ABSTRACT

An Evaluation of Error Masking Techniques for Digital Wireless Audio Systems

Renée J. Michaud, M.S.E.C.E.

Mentor: Michael W. Thompson, Ph.D.

This thesis investigates the tradeoffs associated with typical communication system designs for packetized wireless transmission systems. Communication system design requires judicious selection of source data rate, data compression technique, error correction method, modulation scheme, and latency. In particular, we investigate a wireless system for live music, focusing on masking methods to mitigate the affect of dropped data packets. Three different masking methods were developed and tested in combination with three packet sizes, creating nine unique test environments. The efficacy of the masking methods was then evaluated. The packet error rate threshold is defined as the lowest packet error rate for which the subject can hear noise interference. A Matlab graphical user interface was used to automate a human subject protocol, which was designed to investigate the packet error rate threshold for each condition. A comparison of the results and statistical analysis of the effectiveness of the masking methods are presented.

An Evaluation of Error Masking Techniques for Digital Wireless Audio Systems

by

Renée J. Michaud, B.S.E.C.E.

A Thesis

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

Approved by the Thesis Committee

_____

Michael W. Thompson, Ph.D., Chairperson

_____

Russell W. Duren, Ph.D.

_____

Jack D. Tubbs, Ph.D.

Accepted by the Graduate School
May 2011

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis would not have been possible without the leadership and encouragement of my loving, God-fearing parents, Sean and Shelly Michaud. The instruction of my father was an important part of my learning growth from the initial to the final steps of my graduate studies. My mother's reassurance and confidence in me was a staple in my pursuit of this project, during both the good and hard times. Their roles in my life will continue to give me the inspiration I need to follow the path that God has set for me. I owe my deepest gratitude to them both and am blessed to continue the electrical engineering legacy of our family.

I would like to show my gratitude to Dr. Thompson, who gave me understanding during the many phases of conducting my research. He was an important factor in my graduate life at Baylor University and a great advisor.

I extend my regards to all those who have supported me, from forming the definition of my research to completing the finishing touches of this thesis. May God be glorified on account of this thesis, which I have completed through Him.

CHAPTER ONE

Introduction

Development of Technology

The development and commoditization of low-power, low-cost digital radio systems has led to increased interest in using digital wireless audio systems for live performances. Unlike streamed audio applications such as in-home entertainment systems, the real-time constraints on live audio performances limit the ability to retransmit dropped data packets. Furthermore, the latency restrictions of real-time audio limit the use of source encoding and error correction techniques. However, since low-power digital radio systems do experience loss of data, exhibited as bit errors and/or dropped packets, there is a need for a fast, efficient method to mask the effects of lost data in live audio applications. For these reasons, an investigation was conducted on the way in which humans perceive dropped packets, and the effectiveness of several masking methods.

Human Perception and Natural Masking of the Ear

Temporal Masking

Temporal masking is a psychoacoustics phenomenon, which occurs when the signal and the masker take place at different times. The signal is generally defined as a tone or target word that we want the listener to hear, where the definition of the masker is an audible signal designed to make it difficult to hear the original signal. There are three different time locations for the masker in relation to the signal, as defined in [1].

When the masking occurs slightly before the signal is turned on, it is known as forward masking. In this case, the masking can be at most 100ms before the signal, and the effect of the masking essentially occurs forward in time. Similarly, backward masking occurs when the masking begins after the signal has already started. This effect of masking backward in time occurs when the masker starts up to 50ms after the signal. When the masking occurs neither before nor after the signal, but rather at the same time, the term is simultaneous masking. As explained by [1], these temporal masking outcomes "involve interactions between the representations of the signal and the masker within the auditory nervous system. However, there may be some overlapping of the excitation patterns within the cochlea when the intervals between the signal and masker are very brief."

Cochlear Filters

The natural, cochlear filters of the ear are powerful and important. These filters of the cochlea can be categorized into two distinct groups. These two groups are defined as the single point which defines the position on the basilar membrane and its individual filter, and a particular neuron with a unique filter.

In the first type of cochlear filters, "any point on the basilar membrane functions as a filter, attenuating frequencies that are both higher and lower than its optimal frequency," as [2] describes. The movement of the basilar membrane in its entirety defines the range for which the human ear can detect signals. This being said, only a single point of the basilar membrane needs to continue to function. Any noise entering the ear would be filtered by the particular frequency response curve of the basilar

membrane at that single point, and would be a function of the properties of the masking noise.

The second type of cochlear filter requires only a single neuron to continue to function, where each of the functioning neurons would act as a unique filter. As explained by [2], a neuron acts as a filter due to the fact that it fires more strongly in conjunction with particular frequencies, and where the surrounding neurons fire less in comparison. To visualize this more clearly, consider the neural response for a specific frequency to be associated with a given amplitude. All surrounding neurons produce responses with amplitudes of much smaller magnitudes. As with the first type of filter, this second type also depends on the properties of the band-limited masker. This combination of many cochlear filters naturally placed in the ear supports a filter model for the cumulative effect of the natural masking of the ear.

This model of perceptual masking explains the somewhat surprising observation that entire segments of an audio signal can be occasionally dropped without a noticeable loss in sound quality. By choosing appropriate masking signals, we are interested in exploring the duration for which the segments can be dropped without noise interference.

Analog Wireless Live-Audio Products

Wireless audio systems have traditionally been implemented using analog radio transmission. The benefits of an all-analog design include simplicity, low component count and cost, and the ability to maintain an analog signal throughout the system, avoiding the inherent challenges in maintaining signal quality while translating audio data from analog to digital and vice versa.

One consequence of implementing wireless audio transmission as an analog signal is that the over the air signal must be continuous. That is, the injected audio signal must be modulated to a high frequency radio signal in order to be transmitted wirelessly, and then sent and received as a continuous stream of data, occupying a given frequency channel for one hundred percent of the time, and preventing any Radio Frequency (RF) channel media sharing or frequency management. This is true for various radio frequencies and technologies, and for products ranging from high-end professional equipment to low-cost consumer products.

There are also audio quality issues with an analog radio design that are inherent with any analog transmission over a noisy medium. Since the analog radio signal received is directly down-converted to audio, any noise injected into the analog radio signal will directly affect the audio signal, and will result in audible distortion, and even cross-coupling with other signals. This type of audio degradation is familiar to anyone who has listened to AM or FM radio, and has heard static, pops or channel bleeding as the station reception gets weaker or in the presence of interference.

Digital Wireless Live-Audio Products

By contrast, a digital signal will be completely resistant to noise, up to the threshold point where the digital data is no longer recognized as a '1' or '0'. This means that the audio system has noise immunity in the presence of noise and other interfering signals. In a wireless transmission system, loss of signal inevitably occurs not only when the signal is significantly attenuated, but also typically happens intermittently when the limits of range are approached, resulting in brief periods of lost data. We will refer to this as 'dropped packets' and deal with the solution to this phenomenon extensively in

this thesis. Digital radio transmission also allows for efficient use of available radio channel bandwidth, enabling multiple systems to share a common channel or group of channels. Digital systems can take advantage of a higher over the air data rate, multiplex signals over a given channel, and can also switch channels during transmission. This allows a mechanism to handle interferers, which may appear intermittently over time.

Additionally, since digital radios allow for data to be sent in bursts, when the over the air data rate is higher than the actual audio data rate, significant power can be saved since the radio is not transmitting one hundred percent of the time. Power savings can be a critical factor in battery powered systems such as wireless audio systems for microphones and instruments. And finally, digitizing the audio signal allows for audio processing to be handled in a DSP microprocessor chip, which enables a host of features to be implemented in the wireless system, for additional integration and value.

Other motivations for transitioning to a digital wireless audio system include cost, efficiency, and a big factor: availability of wireless frequency channels. Note that we are talking about FCC and worldwide channels which are licensed for Instrumentation, Scientific and Measurement use (ISM). Audio systems have used 'TV channels' for live theater, which requires a qualified tech to set up who knows the local stations and how to avoid conflict. FCC is now opening up traditionally used analog channels for use by new equipment (ref 'white space' given to Google, Microsoft), motivating a change in the industry.

Meanwhile low-cost, battery-operated wireless radios have proliferated through seemingly every industry, calling for a revolution in the live music industry, and a pent-

up demand which can be quickly enabled by meeting the above challenges and matching a digital delivery system to the needs of human ear's perception of audio data.

Packetized Data

The advancement of digital communications has ushered in the development of packetized data transmission. Data is sectioned into small blocks called payload units, which are embedded in a format along with a packet header. The packet header typically contains source and destination address, checksum, and in-band data. While the use of packets requires the overhead of the packet header, other efficiencies are enabled which outweigh the additional bandwidth required.

By packetizing data, a single source can send to multiple destinations, and vice versa. This is done over the same network port by using destination addressing. Since the network channel may be higher bandwidth than is needed, the channel can be shared with other connections. Also, packetized data allows for more sophisticated protocols which make use of handshaking schemes and retransmissions, providing for guaranteed service and Quality of Service (QoS) links.

An example of a communication system which takes advantage of the use of a packet data scheme to carry audio data is the modern telephone network. Begun as an analog system, the telephone network now implements digital voice data, and has transitioned to packetized voice data over time. The use of this technology has resulted in lower costs and higher availability of service. In this day and age, much of the long distance voice traffic is carried over the internet, which is extremely cost efficient.

The application of a packetized data connection in a real-time, high fidelity audio channel presents some technical difficulties, specifically with regard to latency, QoS and

the handling of errors and dropped packets. However, there is strong motivation to find workable solutions to these challenges. A solution which works well and delivers high audio quality with low latency over a packetized, digital data channel will enable lower cost, higher performance, longer range, longer battery life, and more versatility. It will also allow designers and manufacturers to take advantage of the latest and greatest digital radio technology solutions.

CHAPTER TWO

Digital Communication Systems

General Framework

In this thesis we consider first the general framework of digital communication systems. Figure 1 below shows a block diagram of such a system. The input on the top of the diagram is the transmitter, where the data we wish to transmit is referred to as the message. The receiver is shown in the last three blocks. Each of these operations will create a delay, or latency, between the original signal, as injected into the system, and the received message. In the following sections, the components of the digital communication system block diagram are described and summarized.

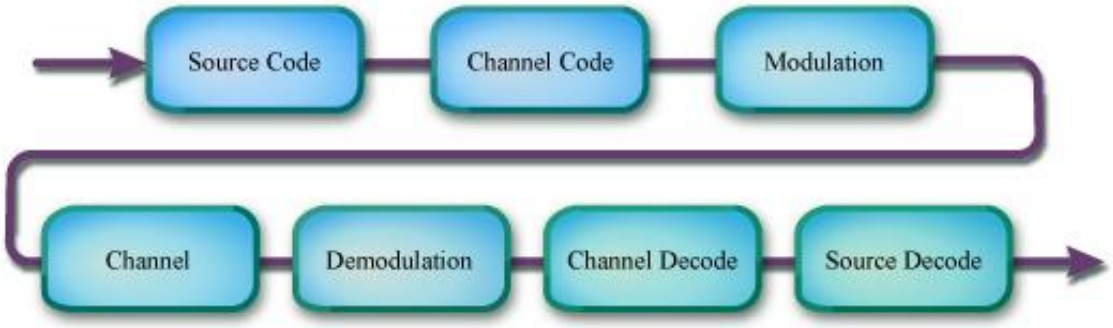

Figure 1: Digital Communication System Block Diagram

Source Coding

Source coding is the process of efficiently representing digitally, the analog source information. Examples of sources include music, speech, video, images and data. Redundancy in the message to be transmitted is significantly reduced by transforming the original message into a more condensed state. This process is also commonly known as

data compression. Examples of data compression include the MP3 and AAC formats for music, and the MPEG and WMV formats for video. A more extensive list of such formats can be found by viewing [3]. Both of these formats effectively compress the original data, while still keeping the important information needed to reconstruct the original signal during playback.

Channel Coding

After source coding, the transmitted message reaches the channel coding block. A common name for this step is error correction coding. It is anticipated that many errors will occur during transmission. Thus, through the use of error correction coding, the system can combat this phenomenon and reliably send an information sequence over the channel. The importance of error correction coding was stated in [4]: "like good equalization and proper synchronization, coding is an essential part of the operation of digital receivers." Further explanation of the code types such as those described below can be found in [5].

Block Codes

Block Codes, one of the two most common types of channel coding, are used for error detection and correction. This family of channel codes takes k-symbol source words and outputs n-symbol codewords, where n is defined as a larger value than k. In this process, redundancy is added before transmission of the message. The inserted redundancy makes it possible to greatly reduce the number of codeword errors after channel decoding. The efficiency of the code is most generally defined by the code rate.

This value is calculated by dividing the number of input bits, k, by the number of output bits, n.

Convolutional Codes

The other most common type of channel coding is Convolutional Codes. This type of channel code differs from Block Codes, in that it is not memoryless. Like Block Codes, it takes k-symbol source words and outputs n-symbol codewords. However, these codewords are computed with the use of m-memory registers (previously sent source words), not the current source word alone. Therefore, Convolutional Codes are defined by three parameters. This third parameter is usually defined not as m, but rather L, the constraint length. This can be found by multiplying the number of input bits by the subtraction of one from the number of memory registers.

Convolutional encoders are finite state machines, and thus can be described as an encoder with n binary cells and 2^n states. A Trellis Diagram is a common type of state diagram for this type of encoder which clearly illustrates the change that each possible input of the encoder makes not only to the output, but also the state transitions.

Turbo Codes

Advancement upon Convolutional Codes, Turbo Codes closely approach the limits defined by Shannon's Theorem. The design of a Turbo Code is the parallel concatenation of two codes, with an interleaver between, whose purpose is to read the bits in a pseudo-random order. These codes were the first of their kind to nearly reach the limit termed as the channel capacity, which is further explained in the Channel section below. Turbo Codes are termed short Convolutional Codes. Note the fact that

this type of code, when designed to produce results meeting the same nearness to channel capacity as previous Convolutional Codes, requires much more complex decoding practices. When choosing a channel code, the tradeoff between energy bandwidth efficiency and energy efficiency is considered. The reliability of the decoding process of a Turbo Code along with its great efficiency towards reaching the channel capacity proves the Turbo Code to be a significant improvement upon Convolutional Codes previously used. However, the cost of this type of coding is computational intensity, to the point where it is not practical with live-audio communication technology.

Modulation

In this step, the modulation block, the signal is converted to analog, and thus prepared to be sent through a medium for an extended range. A typical modulation scheme for live audio applications, specifically 2.4 GHz digital communication radios, is Gaussian Frequency Shift Key (GFSK) modulation. The Gaussian aspect of this modulation is an improvement on the typical FSK. In FSK modulation, the use of a discrete carrier frequency change represents the digital data. Typically, one carrier frequency is chosen to represent a ‘0’ and another to represent a ‘1’, although more can be used in order to increase the transmission rate.

When a GFSK modulation scheme is implemented, the digital signal is sent through a Gaussian filter before going through the FSK modulator. This limits the spectral width of the signal and creates a smoother pulse by filtering the frequency deviations, unlike when simple FSK modulator is used. The GFSK modulation scheme is standard in today’s market, being used in systems such as Wifi and Bluetooth for its spectral efficiency.

Channel

The most important property of the channel is the channel capacity. The channel capacity is a concept from information theory developed Claude Shannon, which establishes the performance capability for a given communications system. The channel capacity result describes the performance bound on error free communications as a function of the bandwidth of the channel and signal-to-noise power ratio [3]. For a real-time transmission system, the Nyquist rule of thumb is where the bit rate is roughly equal to the bandwidth.

The difference between the ideal system and what is achieved in practice can be attributed to two factors. The first of these is the redundancy in the message to be sent through the channel. Secondly, errors due to channel noise can occur and coding methods are required to mitigate the effects. Problems in achieving the desired channel capacity prove to be bounds on how well the system will perform. Achieving performance close to the channel capacity bound results in additional complexity thereby leading to an increase in delay time.

Receiver

Overview

The receiver performs Demodulation of the radio RF signal, Channel Decode of the digital data, and Source Decode of the digital signal to be converted back to analog audio data. In a digital wireless audio system, these functions will typically be performed by three sub-systems; the radio, a digital signal processor, and a digital-to-analog converter.

Demodulation

The radio sub-system must receive and demodulate the RF signal, which is the process of extracting the intended digital data from the carrier. In the case of a GFSK modulation scheme, this demodulation essentially consists of recognizing the carrier frequency, syncing with the modulation baud rate, and translating each frequency shift in the carrier into the corresponding ones and zeros encoded in that RF signal. The translation from analog RF to digital ones and zeros is done in the 'front end' section of the radio, known as the Physical Layer (PHY). The radio PHY layer in a modern digital radio is a combination of analog and digital circuitry, often using significant DSP processing power to interpret the received analog signal and perform the GPSK demodulation.

The radio then processes that stream of digital data, to determine if a valid message is received, and if that message is intended for this particular receiver. This is referred to as the Media Access Layer (MAC). The output of the MAC is a valid data packet, including any packet header information such as source and destination address, and the CRC checksum, as well as the data payload itself.

Channel Decode and Source Decode

The radio then passes this data packet to the digital signal processor, which performs the Channel Decode function. Here the data payload is examined, and processed to derive the actual digital audio data to be converted back to analog. In a digital wireless audio system, such channel decode processing might be very minimal, and consist of simply translating a digital packet stream into a number of sequential digital audio words to be forwarded to the Digital to Analog converter. However, if there

is any Forward Error Correction (FEC) implemented in the Channel Encoding section on the Transmitter, the reverse operation must be performed on the Receiver side by the Channel Decoder DSP. Also, there exists an opportunity to massage the received data on the Receiver side, which is implemented in order to reduce the effects of any dropped packets or bit errors encountered in the channel, independent of any FEC encoding. This opportunity is the basis for the work in this thesis.

Finally, the Channel Decoder delivers a digital stream to the Source Decoder, which converts that data to an analog audio stream, to be output from the digital wireless audio system.

CHAPTER THREE

Particular Constraints of Digital Live-Audio Systems

Audio Quality

Digital audio is delivered in a variety of formats designed to meet trade-offs between data size, and audio quality and usability. For the purposes of this thesis we will focus on CD quality digital audio, which is broadly used, well understood and considered to deliver good quality audio. Setting aside compressed formats such as MP3, which produce a smaller data stream at the cost of some loss of audio data, along with a rather large computational delay due to compression, CD format is by far the most common audio format. Although professional recording equipment may record, mix, and process at a higher quality, the final audio is nominally mixed down to CD quality output. CD quality digital audio is defined as 16-bits per sample, and 44.1k samples per second.

Dynamic Range

The number of bits per sample drives the audio dynamic range, which limits the difference between the loudest and the quietest signal which can be recorded, as well as the ability to separate the audio signal from the background noise inherent in any audio recording or reproduction. By definition, each bit in a sample doubles the number of possible audio levels, and therefore each bit represents 6dB of dynamic range. Audio that is 16-bit then delivers a maximum dynamic range of 96dB. This is assuming that the input signal is called properly for the full dynamic range of the quantizer.

Frequency Response

The number of samples per second determines the frequency response of the delivered audio. The Nyquist limit determines the maximum frequency which can be represented in any digital signal is one half of the number of bits per second (bps). Therefore a 44.1kbps digital audio signal can produce no more than half of 44.1k, or 22.05 kHz audio. The goal of CD quality audio is to represent up to 20 kHz signals; the maximum frequency which a young healthy ear can typically hear in the best of circumstances. The reason that a higher sampling rate was chosen, rather than simply double the 20kHz target, is that when the audio signal is digitized through the Analog to Digital Converter (ADC), the exact phase of that signal, and the ability to sharply cut off higher frequencies which may produce aliasing, requires some headroom above the target maximum desired frequency, and therefore 44.1k samples per second was chosen.

Total Harmonic Distortion

Because the audio signal must be digitized, transmitted through the radio channel, and recreated on the receiving side, there are specific audio quality parameters which must be measured and kept to within boundaries, in be able to carry high quality audio through the signal chain. Among these is Total Harmonic Distortion (THD) which measures the harmonics of a given (single frequency) signal which are unintentionally added to a signal. This type of distortion occurs in the audio sections of the hardware and can be seen and measured with a spectrum analyzer. As this thesis focuses on the handling and masking of audio anomalies due to lost data, we will not focus on THD, but simply note that this is a key parameter measured and controlled in high quality audio systems, especially those in which the volume is expected to be quite loud.

Latency

Latency is defined as the delay in delivering the audio signal through the entire system. This is a negative byproduct of a digital system, and is directly affected by the audio processing methods considered for this study. The latency budget for a live wireless audio system is much stricter than, for example, streaming music. This is because for streaming music, the delay between starting the music and hearing it through the speakers is not seen as a critical problem, whereas immediate response is expected during a live concert.

In the case of wireless audio transmission for a vocal microphone, or an instrument such as a guitar, the total latency needs to be well below 20ms, and a typical goal is to be between 5ms and 10ms total. Of this budget, at least 1-2ms will be taken up by the ADC and Digital to Analog Converter (DAC), and by the radio transmission, and another 1-3ms for audio buffering on both sides of the radio. This leaves approximately 1ms to 8ms for any additional audio processing.

Note that for the current processing speeds under consideration this precludes using such methods as Reed Solomon Encoding, used in audio CDs, to correct bit errors, since these methods use block encoding. Block encoding requires a large block of audio data to be encoded, sent over the channel, then decoded on the receive side, adding latency which is acceptable for CD play, but is not acceptable for live audio systems. Similarly, masking methods chosen to handle lost data should be designed to work well without requiring large blocks of data, or large amounts of CPU time to process.

Connectivity and Range

Finally, a wireless audio system will have been designed and specified to meet a given range, and to maintain connectivity over distances and circumstances that make it usable in the target application. While range is hard to guarantee, and depends greatly on the environment, the measurement of range is assumed to be based on the distance at which the connectivity is very high, and at which the audio signal quality is very high. Therefore, the ability to mask dropped data can boost the effective range of a digital wireless audio system, since it can allow the perceived connectivity to be much higher than without such masking, and allow the use of the system in conditions and at a range which is greater than without masking implemented.

CHAPTER FOUR

Hardware Platform Design and Implications

Advances in Technology

Advances in microprocessor and radio system designs have made real-time transmission of digital audio data feasible. Modern radios have adequate bandwidth and power capabilities. Silicon integration and miniaturization have reduced the power requirements to the point where solutions can be small, battery powered, and cost effective.

There are, however, a number of radio integrated circuits (IC's) available from companies such as Texas Instruments and RF Micro Devices. These are capable of transmitting digital data at a data rate of up to 1Mbps, using a clear channel which is not burdened with the overhead and latency of protocols such as Wi-Fi or Bluetooth.

Performance Requirements

In order to meet the audio quality requirements described above, components must be selected which meet the audio sampling rate and bit depth. Also, a radio system must be selected which can support the required data rate, operate within the selected band, and meet the range requirements. In addition, any audio processing used, including masking of dropped packetized data, must be handled by a processor which meets the required processing speed, memory size, and in/out (I/O) peripherals.

Assuming a CD-quality audio is specified, the effective audio payload data rate which must be supported through the channel is 24 bits per sample multiplied by 44.1k

samples per second. The resulting audio payload data rate is calculated to be 1058.4kbps. This audio payload data rate, along with any overhead, in-band signaling, and frequency band management scheme, drives the total radio throughput.

In addition to audio payload data rate, the latency of the system, defined as the end to end delay, needs to be very low for any live audio system. The latency through the system is the sum of the delay through the ADC, the DAC, and the radio, in addition to any radio protocol layers and audio processing. Common radio protocol layers, such as Wi-Fi and Bluetooth, are generally large and indeterminate. For these reasons, they are considered inappropriate for real-time digital wireless audio data delivery. However, there are QoS protocols that exist, which if applied carefully can limit the latency for the radio protocol.

