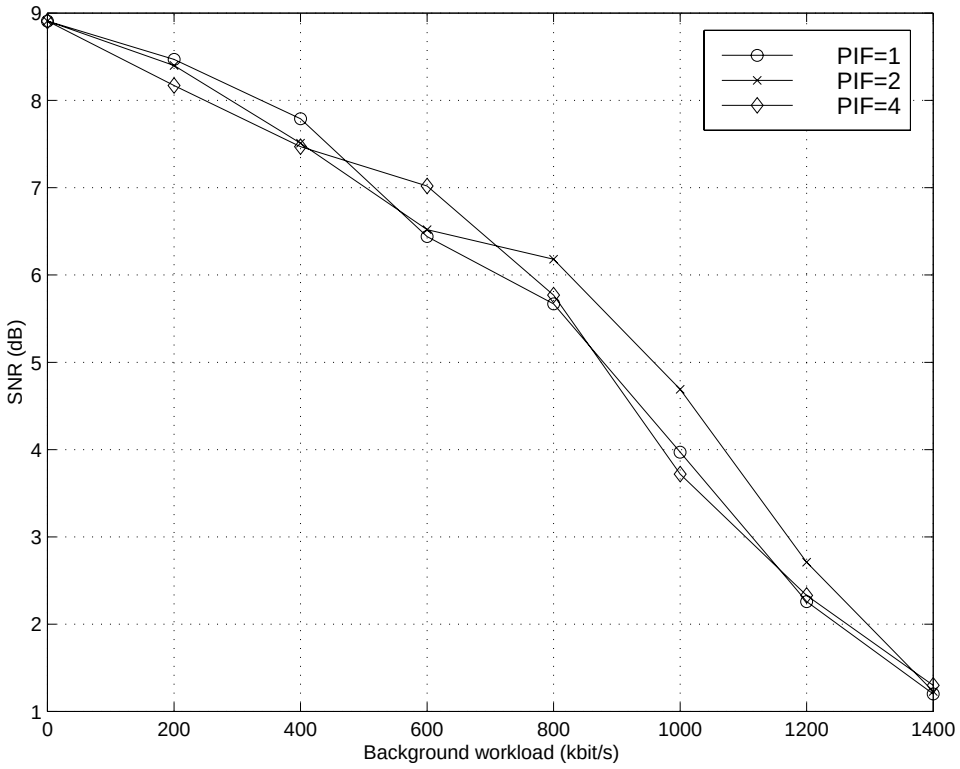

Figure 5.11 SNR for different packet interleaving levels with respect to background workload, SIF = 8, set size = 640 bytes, CST = 5

increase in SNR is small in the average case. Therefore, the insignificant change in SNR for different PIF values shown in Figure 5.11 can be explained by both the low probability of a burst error corrupting more than one packet and the small improvements in SNR. As a result, packet interleaving is not necessary and provides little if any benefits.

These experiment required using a set size no smaller than 640 bytes because for SIF = 8 the packet size would be 80 bytes before compression and 10 bytes after. As described in Chapter 3 the LD-CELP encoder cannot compress a buffer smaller than 80 bytes and still stay on the byte boundary. However, Figure 5.7 shows us that a choice of 2 or 4 for SIF is better than 8 because SNR is greater for all levels of background noise. This would allow the set size to be 320 bytes or smaller. Here we can conclude from Figure 5.10 that a smaller set size and larger prebuffer is favorable to further smooth the

arrival of packets at the output. The smaller set size would decrease the latency (which is already close to the maximum desired), providing more time for prebuffering the data.

CHAPTER 6

SUMMARY

This thesis investigates various methods for optimizing real-time audio signals over a mobile network. A test bed was first developed to classify the performance of the system under different parameters. Interleaving together with different reconstruction methods were explored for improving SNR over networks with various levels of packet loss. Different compression techniques were analyzed for their ability to reduce the bandwidth demands for transmitting audio signals. The independent and coupled effects of applying these methods were observed with SNR and latency measurements. Finally, compression combined with interleaving and reconstruction were examined under different levels of background traffic in order to study the robustness of our testbed.

The results indicate that LD-CELP is the best choice of audio compression. This compression technique results in the greatest SNR with only small penalties in delay and jitter. LD-CELP also proves robust under various amounts of background traffic. The best SIF value was 2 or 4, depending on the system load. These values allow reconstruction to keep the SNR at respectable levels even under high background traffic. The equalizer implemented in Chapter 4 results in insignificant improvements in SNR after incorporating compression. As a result, only interpolation should be incorporated into the reconstruction process.

Future work includes further investigation into altering the LD-CELP coder in order to provide higher SNR for large background traffic scenarios. In addition, exploring other reconstruction methods is another area that may provide promise.

REFERENCES

[1] IEEE Standard 802.11, *Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Networks–Specific Requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1997.

[2] Lucent Technologies, Inc., "WaveLAN products," Aug. 1998, http://www.wavelan .com/products.

[3] N. S. Jayant, "Coding speech at low bit rates," *IEEE Spectrum*, vol. 23, pp. 58–63, Aug. 1986.

[4] American National Standards Institute, Inc., *American National Standard for Telecommunications: Digital Processing of Voice-band Signals–Algorithms for 5-, 4-, 3-, and 2-bit/Sample Embedded Adaptive Differential Pulse Code Modulation (ADPCM)/Secretariat, Exchange Carriers Standards Association*, Jan. 1991.

[5] Federal Standard 1015, *Telecommunications: Analog to Digital Conversion of Radio Voice By 2400 Bit/Second Linear Predictive Coding*, Nov. 1984.

[6] J. Campbell, T. Tremain, and V. Welch, "Proposed federal standard 1016 4800 bps voice coder: CELP," *Speech Technology*, vol. 5, pp. 58–64, Feb. 1990.

[7] CCITT Draft Recommendation G.728, *Coding of Speech at 16 kbits/s Using Low-Delay Code Excited Linear Prediction (LD-CELP)*, 1992.

[8] A. S. Spanias, "Speech coding: A tutorial review," *Proceedings of the IEEE*, vol. 82, pp. 1441–1582, Oct. 1994.

[9] J. Chen, R. Cox, Y. Lin, N. Jayant, and M. Melchner, "A low-delay CELP coder for the CCITT 16 kb/s speech coding standard," *IEEE Transactions on Selected Areas in Communications, Special Issue on Speech and Image Coding*, vol. 10, pp. 830–849, June 1992.

[10] D. E. Double, *Internetworking With TCP/IP: Vol. 1: Principles, Protocols, and Architecture*. Eglewood Cliffs, NJ: Pretice Hall, 1995.

[11] M. L. Honig, *Adaptive Filters*. Boston, MA: Kluwer, 1984.