If a clear channel radio transmission scheme is used, then no radio networking protocol exists, by definition. The latency can then be determined by careful design and implementation. In this particular case, the largest contribution to system latency is the audio processing. This contribution to the system latency, along with the limitations of such audio processing, are described further in this thesis.

ADC and DAC Specifications

The hardware design of a digital audio system includes the selection and design of the ADC and DAC which meet the audio performance requirements. As described above, this includes the sampling rate and bit depth.

Other important specifications in selecting the ADC and DAC include the digital audio interface, the number of channels (e.g. stereo vs mono), the hardware/software control interface, and the power requirements. Furthermore, the hardware design of this

section is critical since the analog section is susceptible to digital noise from the microprocessor and radio sections. This should be handled in the Printed Circuit Board (PCB), which is the physical placement of the system and affects the performance of the analog section. If the digital noise problem is not handled properly in the PCB design, the audio quality of the system will be degraded.

Radio Specifications

The selection of a radio is driven by the data rate, the radio modulation technique, the power requirements, the transmitter output power, the receiver sensitivity, and parameters such as blocking immunity and co-channel rejection. In addition, the support or lack thereof for streaming data, and the ability to quickly change radio frequency channels, drives the decision as whether to use packetized data versus streaming data. This also aids in the decision for how to manage the RF radio channels.

For the purposes of this thesis, we will assume that the hardware platform dictates a packet data radio channel, with roughly twenty percent overhead dedicated to packet headers, inter-packet gaps, and radio channel management.

Typical radio data rates are in integral multiples of 128kbps. For example, a 500kbit, 1Mbps, and even 2Mbps radios are available. Notice these amounts are even numbers, not odd, for example a 1.5Mbps radio. Furthermore, the cost of a higher data rate includes power consumption and range, since the effective receiver sensitivity is generally lower for higher data rate modes. Hence, the lowest data rate which supports the requirements of the system should be chosen.

These practical considerations, alongside the actual audio data rate for CD quality audio, lead to the selection of a 1Mbps, packetized audio data radio solution. The

maximum packet size is then determined by the radio specifications, and the available buffer space. A typical maximum packet size is 127 data bytes, which is driven by the specifications for ZigBee radios.

Important Considerations

Beyond these tradeoffs, the hardware design must consider power, cost, size, and availability of components. In a battery powered system, the current requirements for the system are critical, and each component must be selected to help meet the total power budget. Also, in order to be cost effective, components should be selected which can run on the same voltage, or on as few voltage rails as possible.

Another very critical area is the radio antenna section. A variety of antenna types are available, with associated tradeoffs between size, performance, cost, and also antenna radiation pattern. If a system is to be fixed (not moving) then a directional antenna may be appropriate, and an increase in gain can be achieved by directing the transmission power and reception pattern toward each other. In a system which is handheld, or intended to be used or mounted, in any configuration, an antenna pattern should be chosen which is non-directional. This will help to avoid a null, which could reduce effective range.

CHAPTER FIVE

Error Masking Methods Employed

Forward Error Correction

Hamming (7, 4) Code

The first method attempted in order to correct the dropped packets in the sound samples by way of FEC was the widely used Hamming (7, 4) Code. The use of the three check bits added to the word with the use of the Hamming (7, 4) Code allows the system to correct any 1-bit error. It can also detect the 1-bit and 2-bit errors that arise. One of the advantages of this type of FEC is that you "do not have to decide in advance whether to use [the Hamming Code] in error detection or error correction mode," as explained by [6]. Further understanding of the Hamming (7, 4) Code can be found by reviewing both [6] and [7].

When a binary 4-bit word is encoded with a binary Hamming (7, 4) Code, it is transformed into a binary 7-bit word by adding three parity bits. The method for adding these three check bits is aptly described in [6]. In short, the decision whether to assign a one or zero to each of these three check bits is done by looking at the ones and zeros in the 4-bit word. Certain combinations of the bits in the given 4-bit word and a particular parity bit must give an even number of ones.

In order to decode a transmitted 7-bit word that has been encoded using the binary Hamming (7, 4) Code, parity checking must be employed. Simply put, it must be found whether an even or odd number of ones can be found in each of the unique combinations

previously defined in the received word. With this information, it can be determined how many errors have occurred and correct one, remembering that this particular method can detect all 1-bit and 2-bit errors, and correct a 1-bit error. A schematic of this decoding process is shown in Figure 2 below. Examination of the schematic further explains the method for which decoding using the Hamming (7, 4) Code is completed.

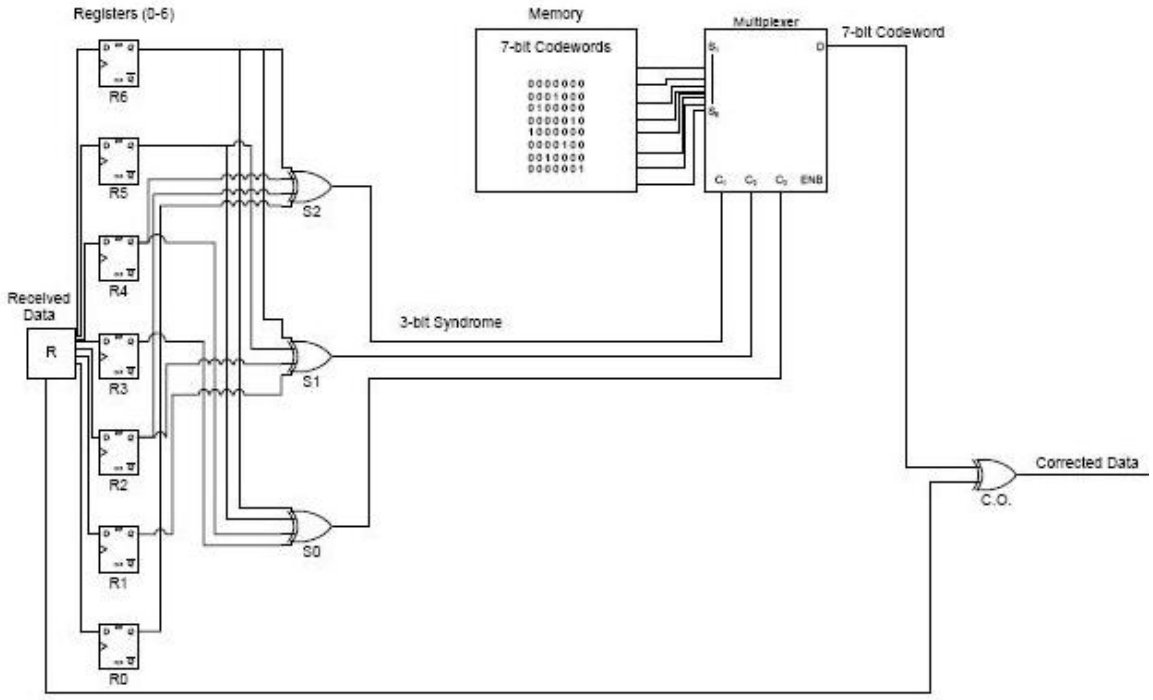

Figure 2: Decoding Schematic for a Hamming (7, 4) Code

This type of encoding and decoding, the Hamming (7, 4) Code, was implemented in Matlab. The Matlab code for the encoder can be found in Appendix G, with the Matlab code for the decoder in Appendix H. It was found that although the binary Hamming (7, 4) Code works well for single-bit errors, it was not suitable for dropped packets which would occur in a typical digital wireless communication system. It was

also determined that with the strict delay constraint of the system, implementing a scheme that requests retransmission of dropped packets was not feasible.

Protecting the Nonzero Bit Positions (NBP) with Parity Bits

Another type of FEC attempted took advantage of the use of parity bits. A string of data words was fed into a function, which can be found in Appendix I, which would then calculate the parity bits for each of the words. Many data samples, when using 24 bit quantization, are sufficiently small in value that several of the upper bits of the word are effectively zero (with sign extension in the case of negative numbers). The benefit of this approach is that parity bits can be used to protect upper four NBPs of the codewords. Errors that occur in the lower bit positions do not affect the signal-to-quantization noise ratio as much as the upper four NBPs. The Matlab code for implementing this can be found in Appendix J.

The desired improvement using this type of compression FEC is just like any other when compression is used: improved bandwidth. This can be done while keeping the efficiency of transmitting the entire words due to the fact that most of the information being sent through a codeword is contained in upper four bits of the NZBs. The FEC was executed by grouping several codewords together and sending them as a string of codewords. Then, the space saved by throwing away the zero-valued bit positions was used to send the parity bits. The encoding was done by taking a 24-bit sample and using a running estimate of the signal scale, and hence the NZB locations, to determine the most significant 16-bit portion of the codeword to apply the FEC. Therefore, the bandwidth is conserved and, by sending the MSB along with the corresponding parity

bits, data can be sent with parity bits used for error correction without significantly delaying the signal.

The data compression and FEC of this method proved to be efficient at small error rates. At the bit error rate level where a difference could be heard, it became apparent that at this noise level, packets would be dropped. However, this method cannot be modified to account for dropped packets. When packets are dropped, the method for which the NSB's are kept with the parity bits simply cannot correct such a loss in data. Furthermore, requesting that packets be retransmitted was also determined not to be practical because of the time-delay constraints. Instead, we determined that masking techniques would be studied and created in order to better fulfill the delay requirements of the system while resolving the problem of dropped packets.

Choosing the Masking Methods

The most complex part of creating the sound samples to be used in the human subject testing is the method of masking used when a simulated dropped packet occurs. The purpose of creating the masking methods is to deliver sound samples with better sound quality under the constraint of a prescribed packet error rate. Although it is believed that allowing a certain level of complexity in the computation of the masking methods would produce improved sound quality, it also generates a delay in the system. As explained previously, the delay constraint of a live-audio digital communication system is a major consideration whilst crafting the three masking methods.

Three masking methods were chosen to implement: blanking, repeat, and low-pass blend. These three masking methods were selected for their tradeoffs in complexity and potential usefulness when masking dropped packets.

Blanking Masking Method

The first masking method features the simplest computations. In this method a dropped packet is "blanked out" by replacing the entire dropped packet length with zeros. The blanking masking method will be used in conjunction with the three packet lengths chosen, simulating a single dropped packet along with a small and long burst of dropped packets. An example of this masking method is shown in Figure 3 below. In this figure, the blanking masking method is used to mask a single dropped packet.

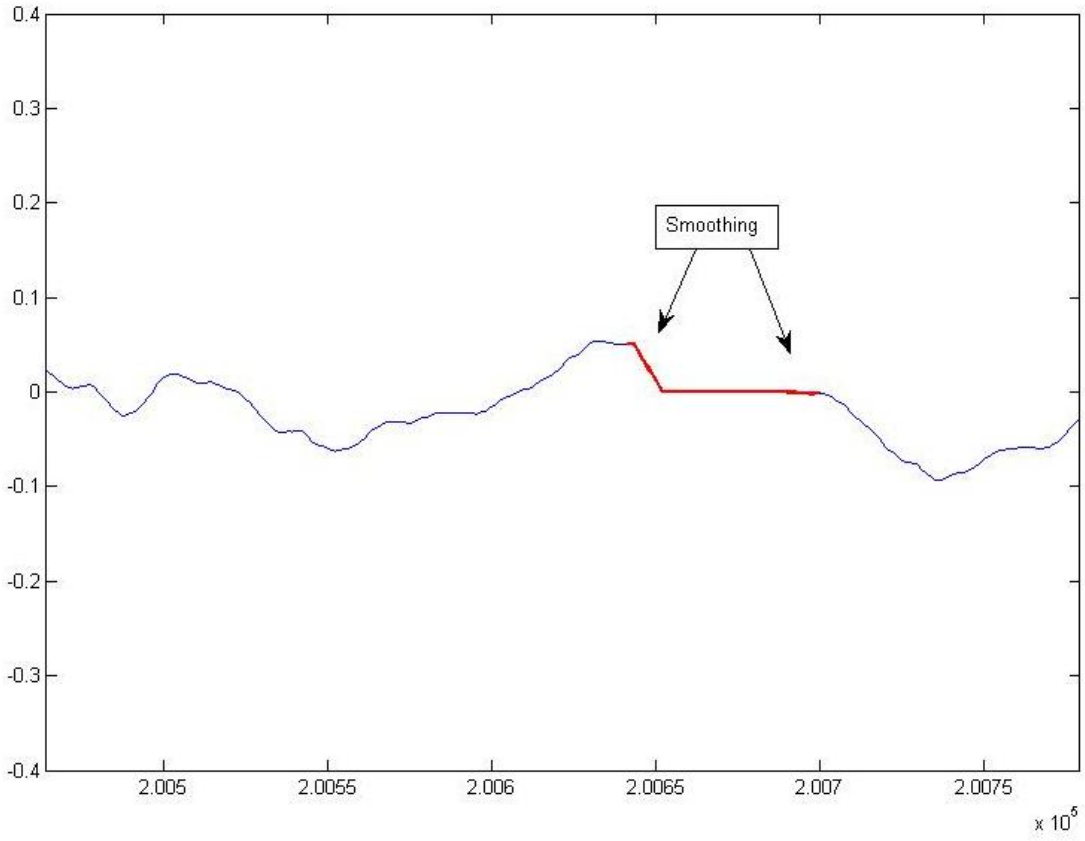

Figure 3: Blanking Masking Method Used to Mask a Single Dropped Packet

The blanking of the entire dropped packet can be seen as highlighted in red in the Figure 3 above, along with short ramped (linearly interpolated) edges on either side.

These ramps are effectively a smoothing technique, used to decrease the consequence of noise and increase the sound quality of the sound sample. This smoothing technique is indicated by the arrows in Figure 3, and will be further described in later section.

Repeat Masking Method

The second method developed is the repeat masking method. The procedure for this masking method keeps track of the last successfully sent packet, and places it wherever a packet is dropped. This masking method involves more computation than the blanking masking method. The computation involved in this repeat masking method is not so great as to exclude this method from consideration for a digital wireless communication system where a strict delay constraint is a priority. Further motivation for repeating a dropped packet is that for many portions of an audio signal, the spectral content is relatively unchanged between adjacent packets.

In Figure 4 below, the repeat masking method is demonstrated. The previous packet, which had been repeated, is enclosed in the box shown in Figure 4 below. The smoothing technique applied to both edges of the replaced packet is indicated with arrows, and will be explained further in a later section. This masking is shown for a single dropped packet, as with Figure 3 for the blanking masking method above. In addition, the masked dropped packet shown here has the same location as the dropped packet masked with the blanking masking method, so as to better compare the success of the masking methods.

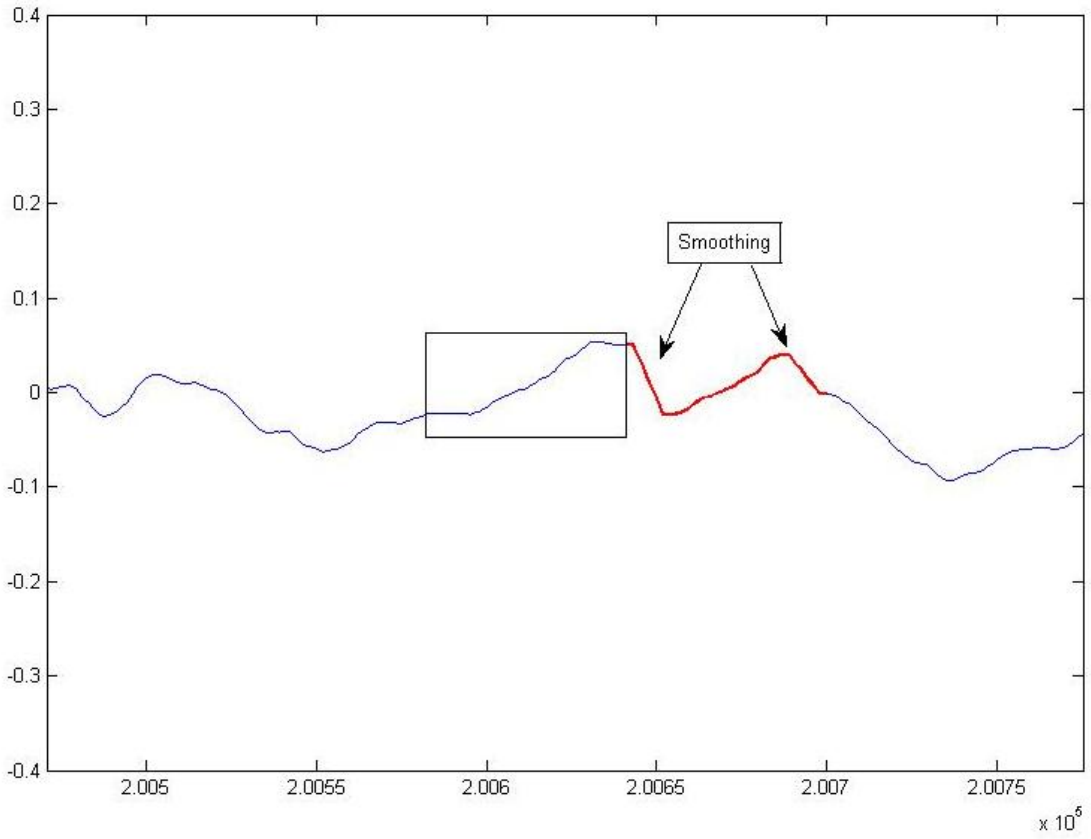

Figure 4: Repeat Making Method Used to Mask a Single Dropped Packet

Low-Pass Blend Masking Method

The most complex of the three proposed masking methods is the low-pass blend masking method. The unique property of this masking method, in contrast to the others, is the increased computation requirement. This was initially conjectured to yield a sound sample of sound quality above the rest. When a packet is to be simulated as a dropped packet, it is first replaced with the last successfully sent packet. This can be seen in the Figure 5 below.

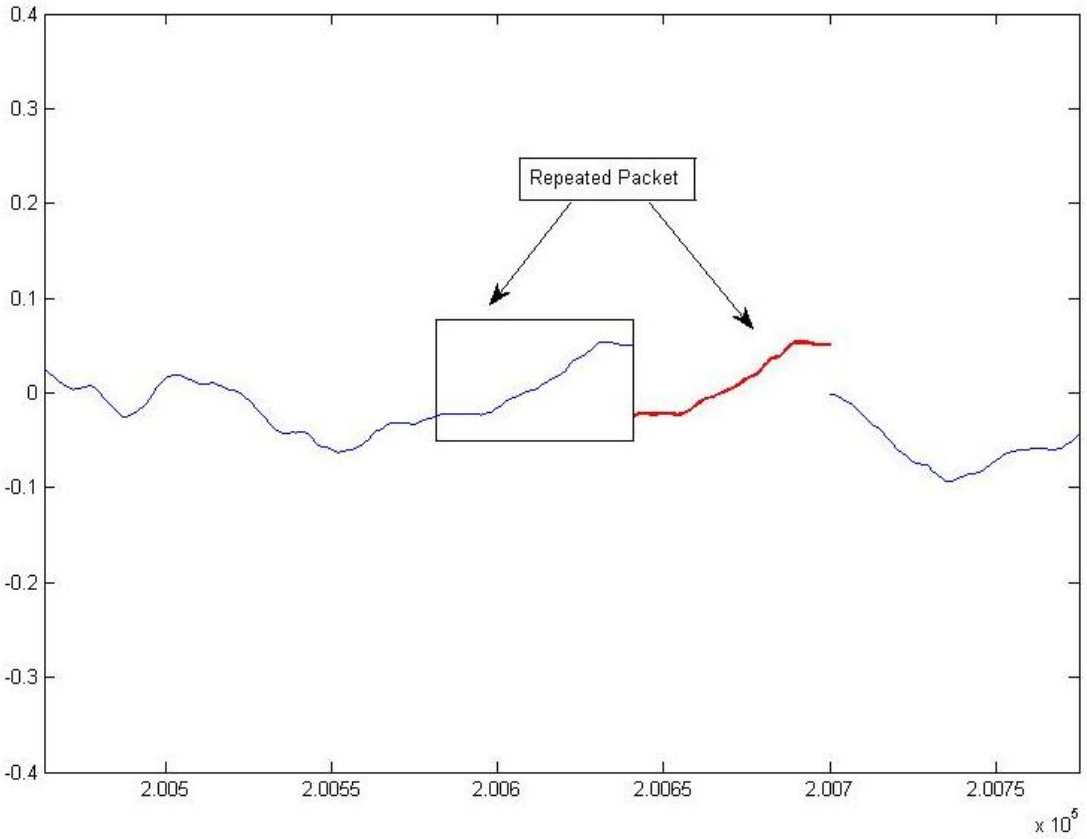

Figure 5: Repeat of a Single Packet for Low-Pass Blend Masking Method

After replacing the dropped packet with the last successfully sent packet, it is then passed, along with the last packet, to another Matlab function. This function implements a blending procedure that effectively interpolates a "low-pass" blend between adjacent packets. Once these two packets are given to the function, the same two packets, now with their connecting edges blended, are the outputs of the function. Because this function blends the current packet only with the last packet, the current packet must logically be passed to this blend function again, this time with the next packet to be sent. Figure 6 shows the low-pass blend masking method employed. As before, the masking is done for a single dropped packet in the same location as the others, for ease of comparison between the masking methods.

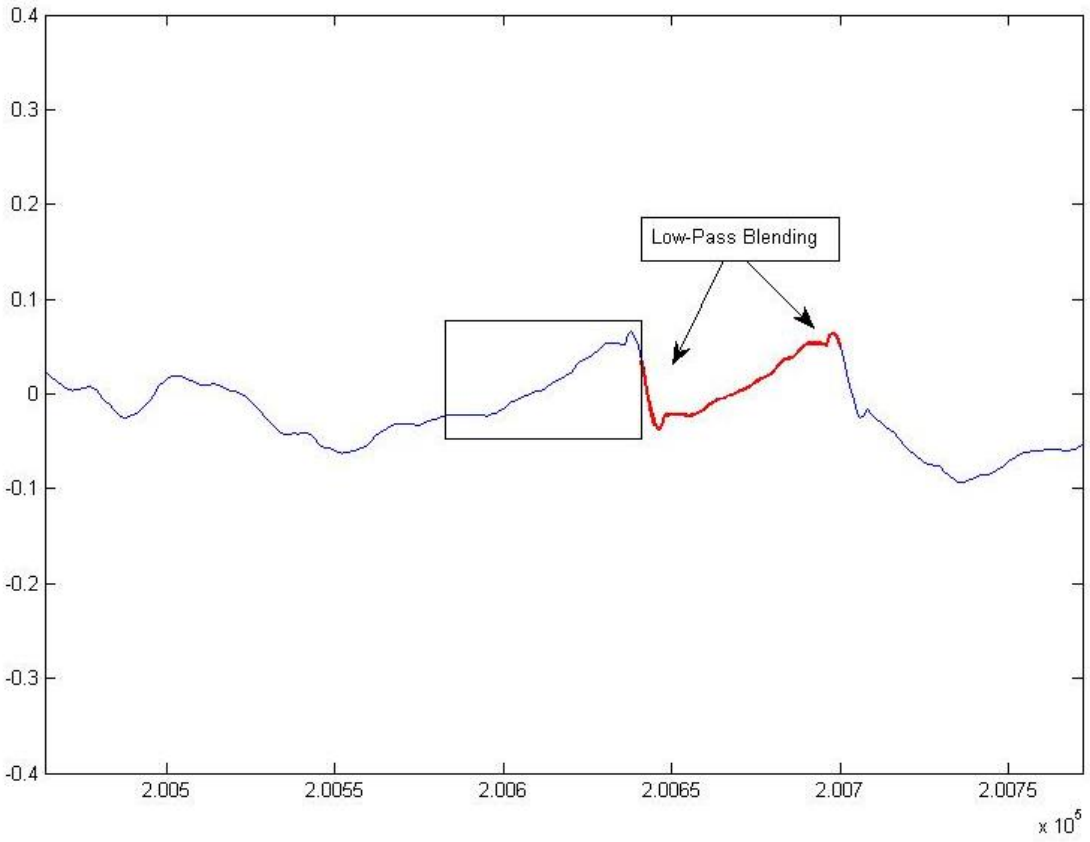

Figure 6: Low-Pass Blend Masking Method Used to Mask a Single Dropped Packet

The blend function is the heart of the low-pass blend masking method, and can be reviewed in Appendix D. As initializations, he number of samples of the dropped packet to be blended with the adjacent packet is defined, along with how many sub-points each point of the dropped packet will be divided into. Using these definitions, the portion of the two packets which are to be blended can be identified as the end of the previous packet combined with the front of the current packet. The low-pass blend masking is then accomplished by decimating the end portion (the portion to be used in the blend) of the previous packet by a given downsample factor, M. A DFT based interpolation method is then implemented which takes the DFT of the decimated samples and then zero pads in the frequency domain in order to preserve the original sampling rate. The inverse

DFT is then taken to produce the low-pass blend. Note that this blending method effective produces an interpolated blended signal that has a maximum frequency of f_s / M. The back end of the previous packet and the front end of the current packet are then swapped for this blended, low-pass filtered portion of the two packets.

This process is repeated for the dropped packet and the packet to follow, thus using the low-pass blend technique on both sides of the simulated dropped packet. This low-pass blending technique is indicated by the arrows in Figure 6 above, where the previously dropped packet is highlighted in red. This method of replacing the dropped packet for the last successfully sent packet, and then implementing a low-pass filter blend boasts a smooth transition both from the previous packet and to the next one.

CHAPTER SIX

Designing the Human Subject Tests

Distinct Specifications

Each of the human subject tests is created with a different combination of a packet size and masking method. With three packet sizes selected and three masking methods created, the combinations of each specify nine unique tests. Twelve sound samples were created for each of these nine tests. A vertical bubble list was then used to listen to the twelve sound samples created. The order in which the subject would take the nine tests was also varied, determined by their subject number using the Circular Order Technique. This is to prevent skewing the data and is further described in chapter seven.

Packet Sizes

Packet size, which is defined as a function of delay parameters, is one of the fundamental attributes used to create the sound samples. Three different packet sizes were chosen, with use of information about current wireless music systems, to represent lost data in a typical wireless live-audio communication system. The sizes of these three lost data sets (LDS) of dropped packets were defined as 180, 540, and 1620 audio words. Errors in a wireless channel are bursty in nature, and a period of poor connectivity is likely to affect not just one or a few bits, but rather one or a few packets in a row. Therefore, the LDS size is chosen to represent one, three, or nine blocks of (180 audio words) lost due to interference or poor connectivity before the channel recovers. Note

that even one block of 180 audio words may represent multiple data packets, depending on the size of the packet chosen, and that the selection of the radio data packet size involves tradeoffs between packet header size, efficiency of the channel, latency and radio buffer sizes However, this thesis focuses on the aggregate effects of LDS of varying sizes, and the ability to hear the effects and to mask the effects to the hearer, not on selecting optimum packet sizes for efficient data transmission.

Error Rates

A set of five error rates was fine tuned to correspond to five of the twelve sound samples used in the tests. To select these error rates, a wide range was first selected according to the calculation which showed how many packets would be dropped on average for the particular sound clip being used. A sub group of participants was used to evaluate the typical packet error rate threshold for the current set of error rates. After a few rounds of testing, the final set of error rates was determined to be the probability static error rate set of [.3e-2 1e-3 .3e-3 1e-4 .3e-4]. This set represents the typical error rates found in a digital wireless live-audio communication system for which the subjects could detect during evaluation. For this reason, the chosen error rates increase logarithmically, as that is the way in which the human ear can detect noise.

The top bubble indicates the most number of dropped packets in the sound clip, likewise having the least number of dropped packets, chosen as none, at the bottom of the bubble list. The placement of the static set of five error rates was shifted at random for each test so that the packet error rate threshold could not be intuitively found. Those sound samples above the static set of five error rates were created using the maximum error rate, whereas those below the static set were left with no errors.

Sound Samples

Due to the way a person compares sound samples, they are limited to seven seconds each. In this way, the sound sample is long enough for the user to listen for noise, yet short enough where if the randomly generated dropped packet positions are at the end, they will still hear them. The type of music is important for this type of testing. Wanting to test a case where the noise will not naturally be masked by the music, I chose a steadily mellow instrumental piece. In this way, the music is loud enough so that noise will be heard, yet not so loud that it will naturally mask too much of the dropped packet noise.

CHAPTER SEVEN

Matlab Graphical User Interface

Purpose

The Graphical User Interface (GUI) was created in order to find the packet error rate threshold for several sound clips with different attributes. In this way, it can be determined which combination of these attributes has the highest packet error rate threshold, and therefore is the best candidate for increasing the range and sound quality of a digital wireless live-music system. The purpose of the GUI is to provide a method for presenting, comparing and recording results for each of the twelve different sound samples in the nine separate tests.

The GUI can be used to generate all of the permutations of the test conditions. Each specific condition is determined by packet size and method for masking the perceptual effects of dropped packets. A screen shot of the interface is shown in Figure 7 below. Each subject will be asked to select the radio button in the lowest position for which dropped packets are audible.

Implementation in Matlab

Dropped Packets

For each of the nine combinations of masking method and packet size, twelve sound samples needed to be created for the test. The Matlab m-file for automatically creating all nine tests can be found in Appendix B. To create these sound samples for the blanking and repeat masking methods, the corresponding packet error rate

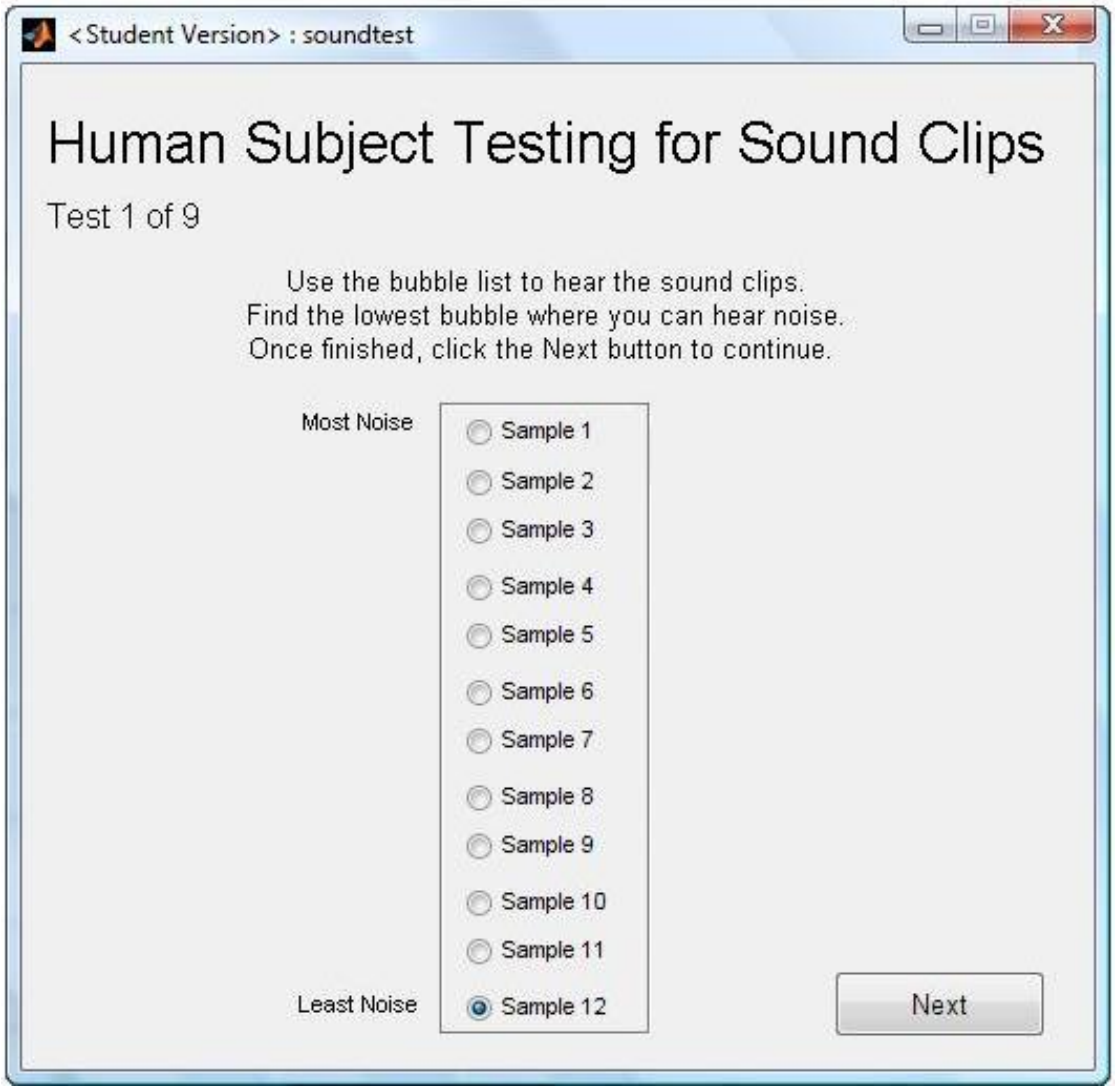

Figure 7: Graphical User Interface for Human Subject Testing

was used in conjunction with the Matlab m-file titled makeclip.m. The Matlab m-file makeblend.m was used in the same way for the low-pass blend masking method. These files can be found in Appendices C and D, respectively.

The code for corrupting the sound samples with dropped packets can be found in these files. It was found that although the sound sample chosen was evenly mellow, the random placements of the dropped packets still made a difference when listening for the

corruption noise. To solve this problem, random locations were selected for the dropped packets to be simulated. This list of locations was then shuffled and saved. The Matlab code for creating this list of locations is titled droparray.m and can be found in Appendix F. For each error rate, dropped packets were simulated in the corresponding number of locations, which were previously calculated. In this way, a sound sample with a large error rate had dropped packets in the same locations as that of a sound sample with a smaller error rate, including extra locations. This method for placement of the simulated dropped packets effectively kept the potential variance in placement from being an issue.

Masking Methods

The three masking methods previously described were each used in conjunction with all three of the defined packet lengths. The Matlab file titled make_clip_params in Appendix B would identify the current masking method to use. Here the unique parameters to create the tests were also defined before being passed to the function which would create the individual sound samples for testing in the GUI.

In order to implement the blanking masking method and the repeat masking method, the Matlab file make_clip in Appendix C was created. After the properties are passed to this function from the higher-level code in the make_clip_params file, this function will have everything that it needs to create a unique sound sample, including which masking method to implement.

As a simple overview, definitions and initializations are first created. Then, according to which masking method is assigned, the 'replacements' is defined as either zero or the last packet successfully sent. With the error locations previously calculated using the particular packet as described in the previous section, the placement for a

dropped packet can simply be found by checking the current indexing with the array of error locations. When it is found that the current packet should be simulated as dropped, it is replaced with the contents of 'replacements' and smoothing is done on both sides of the new packet.

For the third masking method, low-pass blend, the overall process is the same as the previous two. The definitions and initializations are done in the same manner. However, while examining the Matlab code in Appendix D, titled make_blend, it can be seen that there are distinct differences in the calculations.

As described in a previous chapter, the low-pass blend masking method is somewhat the combination of the repeat masking method and complex blending techniques. The implementation of the blend for this technique can be found in the function titled blend, which can be found in Appendix E. Here the complex blending is implemented as already described in a previous chapter.

In order to blend both sides of the current packet successfully, there must be an added delay. The location for the next packet must first be checked against the error locations. If it is to be dropped, then the replacement of the last packet must be done before blending with the previous packet. To keep track of when two packets are dropped in sequence, flags are used. This can be seen in the make_blend code, as seen in Appendix D. When this is finished implementing the low-pass blend masking method, each simulated dropped packet will have been replaced with the last successfully sent packet, and its edges blended with both the last and the next packets.

Smoothing

Although the masking methods created made dramatic differences in the sound quality of the sound samples, graphs of the masked dropped packets showed that the sides of the dropped packet still created sharp edges. Figure 8 below shows an instance where the blanking masking method was used to mask the dropped packet. However, it can be seen that the sharp edges are left on either side of the masked dropped packet, creating a pop to the human ear. This problem occurred both when using blanking masking, as shown, and likewise with the repeat masking method. The other masking method, low-pass blend, did not require smoothing on its edges, for it already used a complex blending method in its implementation.

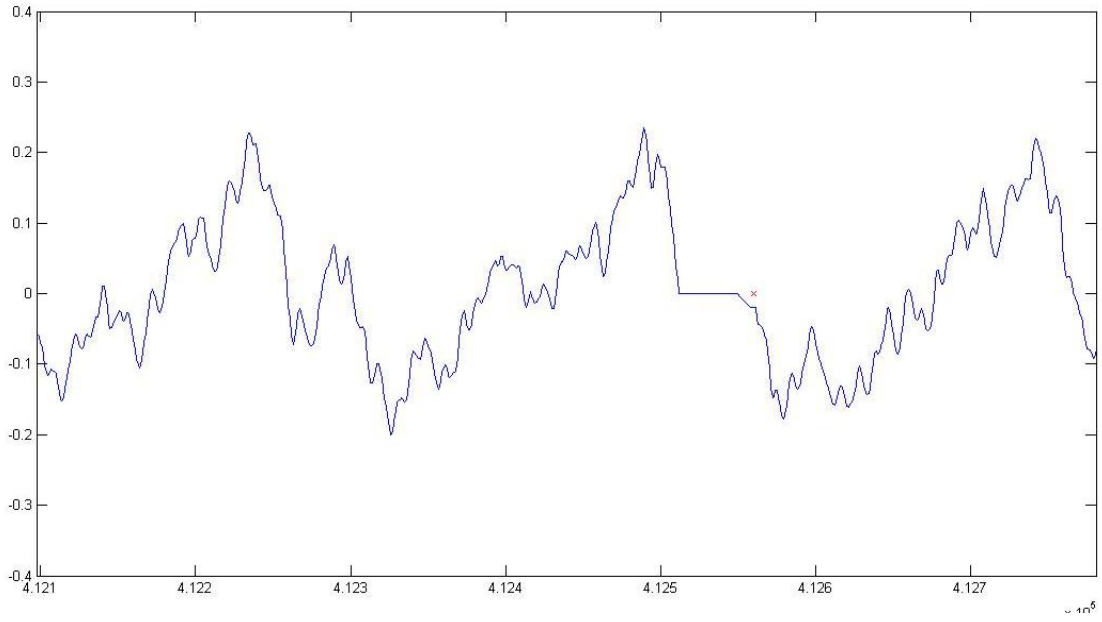

Figure 8: Blanking Masking Method for a Single Dropped Packet

The first method used to get rid of these sharp edges was cross fading. To apply this to the front end of the packet, a piece of the end of the last packet is flipped. It is

then weighted and added to the front of the masked dropped packet of the same size, with opposite weights. This method creates a fade from the last packet into the current one. The same steps were taken to cross fade the end of the current packet. However, after examination, it was found that cross fading in conjunction with both blanking and repeat methods did not smooth the current packet with its surrounding ones as expected. With the edges being strong for both ends, a method was needed which would further blend the edges of the packets together.

After experimentation, a successful smoothing method was created for both blanking and repeat masking. To reduce the popping effect of the harsh edges on either side of the masked dropped packet, the edges were blunted. To do this for the front of the dropped packet, the last sample of the end of the last packet was held for a short period. The same was done with the first sample of the current packet. Although this did not completely get rid of the edges, it greatly reduced them. Next, the two points were connected with a simple linear equation. The result was effective; the pop created by the harsh edges was reduced. Both simple and effective, this smoothing technique was a great asset to the masking methods.

The figures below show an example of this smoothing technique in action in combination with the repeat masking method. The first figure, Figure 9, a single dropped packet is shown. This is what would occur in a digital wireless communication system when the packet length is 180/44.1ms. The loud pop associated with such a break in data is what our masking methods aim to ease for the human ear.

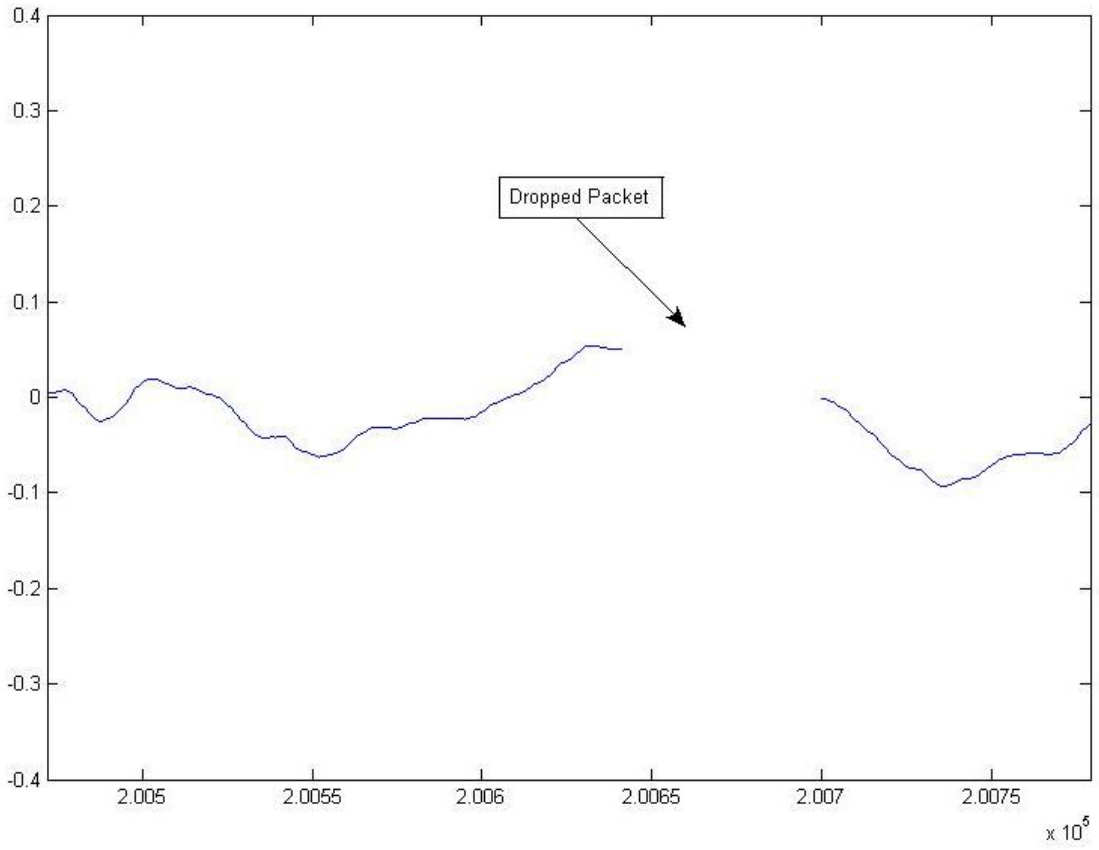

Figure 9: A Single Dropped Packet

For the repeat masking method, when a dropped packet occurs it replaced the hole where data is missing with the last successfully sent packet. This can be seen in Figure 10. The repeated packet is highlighted in red, with the last successfully sent packet enclosed in a box. It is obvious from reviewing this figure that simply repeating the last packet will not be an effective masking method. With the sudden break in data at both edges of the previously dropped packet, pops will occur. Although not as loud as the pop when the entire dropped packet is missing, these two close pops will most certainly be heard by the human ear.

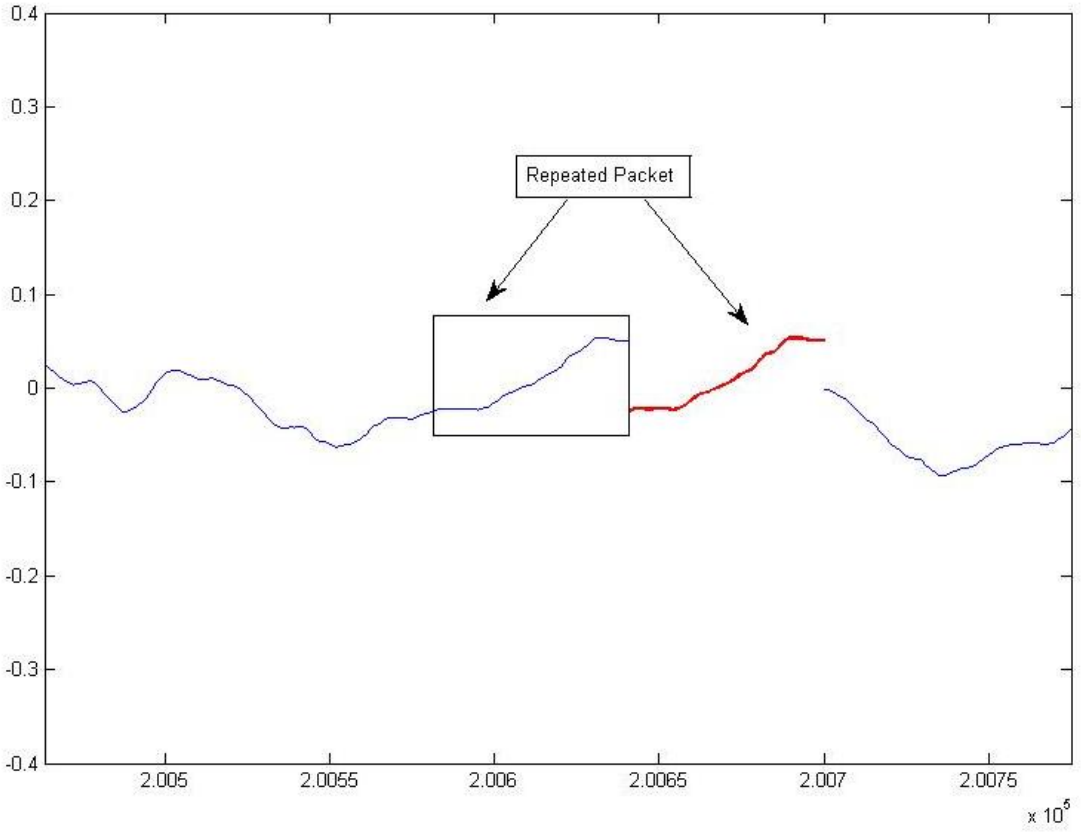

Figure 10: Repeat of Last Successfully Sent Packet

To ease these pops, the smoothing technique, as described, is implemented on both ends of the previously dropped packet. Figure 11 below shows the result of this additional smoothing. Comparing Figures 10 and 11, it can be deducted that although repeating the last successfully sent packet would not suffice as a valid masking method, when additional smoothing is added to the ends of the repeated packet the dropped packet is, in fact, masked quite well.

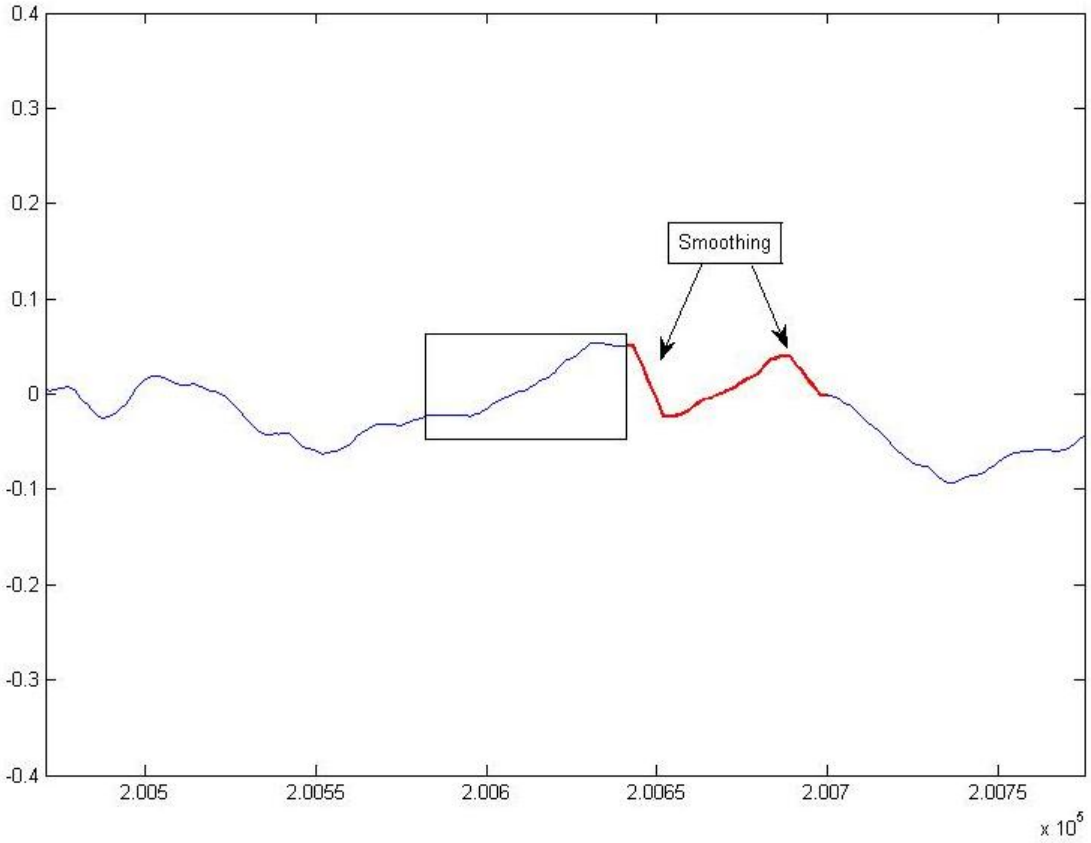

Figure 11: Repeat Making Method and Smoothing

The effectiveness of this smoothing technique can also be shown for the blanking masking method. The first step, as with the repeat masking method, is to view the dropped packet as missing data. This can be seen in Figure 9. Remembering the large pop associated with the break in data, there is a dire need for implementing a masking method to ease the pop for the human ear. The simplest masking method, the blanking masking method, replaces the missing data with 0's. This process can be reviewed in Figure 12 below.

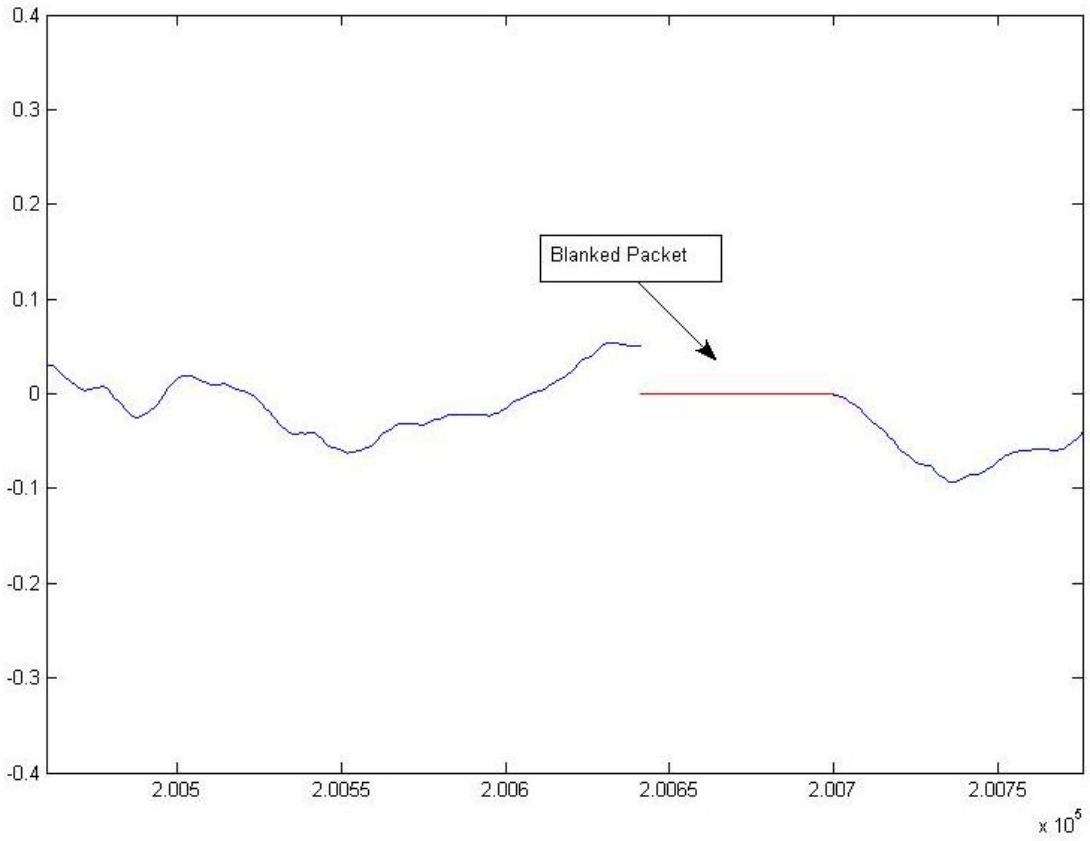

Figure 12: A Blanked Dropped Packet

With very minimal computation, this is already an improvement from missing data completely. However, harsh edges typically occur at both ends of the previously dropped packet. As seen in the Figure 12, the next packet after this particular dropped packet happens to start at 0. Therefore, the pop due to a harsh edge only occurs on the left side of the blanked dropped packet. To reduce this effect, the smoothing technique described before is executed. Figure 13 below shows smoothing on both ends of the blanked packet. By comparing Figures 12 and 13, it can easily be seen that the smoothing technique plays a big role in the effectiveness of the blanking masking method. As with the repeat masking method, simply blanking the dropped packet would

not be enough for a valid masking method. However, when the smoothing is applied, the result of the masking method in Figure 13 looks quite effective.

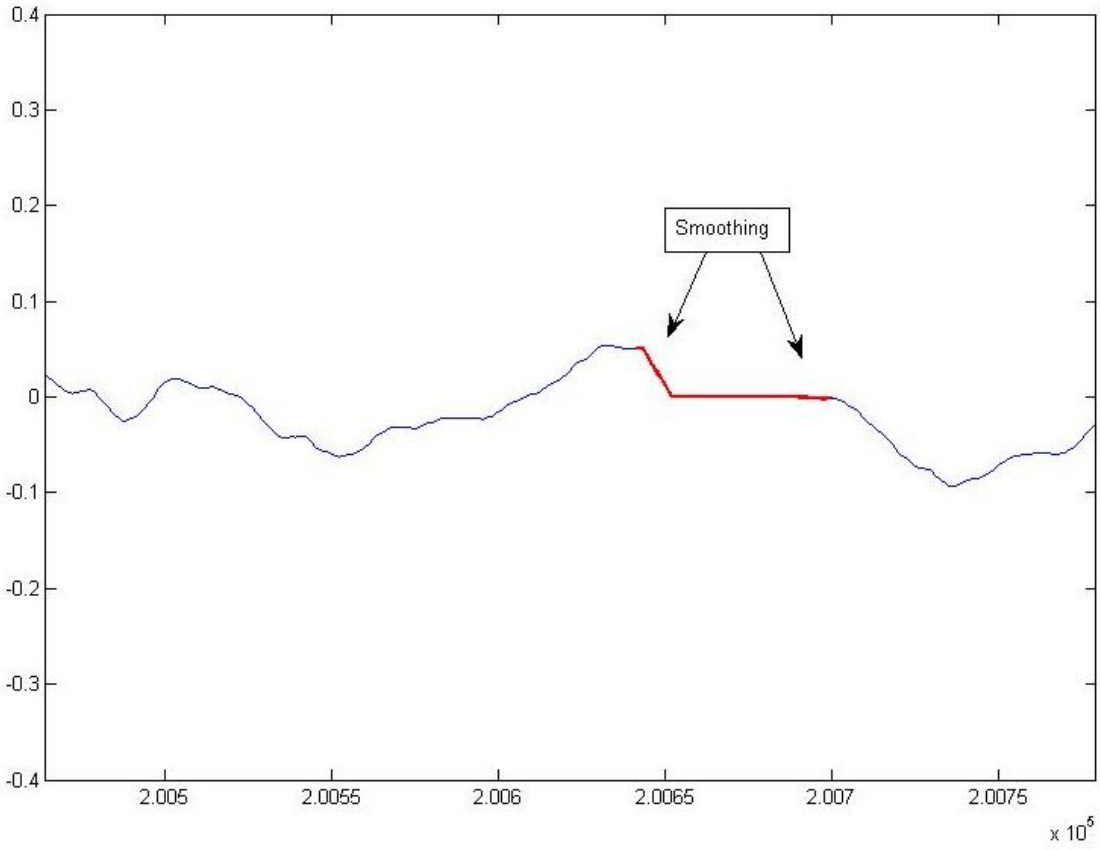

Figure 13: Blanking Masking Method and Smoothing

Data Collection Automation

With testing so many subjects, and each having data for all nine tests, the data to be stored and analyzed quickly amounted to quite a bit. To ease this process, some automation techniques were used. The m-file for the GUI, found in Appendix A, automatically collects the answers from all nine tests. The subject's bubble choice is stored for each test, as well as the subject number.

After the subject is done taking the tests, the packet error rate thresholds found were easily inserted into Microsoft Excel. Through many equations in Excel, the test answers were sorted so that they matched the correct test number. The error rate was also calculated, using the placement of the static set of error rates along with the particular test number. It was then a simple task to see the trends of effectiveness of the methods for the different tests. This also helped in discerning if the subject's test data was erroneous data, which is further described in the Human Subject Testing Data section below.

Format

The format for the GUI was designed in such a way that the subject would easily be able to listen to the sound samples, compare them, and find the packet error rate threshold for each test. To accomplish this layout, the intuitive design of a vertical bubble list and button was used. In this way, the subject could simply click between bubbles to compare sound samples, and easily move to the next test. The Matlab m-file which creates and runs the GUI can be found in Appendix A. Both [8] and [9] were used in order to create such a complex GUI format, which not only runs all nine tests, but also stores the data in a concise way.

Using the GUI

On the first screen of the GUI, the subject number will be chosen from a drop down menu. This will choose the order that the subject will take the nine tests. This Circular Ordering Technique ensures that the data from all subjects is evenly distributed, and is explained further in the section below.

When a bubble is selected by the user, it will play the sound sample with the corresponding packet error rate. For the case where the subject forgets what the goal of the test is, clear and precise instructions are given at the top of each test page. In layman's terms, the goal for the user is to find the lowest bubble for which when clicked plays a sound clip where corruption noise can be heard. The packet error rate threshold will thus be found for each test, showing the highest error rate which can be successfully masked for a certain test.

Once the user has compared the sound samples and found the packet error rate threshold, simply clicking the button labeled 'Next' will prepare the GUI for the following test. Once all nine tests have been completed, the GUI's screen will instruct the subject to notify me that they are done, and thank them for their help in taking the tests for my study.

Test Ordering Technique

When conducting human subject testing, there are important measures that must be taken to ensure true data. There are many factors at play due to the nature of human beings. Having enough data is crucial to conclusive results, therefore enough people to participate in the testing is needed. While taking the nine distinct tests, the subjects will learn how to distinguish noise with more accuracy, and thus better find the packet error rate threshold. Therefore, the last test that they take will be more accurate, with use of their newfound understanding, than the first test that they take. This ability to find the packet error rate threshold better by the end of the session is called a learning effect. Naturally, having this advantage on the same test for every user would not make for the most accurate data.

Another effect that comes into play with human subject testing is fatigue. As with any period of test taking, fatigue can cause the subject to lose concentration through testing. The set of nine tests has been kept brief to combat this. Fatigue is a very natural occurrence during human subject testing and can be due to a number of reasons. The most common of these include lack of sleep, lack of motivation to stay attentive, and distraction.

Not wanting the results to be order dependent, an ordering technique was used to mix up the order of test-taking for the subjects. This technique was inspired by the Latin Squares Method, although the noticeable difference is the circular property of the ordering technique used, thus it will be referred to as the Circular Ordering Technique. The difference between the results of the Latin Squares Method and the Circular Ordering Technique is the robustness in which they counter the fatigue and learning effects already discussed. Although the Circular Ordering Technique is not as robust, it is efficient enough for our human subject testing. It should also be noted that although the Circular Ordering Technique did account for the fatigue and learning effects, it did not test for the order of adjacent tests. In other words, each test had the same two adjacent tests for each of the nine orders. Unlike the Circular Ordering Technique, the Latin Squares Method mixes up the order of the tests in such a way that, for instance, test number two would sometimes be followed by test number five, and other times preceded by test number five.

For the Circular Ordering Technique, every user would obtain a subject number, and the order in which they take the nine distinct tests would change according to their given number. The orders were defined as keeping the order of the nine tests, and simply

changing the starting test for the subject. For example, if the starting test was found to be test three for a particular subject, then the order in which they would take the nine tests would be test numbers three, four, five, six, seven, eight, nine, one, and two. A table showing which subject numbers gave a particular starting test number is given below.

Table 1: Combination Order for Each Subject

Starting Test	Subject Number					
1	8	17	26	35	44	53
2	9	18	27	36	45	54
3	1	10	19	28	37	46
4	2	11	20	29	38	47
5	3	12	21	30	39	48
6	4	13	22	31	40	49
7	5	14	23	32	41	50
8	6	15	24	33	42	51
9	7	16	25	34	43	52

With nine different test order combinations, data for six subjects was kept for each combination. This gives data for a total of fifty-four subjects, excluding the erroneous data, using each of the test orders, using subject numbers one through fifty-four. In this way, the effects of both fatigue and the learning effect would be distributed throughout all of the results, eliminating any bias towards a particular test.

CHAPTER EIGHT

Results and Analysis

Expected Results

Primary Goal

The results of the human subject testing were of great important for the analysis of the properties of the nine tests. The primary goal from collecting the answers was to obtain an average packet error rate threshold for each combination of packet size and masking method. Using this, I would then be able to evaluate the different tests, with the data showing which combination of masking method and packet length masked the largest packet error rate.

Of the three masking methods, the blanking masking method has the least computation involved, and I expected the worst results from it. On the other end, the low-pass blend masking method was computationally intensive. I expected this to serve as an asset to the masking method, and for the effectiveness of the masking method to be above the rest. With this result, I would evaluate the tradeoff between computation, resulting in a delay in the system, and effectiveness of the system, where a better sound quality and longer range would be obtained. The middle masking method, where a dropped packet is replaced with the last successful packet, was expected to be a good balance between computation and result, and thus the best choice between the three masking methods.

The three packet lengths are previously defined in this thesis as the original packet length (180/44.1 ms), a small burst of packets (540/44.1 ms), and a large burst of packets (1620/44.1 ms). When masking simulated dropped packets, it was expected that the larger the burst of dropped packets, the worse the resulting sound sample would sound. This is because a long string of dropped packets will result in interfering noise that is hard to mask due to the nature of the masking techniques. It can then be said that the smallest packet length, defined as a single packet dropped, would produce the best quality in the resulting sound sample. This assumption is due to the fact that a smaller gap in the masking method to fix, the easier it would be to mask it.

Erroneous Data

Although most subjects asked to participate in the human subject testing would be of college age and should still have good hearing intact, I expected some subjects' data to not meet the typical hearing requirements or to be uncooperative. For a worst case scenario, I expected twenty percent of the subjects' data to end up being set aside as erroneous data. Data was determined to be erroneous by simply analyzing the subject's results for all nine tests, looking for indications of hearing loss or lack of attentiveness.

Analysis of Results

Data from the Human Subject Testing

Once the data was collected from all of the subjects, Microsoft Excel was used to organize the data to be used. The excluded data, erroneous data, is discussed further in the section below. The goal of evaluating the data was to determine the quality of masking done by each of the nine distinct pairs of packet length and masking method. To

do this, the average, mode, and median were calculated for each of these pairs. Tables showing these values, broken down by masking method used, are given below.

Table 2: Evaluation for Blanking Masking Method

Analysis	p1	p2	p3
Average:	2.29E-03	2.22E-03	3.29E-04
Mode:	3.00E-03	3.00E-03	3.00E-05
Median:	3.00E-03	3.00E-03	3.00E-05

Table 3: Evaluation for Repeat Masking Method

Analysis	p1	p2	p3
Average:	1.29E-03	2.50E-03	2.56E-03
Mode:	3.00E-04	3.00E-03	3.00E-03
Median:	1.00E-03	3.00E-03	3.00E-03

Table 4: Evaluation for Low-Pass Blend Masking Method

Analysis	p1	p2	p3
Average:	3.97E-04	3.99E-04	2.46E-05
Mode:	3.00E-04	3.00E-05	3.00E-05
Median:	3.00E-04	3.00E-05	3.00E-05

Careful viewing of the above tables does, in fact, determine which pairs effectively masked the simulated dropped packets better than the rest. It should first be noted that the mode and median are close in number to the average for most of the nine

pairs. This being said, the average answer is also the most common answer during the human subject testing, and the shape of the distribution is as expected. This is further examined in the statistics analysis section below. Remembering that higher numbers are better, meaning that it took a higher dropped packet rate to hear noise, the tables can then be deciphered.

When categorizing the results by packet length, it can be seen that for the smallest packet length, p1, the blanking masking method works the best. However, for the medium and large packet lengths, p2 and p3, respectively, the repeat masking method works the best. Therefore, there is not one masking method that clearly works more effectively than the rest for all three packet lengths.

It was also found that by placing many small dropped packets close together the interfering noise was more noticeable than when a single, large dropped packet occurred. In other words, several single dropped packets occurring in a close cluster proved to produce more unwanted noise in the sound sample than a burst of many dropped packets in a row. This discovery could prove to be an interesting subject for future research in this field.

Another interesting find was that the ordering technique used to counter fatigue and learning effects was, in fact, not needed. The noise heard in the nine different tests could be uniquely heard, no matter when it was heard. Therefore, the consideration taken for the ordering of the tests was a good precaution to take, even though in the end it was not needed for this particular human subject testing. It should, however, be kept as an important factor of this project, for most projects would show great biases due to effects such as fatigue and the learning effect.

Erroneous Data

Some subjects' data from the tests was classified as erroneous data. This could be due to anything from lack of attention to a good deal of hearing loss. In some cases, the packet error rate threshold was consistently chosen where there was no error in the sound samples. In others, they could not hear the noise from the dropped packets and therefore chose the maximum error rate for the tests. In cases with data which was obviously erroneous, the data was simply set aside. For instance, this could be consistently choosing the highest possible packet error rate threshold, indicating a large amount of hearing loss, or choosing the packet error rate threshold to be in a position where no packets were dropped, typically due to lack of attentiveness.

Eighteen percent of the subjects that completed the GUI for human subject testing were classified as erroneous data. This was below the estimated twenty percent, yet very close to it. This data was not used in the final analysis. Another subject took the tests with the same subject number to replace the erroneous data for analysis purposes. Therefore, the erroneous data was accounted for whilst aiming for a particular number of subjects' data for each testing order due to the Circular Order Technique constraints, described previously.

Statistical Analysis of Results

To further analyze the data from the human subject testing, statistical models were applied. Dr. Jack Tubbs, the chair of Baylor University's Department of Statistics, analyzed the data from the human subject tests using SAS.

Some of the results of the statistical analysis are discussed below, whereas the complete statistical analysis results from the human subject testing are shown in

Appendix K. Figure 14 shows a histogram of the response data by masking method distribution, whereas Figure 15 shows a histogram of the response data by packet length distribution. Note that the estimated distributions do not follow the curve of a normal distribution; therefore the data cannot be considered normal. It can be seen, for instance, which masking method and packet length combination work most effectively. When evaluating the histograms, it is important to define the axes. The bottom axis represents the packet error rates, whereas the left axis shows how many subjects chose that particular packet error rate. What we are looking for here is for the largest bar of a particular histogram to be on the right. This would tell us that for most subjects taking that particular test, it took a higher packet error rate to hear interfering noise. Thus, that test combination would more successfully mask dropped packets.

By evaluating the histogram in Figure 14, we can see that method 2, the repeat masking method, is the most effective masking method of the three. It can also be seen that method 3, the low-pass blend masking method, has a histogram quite the opposite of what we would like to see. This tells us that for most subjects, they could still hear interfering noise when the packet error rate was small, indicating that this masking method did not work as well as expected. We anticipated the low-pass blend masking method to be the most effective of the three, although at the cost of computational delay. However, even with higher computational costs, the method did not compare favorably to the repeat masking method.

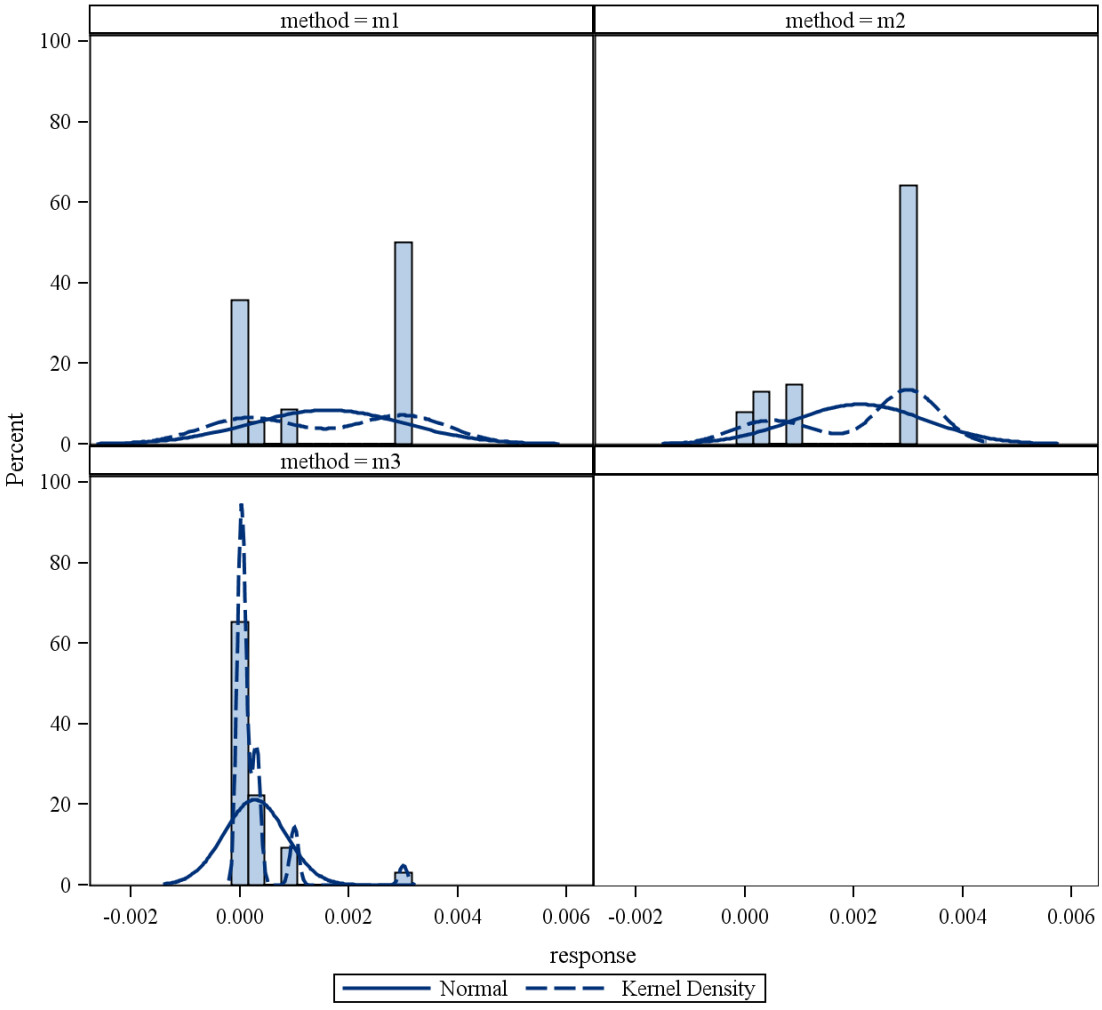

Figure 14: Histogram of the Response Data by Method Distribution

To evaluate the effectiveness of the packet lengths used, Figure 15 can be evaluated in the same manner as Figure 14. Here we are still looking for the highest bar to be on the right side of the graph, with the definitions of the axes remaining the packet error rates and number of subjects as before. With these criteria in mind, we notice that the histogram for the third packet length, defined as the largest packet length, on the bottom left has a tall bar on the left side of the graph. This tells us that this large packet length did not help in masking the dropped packets, but rather made noise more apparent.

This analysis makes sense logically since a large string of dropped packets would be much harder to mask than a short burst of them. Now bringing attention to the histogram of the second packet length, defined as the medium packet length, on the top right we see the shape that we are looking for. Therefore, the medium packet length is the best length when evaluating the effectiveness of the masking techniques. This comes as somewhat of a surprise, for the smaller packet length was expected to show better results. Thus, it is not the single dropped packet that is the easiest for the masking methods to handle, but a small burst of dropped packets. We conjecture that is in part due to the smoothing of the edges of the masked packet.

We now consider another type of analysis performed by Dr. Tubbs. Figure 16 shows this histogram of both the masking methods and the packet lengths. The benefit of this graph is the ability to see which masking methods work the best for each packet length. The left axis represents the packet error rates, where the bottom axis represents the three different packet lengths used.

Looking above p1, the smallest packet length, we can see that the blue line, representing m1, the blanking masking method, corresponds to the highest packet error rate. This tells us that for the smallest packet length, the blanking masking method results in the highest sound quality. Likewise, for p2, the medium packet length, we see the blue and red dashed lines, blanking and repeat masking methods, respectively, compete for the best masking method, with the repeat masking method showing slightly better results. Finally, on the far right we see for p3, the largest packet length, the repeat masking method works the best. Here both the blanking and low-pass blend masking methods are far below the repeat masking method.

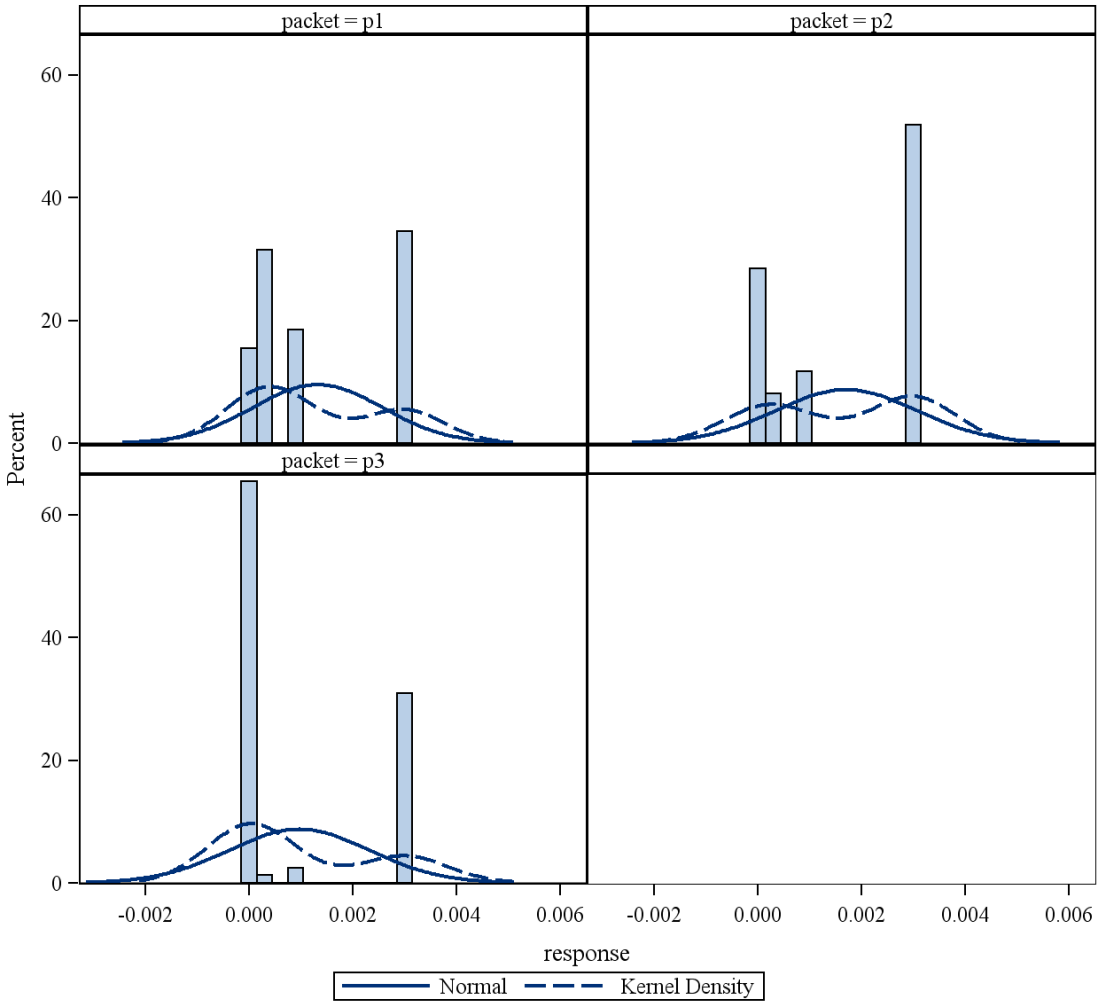

Figure 15: Histogram of the Response Data by Packet Distribution

It should be noted that the low-pass blend masking method is far below the top-performing masking method for all three packet lengths. This confirms the previous finding that the low-pass blend masking method does not work as first expected. Instead, its results are far below the other two masking methods.

Overall, it can be said that the red dashed line is the highest on the graph, on average. This tells us that the repeat masking method gives better sound quality during masking, on average, for the three tests where it is implemented. This is in concurrence

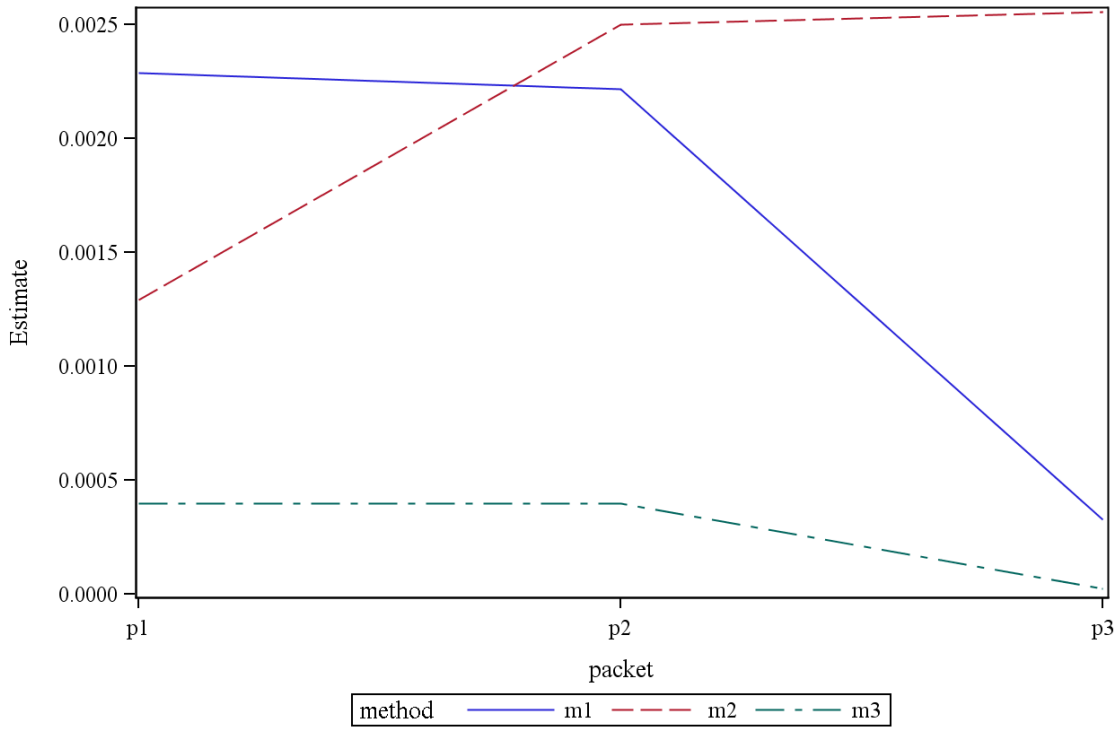

Figure 16: Plot of the Response Data for Methods and Packet Lengths

with the analysis of the previous graphs, confirming the repeat masking method as the best masking method when choosing one particular masking method for all three packet lengths.

Conclusion

Findings

After reviewing the analyses described above, we can see that there are multiple findings from this research project. For a single, short dropped packet (180/44.1 ms), the blanking masking method was the most effective when comparing sound quality of the resulting sound samples. For burst of dropped packets (540/44.1 ms and 1620/44.1 ms),

however, the repeat masking method was much more successful at creating a sound sample with superior sound quality.

There were two results found that I had not expected. Initially, a small burst of dropped packets was masked better than a single dropped packet. This can be logically explained due to the nature of the masking methods developed, particularly their smoothing property. Although the smoothing technique used generally improved the sound quality of the sound sample, when a single dropped packet occurs there is simply not enough time for both masking and smoothing to take effect. Therefore, the sound quality, when compared to the masking of a small burst, is inferior.

The second surprising find was the value of the low-pass blend masking method. Presented as the most complex of the three masking methods developed, it was expected to have a far better quality of sound sample resulting when evaluated against the results of the other two masking methods. However, it was found that although much more computation was executed in the low-pass blend masking method, its resulting sound sample was of much lesser quality than the rest. This came as somewhat of a disappointment, but an interesting discovery nonetheless.

The main deduction from this research is that for a typical digital wireless communication system of today's day and age, the repeat masking method is the best choice. This judgment is for the condition where a system is being designed for maximum sound quality as a whole, not only for single dropped packets, but short and long bursts of dropped packets as well.

Applications

The digital wireless communication systems of today can greatly benefit from my research. There are many products for which my research can be applied, not only in wireless instruments and microphones, but also in-ear monitors and wireless speaker systems. Not only is there a variety of products for which my research is applicable, but there are also many venues. Intimate venues, such as a musician practicing at home or an at-home karaoke machine; houses of worship, where the entire band could be using wireless systems and large venues, such as a concert at a stadium, can all benefit from the findings of my research. With the many products and venues for which my research can be implemented, it can easily be seen that my findings are a great contribution to the digital wireless communication systems of today.

APPENDICES

APPENDIX A

Matlab Code for Creating and Running the GUI

```
function varargout = soundtest(varargin)
% SOUNDTEST M-file for soundtest.fig
%      SOUNDTEST, by itself, creates a new SOUNDTEST or raises the existing
%      singleton*.
%      H = SOUNDTEST returns the handle to a new SOUNDTEST or the handle to
%      the existing singleton*.
%      SOUNDTEST('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SOUNDTEST.M with the given input arguments.
%      SOUNDTEST('Property','Value',...) creates a new SOUNDTEST or raises the
%      existing singleton*.

% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @soundtest_OpeningFcn, ...
                   'gui_OutputFcn',  @soundtest_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code

% --- Executes just before soundtest is made visible.
function soundtest_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args

% Choose default command line output for soundtest
handles.output = hObject;
```

```
%initializes the test number to load and the number of tests completed
handles.testnumber=0;
handles.numtest=0;
 %this initializes the subject number (k) as 1
handles.k=1;

%this initializes the bubble answers
handles.bubbles = [];

%this initializes the radio button choice
handles.rdbtn=[];

%This is for when you click the different buttons
set(handles.clips_buttongroup,'SelectionChangeFcn',@clips_buttongroup_SelectionChan
geFcn);

% Update handles structure
guidata(hObject, handles);

% --- Executes on selection change in subNums.
function subNums_Callback(hObject, eventdata, handles)
%       contents{get(hObject,'Value')} returns selected item from subNums

%this grabs a copy of my handles
handles = guidata(hObject);

%this will read in the subject number the user has selected
handles.k = get(hObject,'Value');

%this updates my handles
guidata(hObject,handles);

% --- Executes on button press in btnNext.
function btnNext_Callback(hObject, eventdata, handles)

%this will stop the sound so that the GUI can change
clear playsnd

%this grabs a copy of my handles
handles = guidata(hObject);

%output.bubbles will be the bubble answers
handles.bubbles = [handles.bubbles handles.rdbtn];

%this will start the first test
if handles.numtest == 0
```

```
    %this initializes the button choice in case they don't change bubbles
    handles.rdbtn=1;

    %this will make the subject number list and text invisible
    set(handles.subNums, 'Visible', 'off');
    set(handles.txtSubNum, 'Visible', 'off');

    %all of the testing elements will now be made visible
    set(handles.txtInstruct, 'Visible', 'on');
    set(handles.clips_buttongroup, 'Visible', 'on');
    set(handles.txtHigh, 'Visible', 'on');
    set(handles.txtLow, 'Visible', 'on');
    set(handles.txtNum,'string','Test 1 of 9');
    set(handles.txtNum, 'Visible', 'on');
end

%this will only run if there are still tests to run
if handles.numtest < 9

    %this increments the current test number
    handles.numtest = handles.numtest+1;

    %this sets the test number to load and run
    %here there are 9 different tests (test0-test8)
    handles.testnumber = mod(handles.k+handles.numtest,9);

    %txtNum will display the current test number
    tnum=num2str(handles.numtest);
    currentTest =['Test ' tnum ' of 9'];
    set(handles.txtNum,'string',currentTest);

    %this resets the selection to the last bubble
    handles.rdbtn=12;
    set(handles.rdbtn12,'Value',1);

    %this will load the next test's sound clips (y's and Fs's)
    tload=num2str(handles.testnumber);
    soundClips =['test' tload];
    load(soundClips)

    %this update the y's and Fs's
    handles.y1=y1;
    handles.y2=y2;
    handles.y3=y3;
    handles.y4=y4;
    handles.y5=y5;
```

```
    handles.y6=y6;
    handles.y7=y7;
    handles.y8=y8;
    handles.y9=y9;
    handles.y10=y10;
    handles.y11=y11;
    handles.y12=y12;
    handles.Fs1=Fs1;
    handles.Fs2=Fs2;
    handles.Fs3=Fs3;
    handles.Fs4=Fs4;
    handles.Fs5=Fs5;
    handles.Fs6=Fs6;
    handles.Fs7=Fs7;
    handles.Fs8=Fs8;
    handles.Fs9=Fs9;
    handles.Fs10=Fs10;
    handles.Fs11=Fs11;
    handles.Fs12=Fs12;

    %this updates my handles
    guidata(hObject,handles);
else

    %all of the testing elements will now be made invisible
    set(handles.txtInstruct, 'Visible', 'off');
    set(handles.clips_buttongroup, 'Visible', 'off');
    set(handles.txtHigh, 'Visible', 'off');
    set(handles.txtLow, 'Visible', 'off');
    set(handles.txtNum, 'Visible', 'off');
    set(handles.btnNext, 'Visible', 'off');

    %this is the closing statement to the user, now visible
    set(handles.txtClosing, 'Visible', 'on');

    %this is the final result for subject number and bubble selections
    varargout{1}.bubbles=handles.bubbles;
    varargout{1}.k=handles.k;
    handles.output1=varargout;

    %this updates my handles
    guidata(hObject,handles);

    %this resumes the output
    uiresume;
end
```

```
 % this function will run for when the different buttons are pressed
function clips_buttongroup_SelectionChangeFcn(hObject, eventdata)

%this will stop whatever is playing so that something else can play now
clear playsnd

%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
   case 'rdbtn1'
     %this keeps record of which button was played
     handles.rdbtn = 1;
     %execute this code when rdbtn1 is selected
     sound(handles.y1,handles.Fs1);

   case 'rdbtn2'
     %this keeps record of which button was played
     handles.rdbtn = 2;
     %execute this code when rdbtn2 is selected
     sound(handles.y2,handles.Fs2);

   case 'rdbtn3'
     %this keeps record of which button was played
     handles.rdbtn = 3;
     %execute this code when rdbtn3 is selected
     sound(handles.y3,handles.Fs3);

   case 'rdbtn4'
     %this keeps record of which button was played
     handles.rdbtn = 4;
     %execute this code when rdbtn4 is selected
     sound(handles.y4,handles.Fs4);

   case 'rdbtn5'
     %this keeps record of which button was played
     handles.rdbtn = 5;
     %execute this code when rdbtn5 is selected
     sound(handles.y5,handles.Fs5);

   case 'rdbtn6'
     %this keeps record of which button was played
     handles.rdbtn = 6;
     %execute this code when rdbtn6 is selected
     sound(handles.y6,handles.Fs6);
```

```
    case 'rdbtn7'
      %this keeps record of which button was played
      handles.rdbtn = 7;
      %execute this code when rdbtn7 is selected
      sound(handles.y7,handles.Fs7);

    case 'rdbtn8'
      %this keeps record of which button was played
      handles.rdbtn = 8;
      %execute this code when rdbtn8 is selected
      sound(handles.y8,handles.Fs8);

    case 'rdbtn9'
      %this keeps record of which button was played
      handles.rdbtn = 9;
      %execute this code when rdbtn9 is selected
      sound(handles.y9,handles.Fs9);

    case 'rdbtn10'
      %this keeps record of which button was played
      handles.rdbtn = 10;
      %execute this code when rdbtn10 is selected
      sound(handles.y10,handles.Fs10);

    case 'rdbtn11'
      %this keeps record of which button was played
      handles.rdbtn = 11;
      %execute this code when rdbtn11 is selected
      sound(handles.y11,handles.Fs11);

    case 'rdbtn12'
      %this keeps record of which button was played
      handles.rdbtn = 12;
      %execute this code when rdbtn12 is selected
      sound(handles.y12,handles.Fs12);

    otherwise
       %the button pressed stored will be the last button pressed still
end
%updates the handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = soundtest_OutputFcn(hObject, eventdata, handles)
```

```
%this will make the output function wait for the data
uiwait;

%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

varargout=handles.output1;
```

APPENDIX B

Matlab Code for Creating Sound Clips for All Nine Tests

```
function make_clip_params
%this function makes sound clips by calling make_clip with parameters
%three types of masking (blanking, repeat, spectral)
%three lengths of packets
%there are 9 tests, each test has 12 bubble choices
%5 particular fixed error rates for sound clips that will shift
%use:
%   make_clip_params();
%inputs:
%   NONE
%outputs:
%   NONE, creates .mat files for test cases 0-8

%% Intiaializations

%length of the packet (*24)
plbytes=[180 540 1620];

%this loads dropall - the 3 lists of error locations
load droparrays

%this randomly decides how many 'blanks' are at the bottom of each test
% for i=1:9
% numBlank(i)=randi(7);
% end
numBlank=[5 3 6 2 2 5 7 6 5];

%these are the different error rates used to generate the sound clips
err_rates=[.3e-2 1e-3 .3e-3 1e-4 .3e-4];

%this will be which test out of 9
testnum=0;

%% Calculations

%type=0 -> blanking (test0,test1,test2)
%type=1 -> repeat(test3,test4,test5)
%type=2 -> Low Pass/Interpolated Blend(test6,test7,test8)
```

```
 %this will generate sound clips for the first and second masking methods
for type=0:1
   %this runs for all 3 packet sizes
   for tnum=1:3

      %this will be the list of dropped packet locations to use
      currentdrop=dropall(tnum,:);

      %length of the packet
      payloadbytes=plbytes(tnum);

      %these are used for the loops
      %tells how many no-error clips will be at the bottom
      temp1=12-numBlank(tnum+3*type);

      %this sets the rest (top) to the same as the most error packet
      probdrop=err_rates(1);
      for i=1:temp1-5
         %this creates the 'names' for the bubble
         inum=num2str(i);
         currentY=['y' inum];
         currentFs=['Fs' inum];
         currentErrLoc=['errorlocations' inum];
         %this sets the values for the bubble
         eval(['[' currentY ' ' currentFs ' ' currentErrLoc ']' ' = make_clip(payloadbytes,
            probdrop, type, currentdrop);']);
      end

      %this sets the static chunk in place
      k=1;
      for i=temp1-4:temp1
         %this creates the 'names' for the bubble
         inum=num2str(i);
         currentY=['y' inum];
         currentFs=['Fs' inum];
         currentErrLoc=['errorlocations' inum];
         %this sets the drop rate
         probdrop=err_rates(k);
         k=k+1;
         %this sets the values for the bubble
         eval(['[' currentY ' ' currentFs ' ' currentErrLoc ']' ' = make_clip(payloadbytes,
            probdrop, type, currentdrop);']);
      end

      %this sets the bottom numBlank(tnum) values with 0 drop rate
      probdrop=0;
```

```
        for i=temp1+1:12
            %this creates the 'names' for the bubble
            inum=num2str(i);
            currentY=['y' inum];
            currentFs=['Fs' inum];
            currentErrLoc=['errorlocations' inum];
            %this sets the values for the bubble
            eval(['[' currentY ' ' currentFs ' ' currentErrLoc ']' ' = make_clip(payloadbytes,
                probdrop, type, currentdrop);']);
        end
        %this will save the data for each test
        tempName=num2str(testnum);
        currentName=['test' tempName];
        eval(['save ' currentName ' y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 Fs1 Fs2 Fs3 Fs4
                Fs5 Fs6 Fs7 Fs8 Fs9 Fs10 Fs11 Fs12 errorlocations1 errorlocations2
                errorlocations3 errorlocations4 errorlocations5 errorlocations6 errorlocations7
                errorlocations8 errorlocations9 errorlocations10 errorlocations11
                errorlocations12']);
        testnum=testnum+1;
    end
end

%this will generate sound clips for the third masking methods
type=2;
for tnum=1:3
    %this will be the list of dropped packet locations to use
    currentdrop=dropall(tnum,:);

    %length of the packet
    payloadbytes=plbytes(tnum);

    %these are used for the loops; tells how many no-error clips will be at the bottom
    temp1=12-numBlank(tnum+3*type);

    %this sets the rest (top) to the same as the most error packet
    probdrop=err_rates(1);
    for i=1:temp1-5
        %this creates the 'names' for the bubble
        inum=num2str(i);
        currentY=['y' inum];
        currentFs=['Fs' inum];
        currentErrLoc=['errorlocations' inum];
        %this sets the values for the bubble
        eval(['[' currentY ' ' currentFs ' ' currentErrLoc ']' ' = make_blend(payloadbytes,
                probdrop, currentdrop);']);
    end
```

```
    %this sets the static chunk in place
    k=1;
    for i=temp1-4:temp1
        %this creates the 'names' for the bubble
        inum=num2str(i);
        currentY=['y' inum];
        currentFs=['Fs' inum];
        currentErrLoc=['errorlocations' inum];
        %this sets the drop rate
        probdrop=err_rates(k);
        k=k+1;
        %this sets the values for the bubble
        eval(['[' currentY ' ' currentFs ' ' currentErrLoc ']' ' = make_blend(payloadbytes,
                probdrop, currentdrop);']);
    end

    %this sets the bottom numBlank(tnum) values with 0 drop rate
    probdrop=0;
    for i=temp1+1:12
        %this creates the 'names' for the bubble
        inum=num2str(i);
        currentY=['y' inum];
        currentFs=['Fs' inum];
        currentErrLoc=['errorlocations' inum];
        %this sets the values for the bubble
                    eval(['[' currentY ' ' currentFs ' ' currentErrLoc ']' ' =
make_blend(payloadbytes, probdrop, currentdrop);']);
    end

    %this will save the data for each test
    tempName=num2str(testnum);
    currentName=['test' tempName];
    eval(['save ' currentName ' y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 Fs1 Fs2 Fs3 Fs4
            Fs5 Fs6 Fs7 Fs8 Fs9 Fs10 Fs11 Fs12 errorlocations1 errorlocations2
            errorlocations3 errorlocations4 errorlocations5 errorlocations6 errorlocations7
            errorlocations8 errorlocations9 errorlocations10 errorlocations11
            errorlocations12']);
    testnum=testnum+1;
end

end
```

APPENDIX C

Matlab Code for Creating Sound Clips for Blanking and Repeat Masking Methods

```
function [y Fs errorlocations]=make_clip(payloadbytes, probdrop, type, droparray)
%this function will give an output (y) of a sound clip
%%THIS USES A 3-PART LINEAR CONNECTION
%use: [y Fs]=make_clip(payloadbytes, probdrop, type, droparray)
%outputs:
% y - seven second sound clip
% Fs - sampling rate returned from the wavread call
% errorlocations - array of locations where dropped packets occur % <-
%   could put back in
%inputs:
% payloadbytes - num of bytes in payload, min of 128
% probdrop - probability of a packet being dropped (changes packet size)
% type - what type of masking will be used for the dropped packets
%   0=blanking, 1=repeat
% droparray - this is the list of error locations to use

%% Initializations

%y will be the clip where packets are dropped
[y Fs]=wavread('Allegro.wav');
y=.5*y(10*Fs:20*Fs);

%converts payloadsize (in bytes) to samples
packsize=floor(payloadbytes/3);

%m divides the sound clip into chunks depending on the number of samples
m=floor(length(y)/(packsize));

%this is how many dropped packets there will be
dropnum=ceil(m*probdrop);

%this is the list of locations to use for dropped packets
errloclist=droparray(1:dropnum);

%mblank will tell where the packets were dropped
errorlocations=[];
```

```
%this is the part of the current/adjacent blocks for linear connections
B=15;

%this is the length of the horizontal linear portions
Bsides=3;
%this is the length of the sloped linear connection
Bmid=B-Bsides*2;

%% Calculations

%here blanking is implemented
if type == 0

    %this is what will replace the dropped packet
    replacements = 0;

    %this is the process to drop the packets in the sound clip
    for k=1:m

        if sum(ismember(errloclist,k)) == 1
            %when a packet is dropped, the entire packet is replaced with
            %zeros, blanking the packet
            y((k-1)*packsize+1:k*packsize)=replacements;

            %each time a packet is dropped, mblank will hold the location
            errorlocations=[errorlocations k];

            %this grabs the last point of the last packet
            left_temp=y(((k-1)-1)*packsize+1:(k-1)*packsize);
            leftend=left_temp(end);

            %this grabs a point of the current/dropped packet B from front
            curr_temp=y((k-1)*packsize+1:k*packsize);
            frontB=curr_temp(B);
            %this grabs a point of the current/dropped packet B from back
            backB=curr_temp(end-B+1);

            %this grabs the first point of the next packet (assumed right)
            right_temp=y(((k+1)-1)*packsize+1:(k+1)*packsize);
            rightfront=right_temp(1);

            %this creates linear connections for the front end (lcf's)
            %this is the sloped/middle connection
            n=1:Bmid-1;
            lcf_mid=leftend+((n/Bmid)*(frontB-leftend));
            %this is the front left horizontal piece
```

```
            lcf_left=[];
            for i=1:Bsides
               lcf_left=[lcf_left leftend];
            end
            %this is the front right horizontal piece
            lcf_right=[];
            for i=1:Bsides
               lcf_right=[lcf_right frontB];
            end
            %this puts the three front linear connections together
            lcf=[lcf_left lcf_mid lcf_right];

            %this creates a linear connections for the back end (lcb's)
            %this is the sloped/middle connection
            lcb_mid=backB+((n/Bmid)*(rightfront - backB));
            %this is the back left horizontal piece
            lcb_left=[];
            for i=1:Bsides
               lcb_left=[lcb_left backB];
            end
            %this is the back right horizontal piece
            lcb_right=[];
            for i=1:Bsides
               lcb_right=[lcb_right rightfront];
            end
            %this puts the three back linear connections together
            lcb=[lcb_left lcb_mid lcb_right];

            %this puts the linear connections into the current packet
            %there are B values 'created' with the linear connections
            ynew=[lcf curr_temp(B:end-B+1) lcb];
            %this places the new current packet into place
            y((k-1)*packsize+1:k*packsize)=ynew;
         end
      end

%here repeat masking is implemented
elseif type == 1

   %this is what will replace the dropped packet
   replacements = 0;

   %this is the process to drop the packets in the sound clip
   for k=1:m

      if sum(ismember(errloclist,k)) == 1
```

```
%when a packet is dropped, the entire packet is replaced with
%the last packet successfully sent
y((k-1)*packsize+1:k*packsize)=replacements;

%each time a packet is dropped, mblank will hold the location
errorlocations=[errorlocations k];

%this grabs the last point of the last packet
left_temp=y(((k-1)-1)*packsize+1:(k-1)*packsize);
leftend=left_temp(end);

%this grabs a point of the current/dropped packet B from front
curr_temp=y((k-1)*packsize+1:k*packsize);
frontB=curr_temp(B);
%this grabs a point of the current/dropped packet B from back
backB=curr_temp(end-B+1);

%this grabs the first point of the next packet (assumed right)
right_temp=y(((k+1)-1)*packsize+1:(k+1)*packsize);
rightfront=right_temp(1);

%this creates linear connections for the front end (lcf's)
%this is the sloped/middle connection
n=1:Bmid-1;
lcf_mid=leftend+((n/Bmid)*(frontB-leftend));
%this is the front left horizontal piece
lcf_left=[];
for i=1:Bsides
   lcf_left=[lcf_left leftend];
end
%this is the front right horizontal piece
lcf_right=[];
for i=1:Bsides
   lcf_right=[lcf_right frontB];
end
%this puts the three front linear connections together
lcf=[lcf_left lcf_mid lcf_right];

%this creates a linear connections for the back end (lcb's)
%this is the sloped/middle connection
lcb_mid=backB+((n/Bmid)*(rightfront - backB));
%this is the back left horizontal piece
lcb_left=[];
for i=1:Bsides
   lcb_left=[lcb_left backB];
end
```

```
            %this is the back right horizontal piece
            lcb_right=[];
            for i=1:Bsides
               lcb_right=[lcb_right rightfront];
            end
            %this puts the three back linear connections together
            lcb=[lcb_left lcb_mid lcb_right];

            %this puts the linear connections into the current packet
            %there are B values 'created' with the linear connections
            ynew=[lcf curr_temp(B:end-B+1) lcb];
            %this places the new current packet into place
            y((k-1)*packsize+1:k*packsize)=ynew;
        end

        %this holds onto the last packet successfully sent
        replacements = y((k-1)*packsize+1:k*packsize);
    end
end

end
```

APPENDIX D

Matlab Code for Creating Sound Clips for Low-Pass Blend Masking

```
function [y Fs errorlocations]=make_blend(payloadbytes, probdrop, droparray)
%this function will give an output (y) of a sound clip
%use: [y Fs]=make_blend(payloadbytes, probdrop, droparray)
%outputs:
% y - seven second sound clip
% Fs - sampling rate returned from the wavread call
% errorlocations - array of locations where dropped packets occur % <-
%   could put back in
%inputs:
% payloadbytes - num of bytes in payload, min of 128
% probdrop - probability of a packet being dropped (changes packet size)
% droparray - this is the list of error locations to use

%% Initializations

%yorig will be a portion of the wav file
%y will be the clip where packets are dropped
[y Fs]=wavread('Allegro.wav');
y=.5*y(10*Fs:20*Fs);

%converts payloadsize (in bytes) to samples
packsize=floor(payloadbytes/3);

%m divides the sound clip into chunks depending on numsamples
m=floor(length(y)/(packsize));

%this is how many dropped packets there will be
dropnum=ceil(m*probdrop);

%this is the list of locations to use for dropped packets
errloclist=droparray(1:dropnum);

%mblank will tell where the packets were dropped
errorlocations=[];

%% Calculations
%this is what will replace the dropped packet
replacements = 0;
```

```
%this will indicate a packet was just dropped
drop_flag=0;

%this is the process to drop the packets in the sound clip
for k=1:m

   %if the last packet was dropped and the current packet is dropped, the
   %  blending of the end of the last packet/front of the current packet
   %  is automatically taken care of
   if sum(ismember(errloclist,k)) == 1
      %when a packet is dropped, the entire packet is replaced with
      %the last packet successfully sent
      y((k-1)*packsize+1:k*packsize)=replacements;

      %this will indicate the packet was dropped
      drop_flag=1;

      %each time a packet is dropped, mblank will hold the location
      errorlocations=[errorlocations k];

      %this holds onto the last packet sent
      lastpac=y(((k-1)-1)*packsize+1:(k-1)*packsize);

      %this holds onto the current packet
      currpac=y((k-1)*packsize+1:k*packsize);

      %this send the last and current packets into a sub-routine that
      %  does low pass/interpolated blending on them
      [lastpac currpac] = blend(lastpac,currpac);

      %this replaces the last and current packets with the blended ones
      y(((k-1)-1)*packsize+1:(k-1)*packsize)=lastpac;
      y((k-1)*packsize+1:k*packsize)=currpac;

   %if the current packet wasn't dropped but the last packet was, blending
   %  of the end of the last packet/front of the current packet is done
   elseif drop_flag

      %this resets the flag indicator
      drop_flag=0;

      %this holds onto the last packet sent
      lastpac=y(((k-1)-1)*packsize+1:(k-1)*packsize);

      %this holds onto the current packet
      currpac=y((k-1)*packsize+1:k*packsize);
```

```
        %this send the last and current packets into a sub-routine that
        %  does low pass/interpolated blending on them
        [lastpac currpac] = blend(lastpac,currpac);

        %this replaces the last and current packets with the blended ones
        y(((k-1)-1)*packsize+1:(k-1)*packsize)=lastpac;
        y((k-1)*packsize+1:k*packsize)=currpac;
    end

    %this holds onto the last packet successfully sent
    replacements = y((k-1)*packsize+1:k*packsize);
end

end
```

APPENDIX E

Matlab Code for Implementing Low-Pass Blend Masking

```
function [packetout1 packetout2] = blend(packetin1,packetin2)
%this function will give 2 outputs, the current and next packets
%use: [packetout1 packetout2] = blend(packetin1,packetin2)
%outputs:
% packetout1 - the last packet, with the end blended
% packetout2 - the current packet, with the front blended
%inputs:
% packetin1 - the last packet sent
% packetin2 - the current packet, which was dropped

%% Initializations

%flips the packets for calculation purposes
packetin1=packetin1';
packetin2=packetin2';

%these are the lengths of the input packets (last and current)
paclen1=length(packetin1);
paclen2=length(packetin2);

%if for some reason the packet lengths are different, an error occurs
if paclen1~=paclen2
   error('packets must be the same length');
end

%Npts will be blended on the back of p1 and the front of p2 (Npts*2 total)
%the variance in Npts is needed for the different packet sizes
Npts=floor(.04*paclen1);

%there are Dfactor points making every point in Npts
Dfactor=4;

%% Calculations

%this will indicate the end portion of packetin1
n1=paclen1:-Dfactor:paclen1-Npts*(Dfactor)+1;
n1=fliplr(n1);
```

```
%this is the end portion of packetin1 to blend
p1end=packetin1(n1);

%this will indicate the front portion of packetin2
n2=Dfactor:Dfactor:Npts*(Dfactor)+1;
%this is the beginning portion of packetin2 to blend
p2beg=packetin2(n2);

%x is the section to be blended (end of packet1, front of packet2)
x=[p1end' p2beg'];

%this does FFT interpolation of x
y=interpft(x,Dfactor*Npts*2);

%this replaces the original packet1 with the new packet, the end blended
packetout1=packetin1';
%the last Npts*Dfactor-3 pts of the last packet are replaced w/ blended y
%  the last 3 points of the blended y section are not used
packetout1(paclen1-(Npts*Dfactor-3-1):paclen1)=y(1:Npts*Dfactor-3);

%this replaces the original packet2 with the new packet, the front blended
packetout2=packetin2';
%the first Npts*Dfactor pts of the current packet are replaced w/ blended y
%  the last 3 points of the blended y section are not used
packetout2(1:Npts*Dfactor)=y((end-3)-(Npts*Dfactor-1):end-3);

end
```

APPENDIX F

Matlab Code for Calculating Locations of Simulated Dropped Packets

```
function droparray=drop_pac_locs(m)
%this function will generate an array of locations for dropped packets
%use:
%   droparray=drop_pac_locs(m);
%inputs:
%   m - the number of packets in the sound clip
%outputs:
%   droparray - the array of locations for dropped packets, use in order

%% Initializations

%this is the maximum number of dropped packets that will be used
dropmax=50;

%this will be the array of locations for the dropped packets
droparray=zeros(1,dropmax);

%% Calculations

%this will randomly grab packet locations
tempnum=0;
while tempnum < dropmax

   %this will grab a random location
   temploc=randi(m-1,1)+1;

   %this will make sure the location hasn't already been chosen
   if sum(ismember(droparray,temploc)) == 0

      %this increments tempnum for the number of locs used
      tempnum=tempnum+1;

      %this grabs the location for the dropped packet
      droparray(tempnum)=temploc;
   end
end

end
```

APPENDIX G

Matlab Code for Encoding Using the Hamming (7, 4) Code

```
codewords=[6 4 6 1 12 14];

codewords=uint8(codewords);

for j=1:length(codewords)
   temp=codewords(j);
   da=0;
   db=0;
   dc=0;
   dap=0;
   dbp=0;
   dcp=0;
   for k=4:-1:1;

      ix=bitget(temp,k);
      dap=db;
      dbp=xor(xor(ix,da),dc);
      dcp=xor(ix,da);
      da=dap;
      db=dbp;
      dc=dcp;
   end

   temp=bitset(temp,7,da);
   temp=bitset(temp,6,db);
   temp=bitset(temp,5,dc);

   codewords(j)=temp;
end
```

APPENDIX H

Matlab Code for Decoding Using the Hamming (7, 4) Code

```
function output=decoder(r)
%this function will decode a word given the word (r) and ht.
%   where r is n long, ht must be n-k wide and n tall
%   this example will use a (7,4) code (n=7, k=4)
%   try using: r=[1 0 1 1 0 1 1];

%this is h transform
ht=[1 0 0; 0 1 0; 1 1 0; 0 0 1; 1 0 1; 0 1 1; 1 1 1];

%this is the particular syndrome
k=r*ht;
k=mod(k,2);

%k can now be used as a particular place in the LUT
k = (2^0)*k(3)+(2^1)*k(2)+(2^2)*k(1);

%this is the LUT of error code words
e=[0 0 0 0 0 0 0; 0 0 0 1 0 0 0; 0 1 0 0 0 0 0; 0 0 0 0 0 1 0; 1 0 0 0 0 0 0; 0 0 0 0 1 0 0; 0 0
1 0 0 0 0; 0 0 0 0 0 0 1];
e2=e';

%s is the index telling which error code to use
s=7*k+1;
%the output is the corrected r
output=xor(r, e2(s:s+6))
```

APPENDIX I

Matlab Code for Determining the Parity Bits

```
function y2=paritybits(codeword)
%this function finds the parity bits for the chunks in given codeword
%use: paritybits(codeword)
% codeword is the list of chunks to get parity bits for
% codeword should be like: [6 4 6 1 12 14]

%%  initialization
codeword=[6 4 6 1 12 14];
y2=zeros(1.2*size(codeword));

codeword=uint8(codeword);
y2=uint8(y2);

%this will be the word of parity bits
p=zeros(15);

%this is used to put only 5 parities into p
count=0;

%this will keep track of where to put p in y2
i=0;
pinsert=0;

%%  finding parity
%this will run for every chunk in codeword
for j=1:length(codeword)

    %this holds the particular chunk to be looked at
    temp=codeword(j);

    %initializations for every encoding cycle
    da=0;
    db=0;
    dc=0;
    dap=0;
    dbp=0;
    dcp=0;
```

```
    %this puts the four info bits of the chunk through the encoder
    for k=4:-1:1;
        ix=bitget(temp,k);
        dap=db;
        dbp=xor(xor(ix,da),dc);
        dcp=xor(ix,da);
        da=dap;
        db=dbp;
        dc=dcp;
    end

    %the parity bits for the particular word (j) are put in place
    %p will look like [p1 p2 p3 p4 p5] as 5 parities, 3 bits each
    p((3*(j-1)+1):(3*(j-1)+3))=[da db dc];

    count=count+1;

    %%  make the encoded codeword

    %this puts the current word into the correct spot of y2
    y2(pinsert+count)=temp;

    %this runs once p has 5 parities in it (it's full)
    if count==5
        %resets count and adds 1 to the p index
        count=0;
        i=i+1;
        %pinsert is the placement for p
        pinsert=5*(i-1)+2;

        %this puts p into the correct spot of y2
        y2(pinsert)=p;

        %resets p
        p=zeros(15);
    end

end
```

APPENDIX J

Matlab Code for Keeping the MSB

```
function output=mbits(d, m, cells)
%This function will grab m bits from d and 0-fill the rest of the cells
%this is for 24-bit values, so 6 cells of a nibble each
%use of the function: mbits(d,m,cells)
%  d (the array of differences for the packet), ex: [27 12 -9 4 -2 -13]'
%  m (the number of bits to keep from d, must be mult of 4)
%  cells (array,the number of cells each number in d is in)

%this is a temporary d, m and cells to test the function
tempd=[27 12 -9 4 -2 -13]';
d=tempd;
m=4;
cells=[4 2 1 1 3 5];

for k=1:length(d) %this will keep the MSB (m) of each d(k)
   if cells(k)==1
      for i = 1:4-m
         a(k) = bitset(d(k),(4-m)-i+1,0);
      end

   elseif cells(k)==2
      for i = 1:8-m
         a(k) = bitset(d(k),(8-m)-i+1,0);
      end

   elseif cells(k)==3
      for i = 1:12-m
         a(k) = bitset(d(k),(12-m)-i+1,0);
      end

   elseif cells(k)==4
      for i = 1:16-m
         a(k) = bitset(d(k),(16-m)-i+1,0);
      end
   else
   fprintf('Value %d is not worth compressing it is more than 16 bits', k);
   end
end
```

APPENDIX K

Statistical Analysis of Human Subject Testing Results

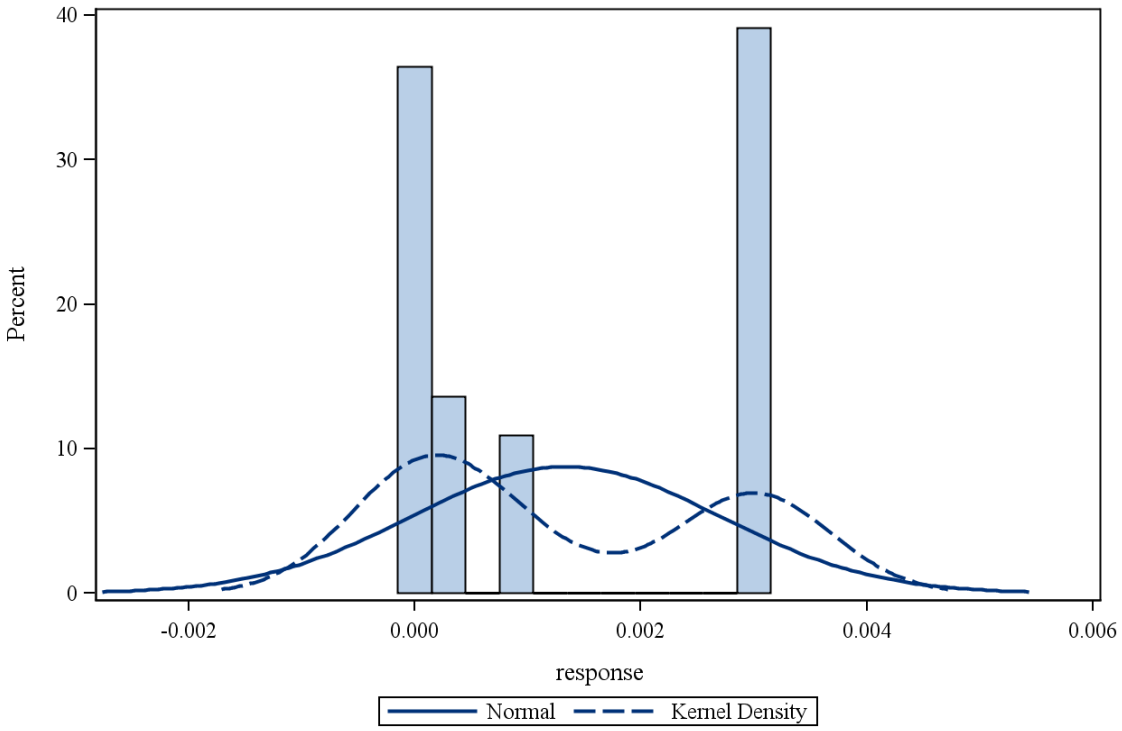

Figure K.1: Histogram of the Response Data by Response Distribution

Response by Packet Distribution – The Mixed Procedure

Model Information	
Data Set	WORK.TEMP
Dependent Variable	response
Covariance Structure	Variance Components
Estimation Method	REML
Residual Variance Method	Profile
Fixed Effects SE Method	Model-Based
Degrees of Freedom Method	Containment

Class Level Information		
Class	**Levels**	**Values**
ORDER	9	1 2 3 4 5 6 7 8 9
SUBJECT	54	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
method	3	m1 m2 m3
packet	3	p1 p2 p3

Dimensions	
Covariance Parameters	2
Columns in X	16
Columns in Z	54
Subjects	1
Max Obs Per Subject	486

Number of Observations	
Number of Observations Read	486
Number of Observations Used	486
Number of Observations Not Used	0

Iteration History			
Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	-5263.93636706	
1	1	-5275.85690666	0.00000000

Covariance Parameter Estimates	
Cov Parm	Estimate
SUBJECT	8.145E-8
Residual	7.939E-7

Convergence criteria met.

Fit Statistics	
-2 Res Log Likelihood	-5275.9
AIC (smaller is better)	-5271.9
AICC (smaller is better)	-5271.8
BIC (smaller is better)	-5267.9

Type 3 Tests of Fixed Effects				
Effect	Num DF	Den DF	F Value	Pr > F
method	2	424	185.00	<.0001
packet	2	424	27.62	<.0001
method*packet	4	424	47.18	<.0001

Least Squares Means							
Effect	method	packet	Estimate	Standard Error	DF	t Value	Pr > |t|
method	m1		0.001612	0.000080	424	20.13	<.0001
method	m2		0.002116	0.000080	424	26.43	<.0001
method	m3		0.000273	0.000080	424	3.41	0.0007
packet		p1	0.001325	0.000080	424	16.55	<.0001
packet		p2	0.001706	0.000080	424	21.31	<.0001
packet		p3	0.000970	0.000080	424	12.12	<.0001
method*packet	m1	p1	0.002288	0.000127	424	17.97	<.0001
method*packet	m1	p2	0.002218	0.000127	424	17.42	<.0001
method*packet	m1	p3	0.000329	0.000127	424	2.58	0.0102
method*packet	m2	p1	0.001291	0.000127	424	10.14	<.0001
method*packet	m2	p2	0.002500	0.000127	424	19.64	<.0001
method*packet	m2	p3	0.002557	0.000127	424	20.08	<.0001
method*packet	m3	p1	0.000397	0.000127	424	3.12	0.0020
method*packet	m3	p2	0.000399	0.000127	424	3.13	0.0019
method*packet	m3	p3	0.000025	0.000127	424	0.19	0.8467

Differences of Least Squares Means											
Effect	method	packet	_method	_packet	Estimate	Standard Error	DF	t Value	Pr > |t|	Adjustment	Adj P
method	m1		m2		-0.00050	0.000099	424	-5.09	<.0001	Tukey-Kramer	<.0001
method	m1		m3		0.001338	0.000099	424	13.52	<.0001	Tukey-Kramer	<.0001
method	m2		m3		0.001842	0.000099	424	18.61	<.0001	Tukey-Kramer	<.0001
packet		p1		p2	-0.00038	0.000099	424	-3.84	0.0001	Tukey-Kramer	0.0004
packet		p1		p3	0.000355	0.000099	424	3.59	0.0004	Tukey-Kramer	0.0011
packet		p2		p3	0.000736	0.000099	424	7.43	<.0001	Tukey-Kramer	<.0001

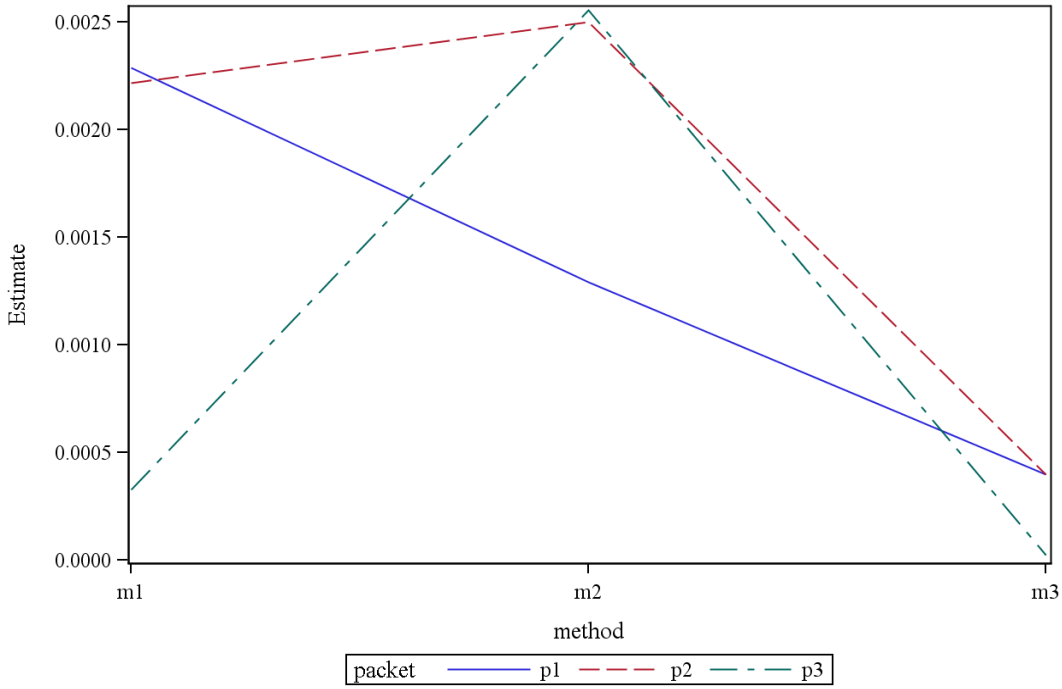

Figure K.2: Plot of the Response Data for Methods and Packet Lengths

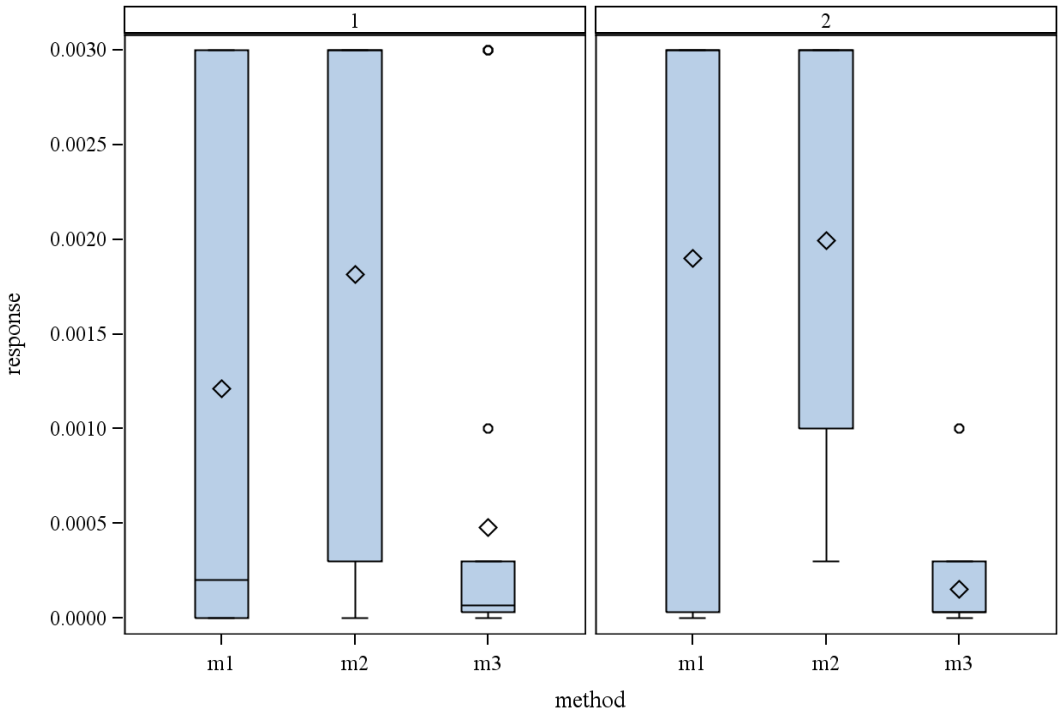

Figure K.3: Analysis of Data by Order

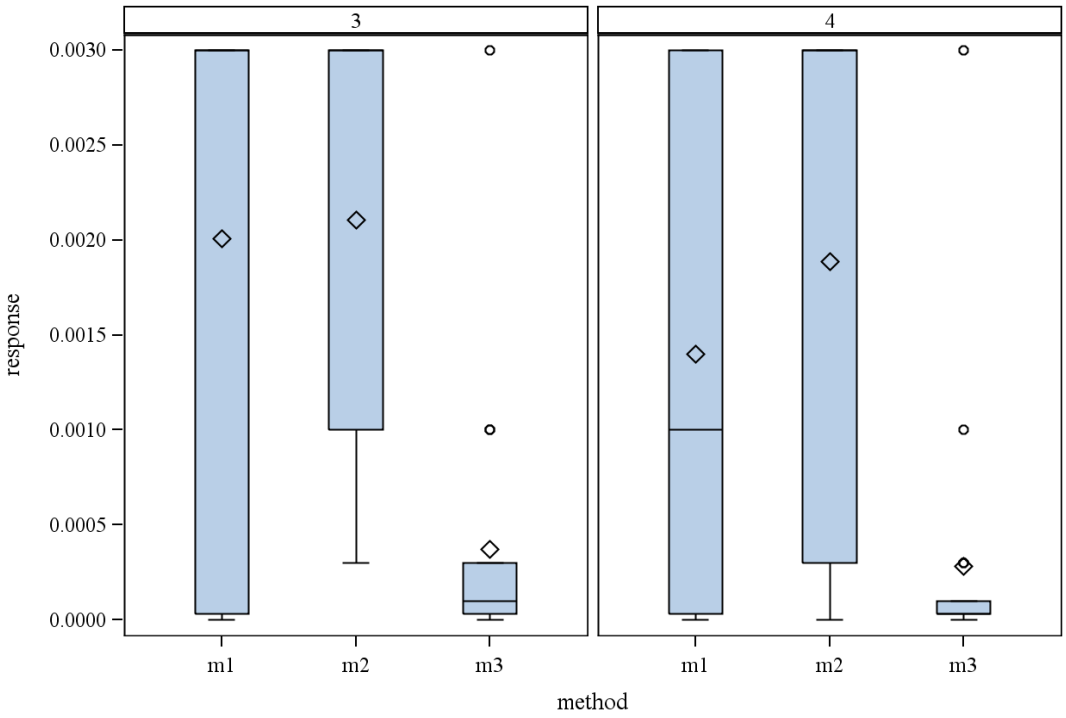

Figure K.4: Analysis of Data by Order

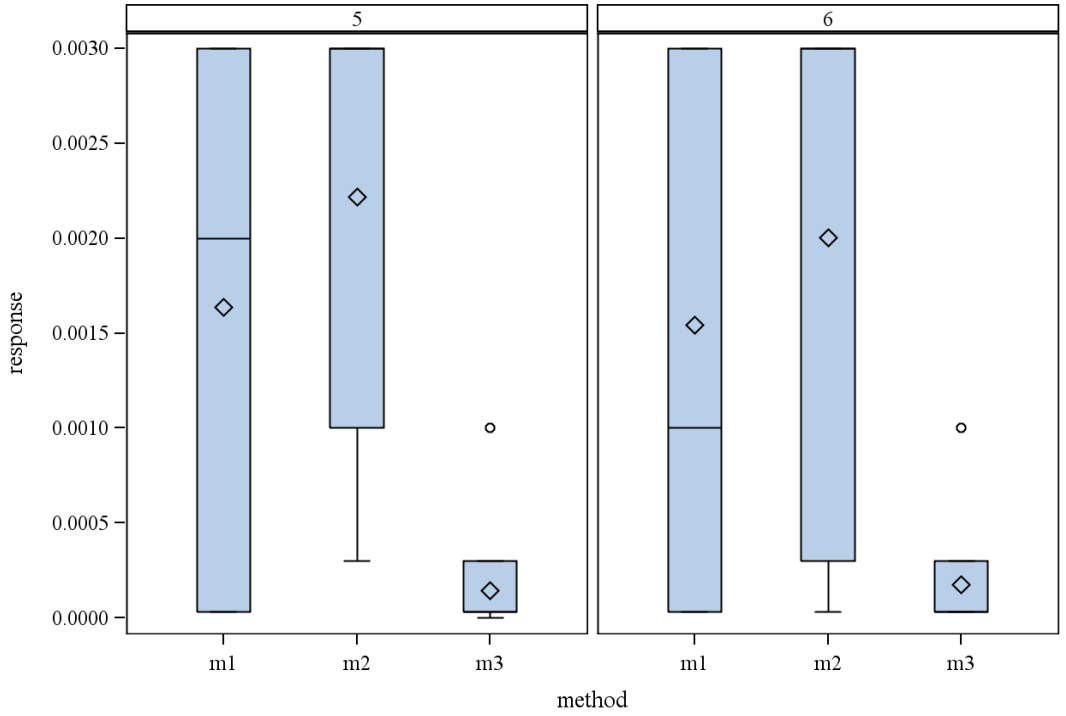

Figure K.5: Analysis of Data by Order

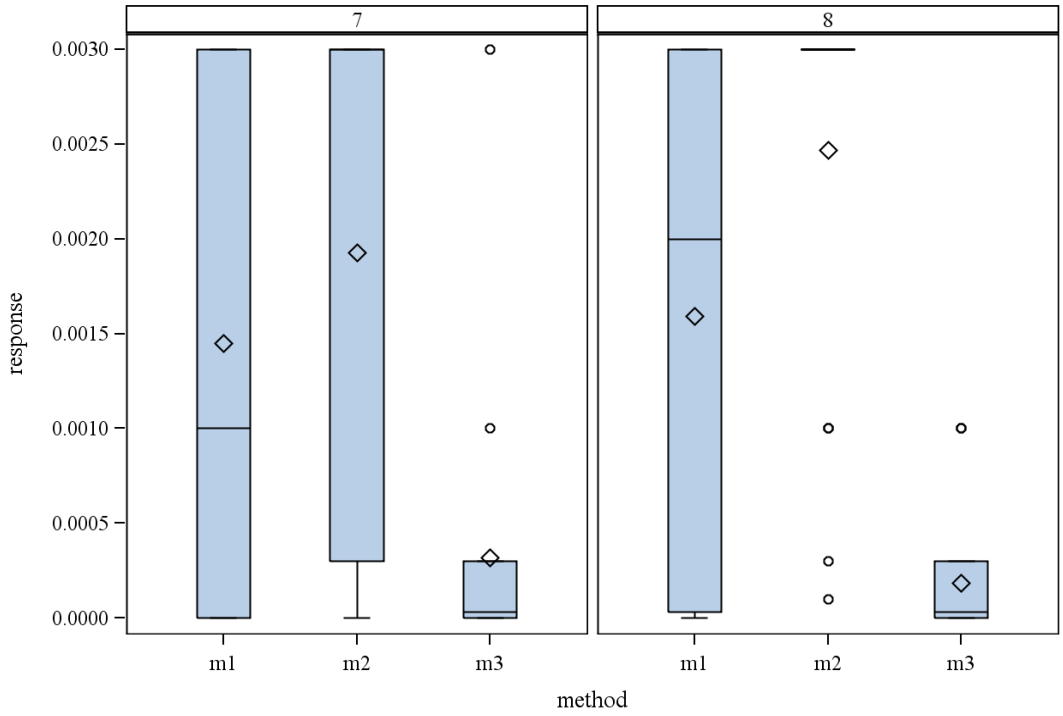

Figure K.6: Analysis of Data by Order

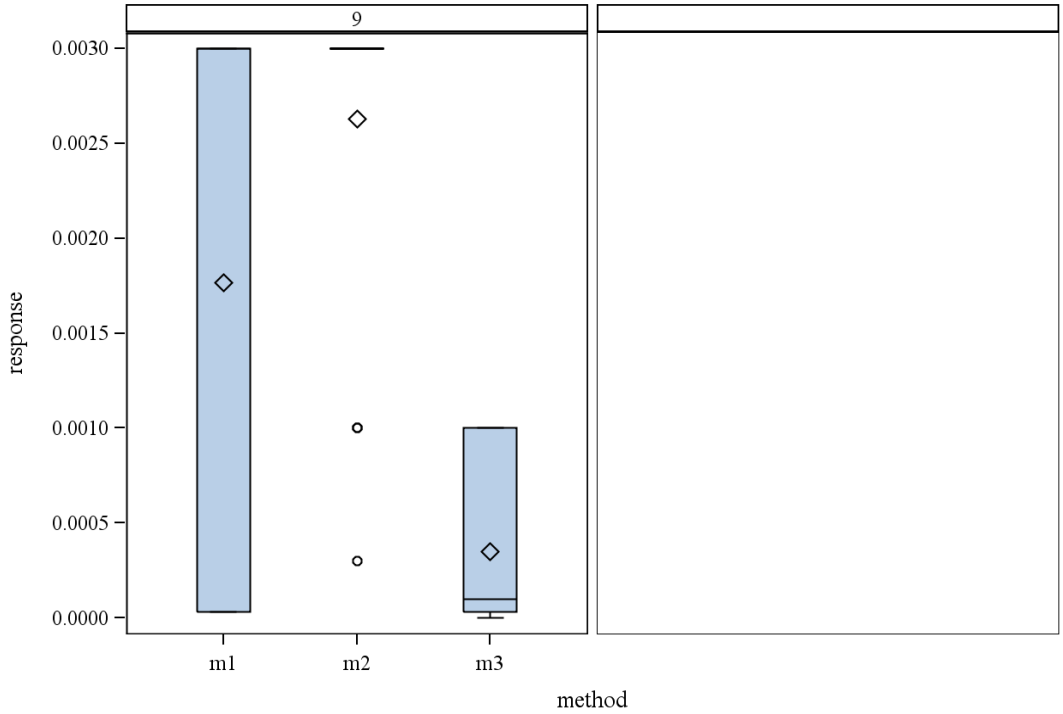

Figure K.7: Analysis of Data by Order

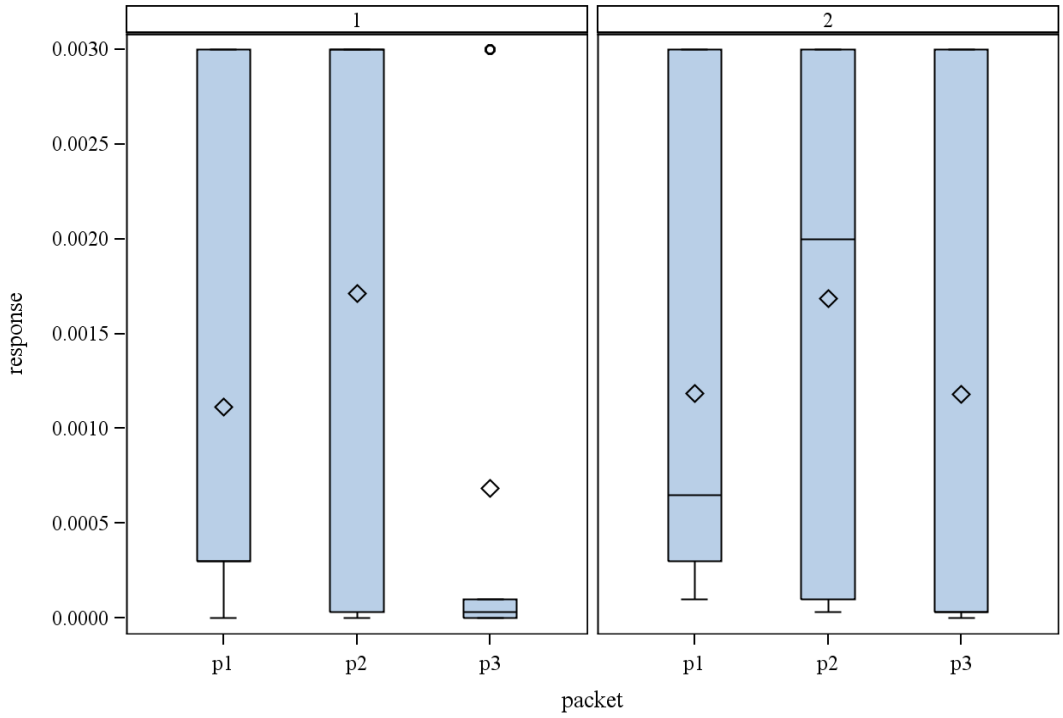

Figure K.8: Analysis of Data by Order

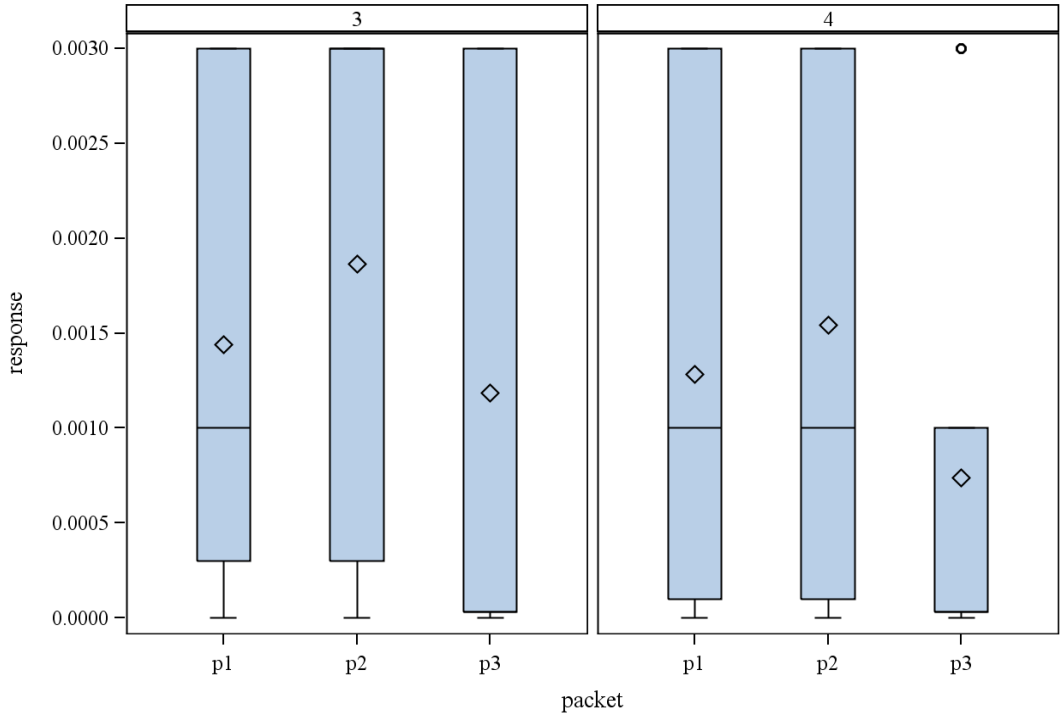

Figure K.9: Analysis of Data by Order

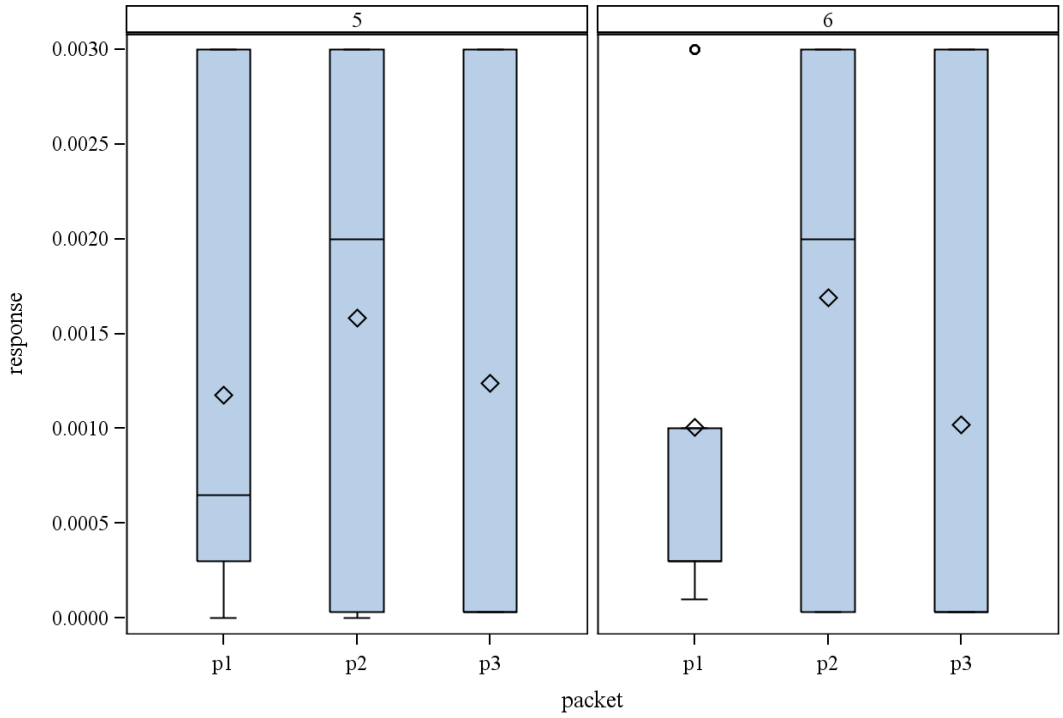

Figure K.10: Analysis of Data by Order

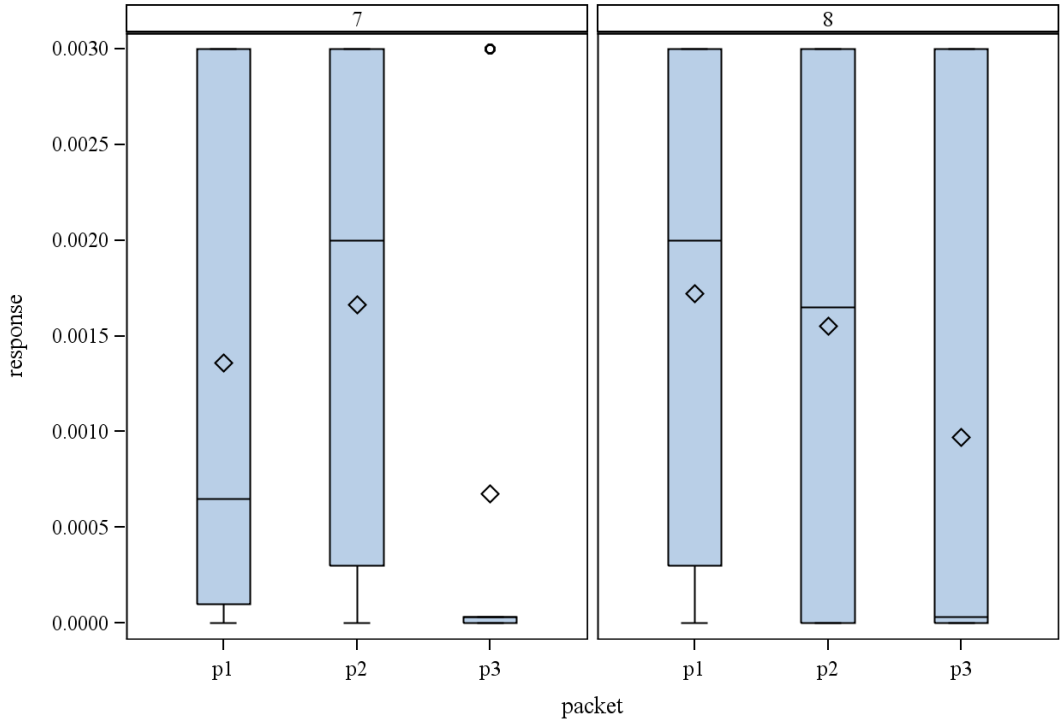

Figure K.11: Analysis of Data by Order

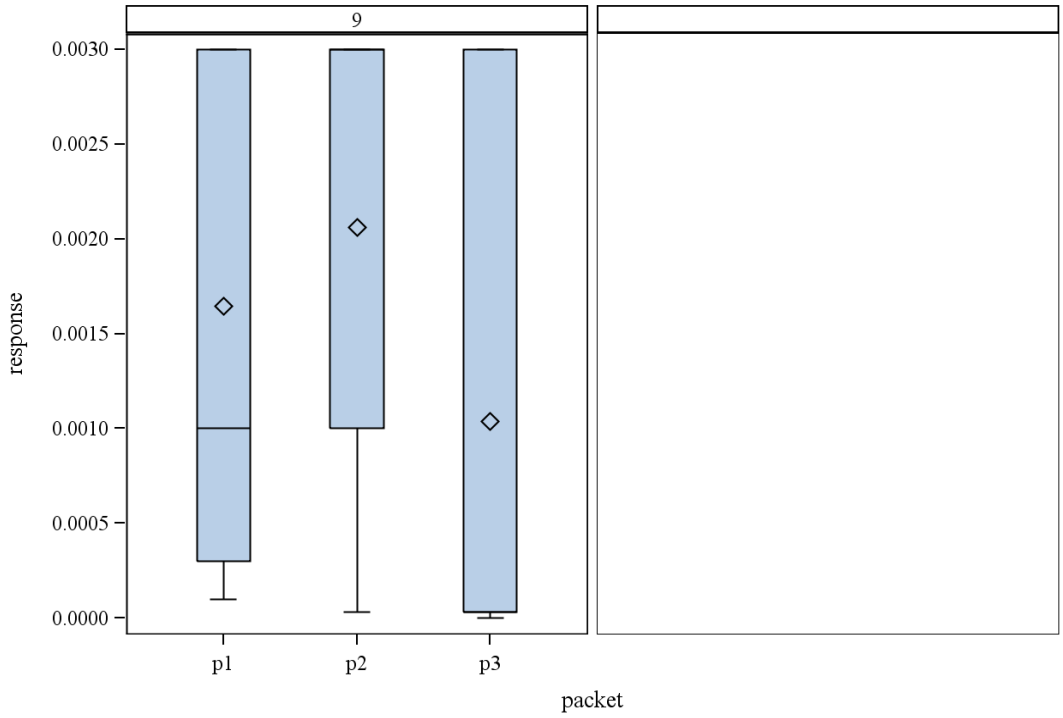

Figure K.12: Analysis of Data by Order

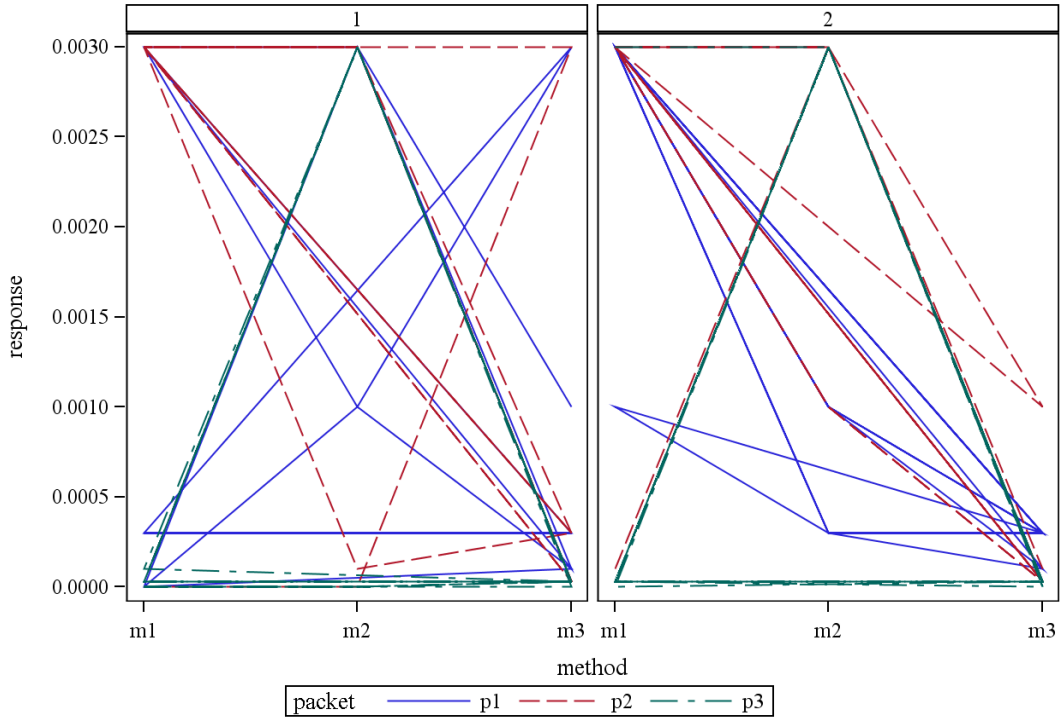

Figure K.13: Analysis of Data by Order

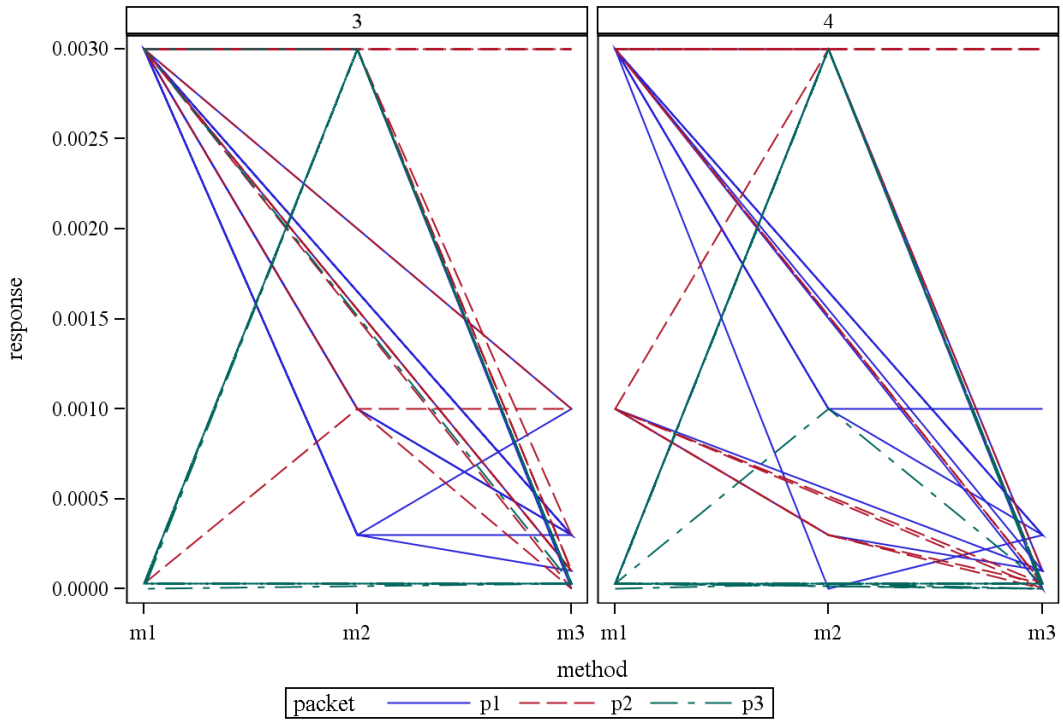

Figure K.14: Analysis of Data by Order

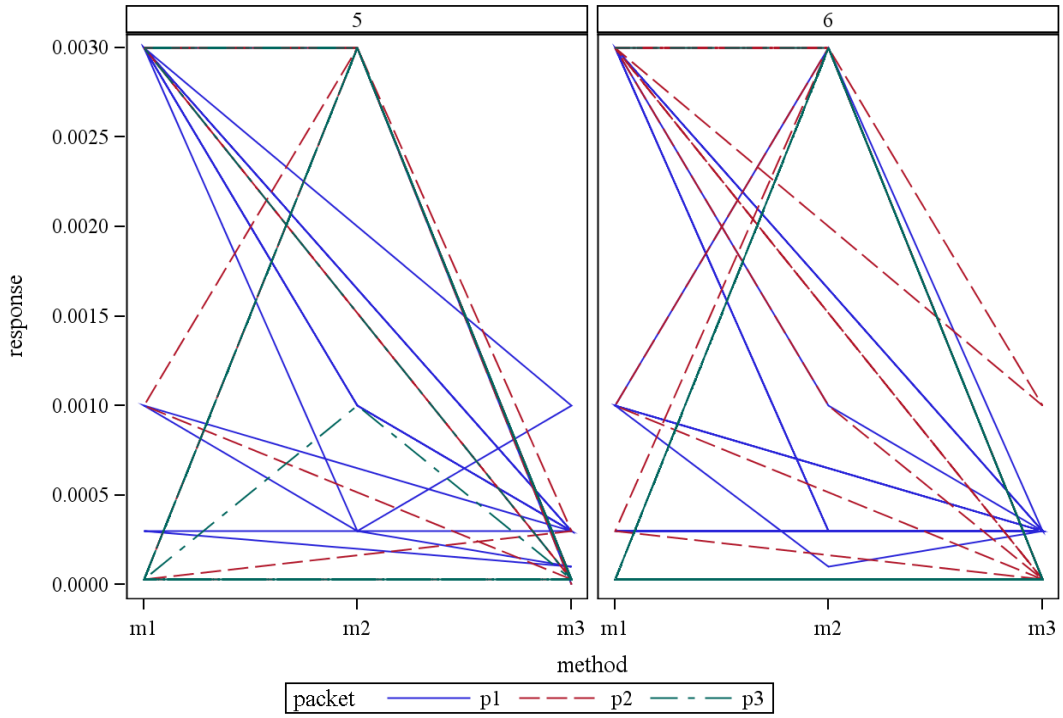

Figure K.15: Analysis of Data by Order

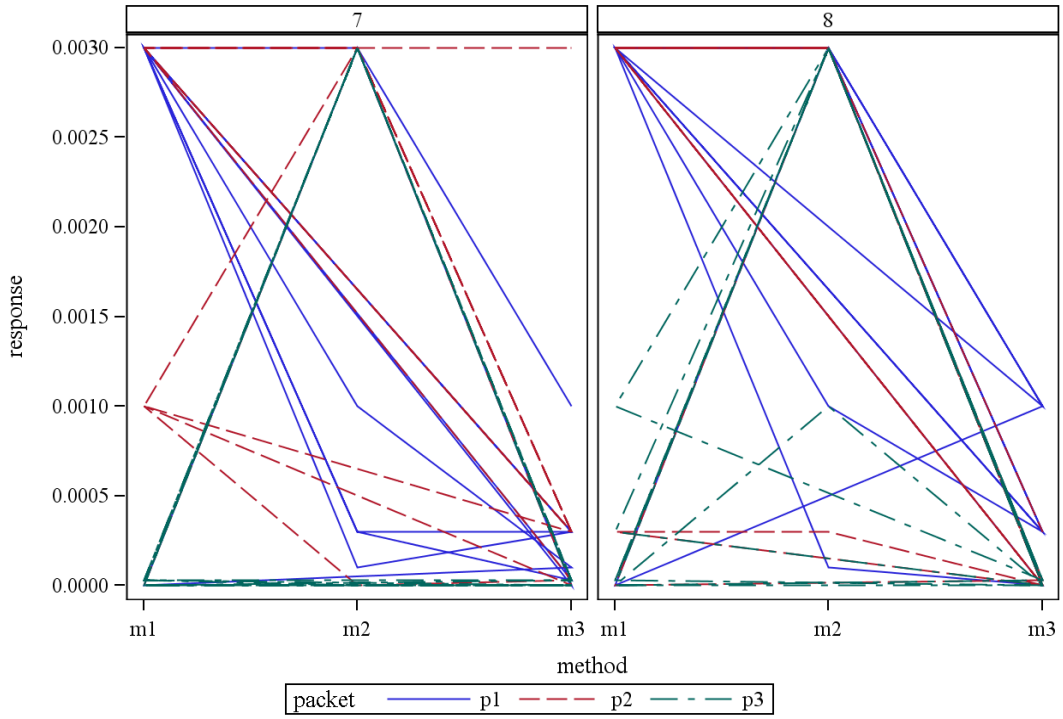

Figure K.16: Analysis of Data by Order

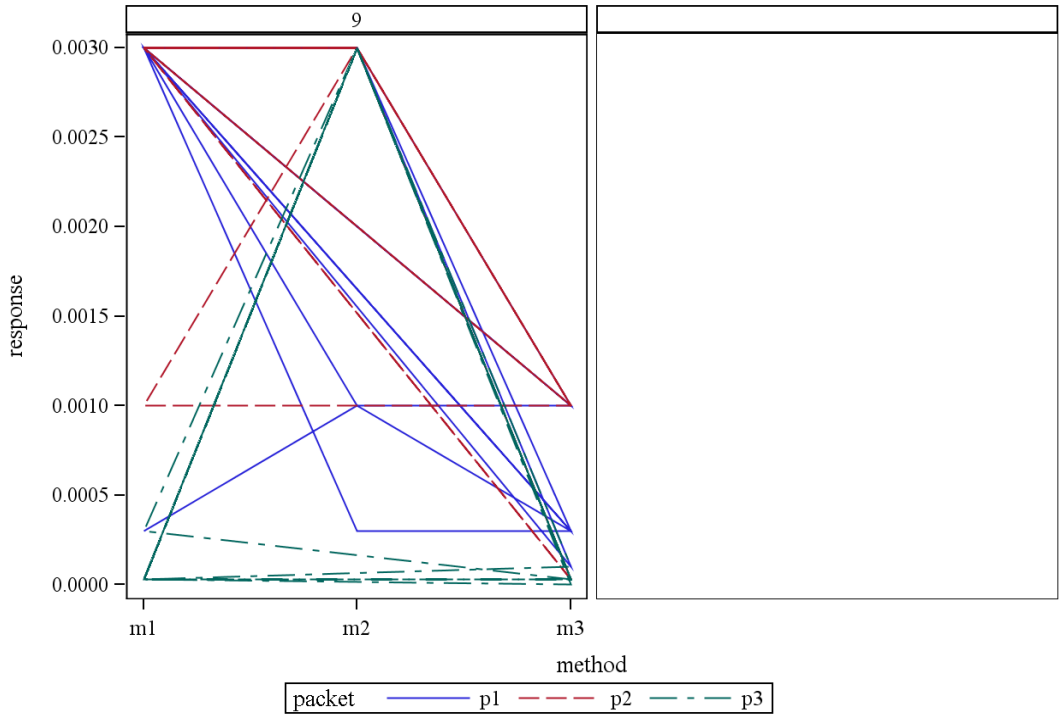

Figure K.17: Analysis of Data by Order

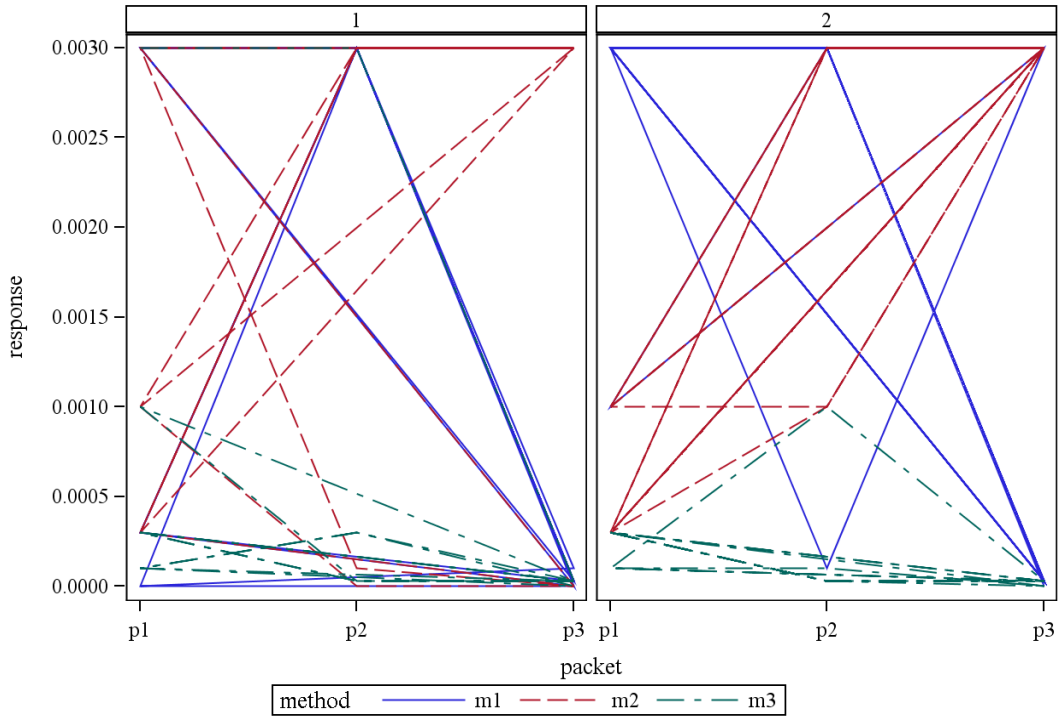

Figure K.18: Analysis of Data by Order

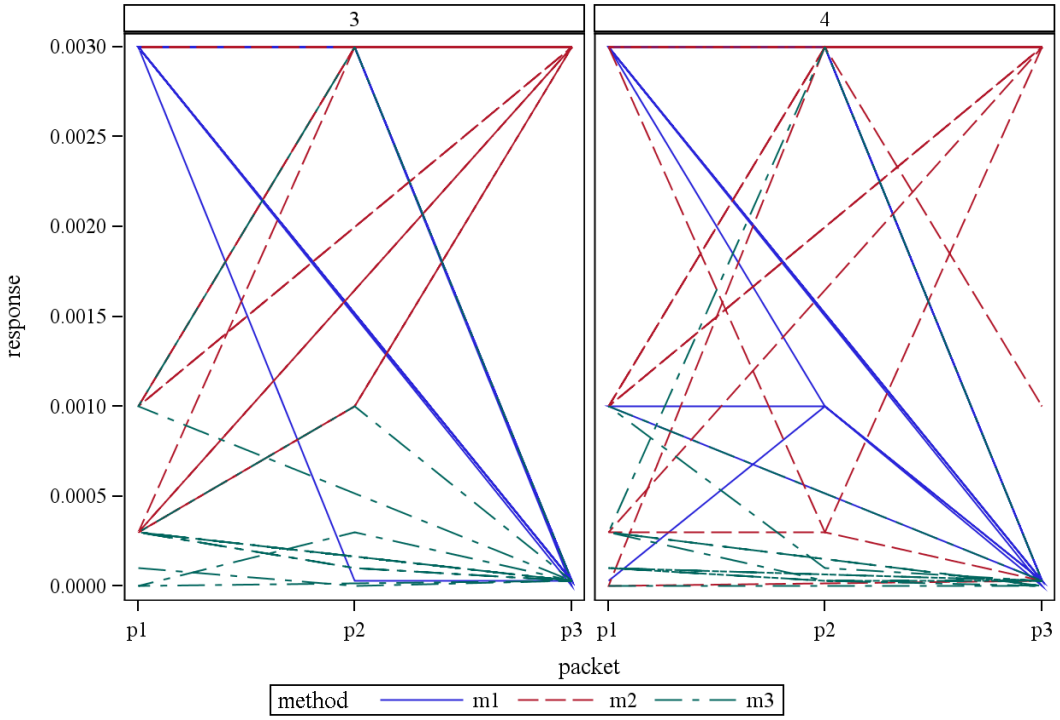

Figure K.19: Analysis of Data by Order

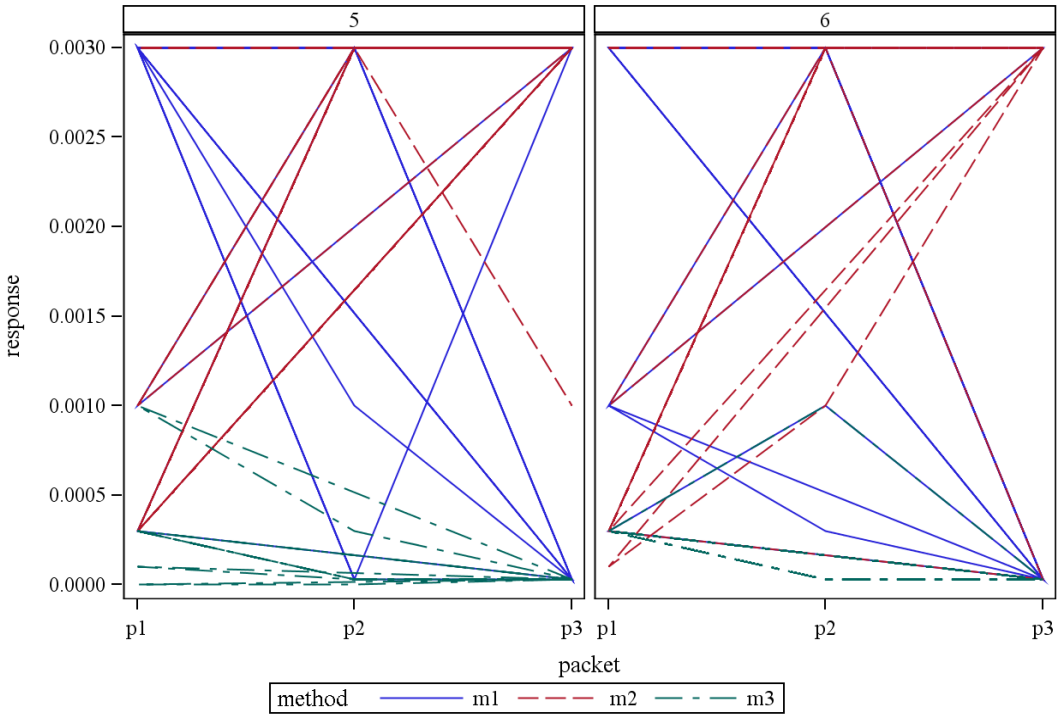

Figure K.20: Analysis of Data by Order

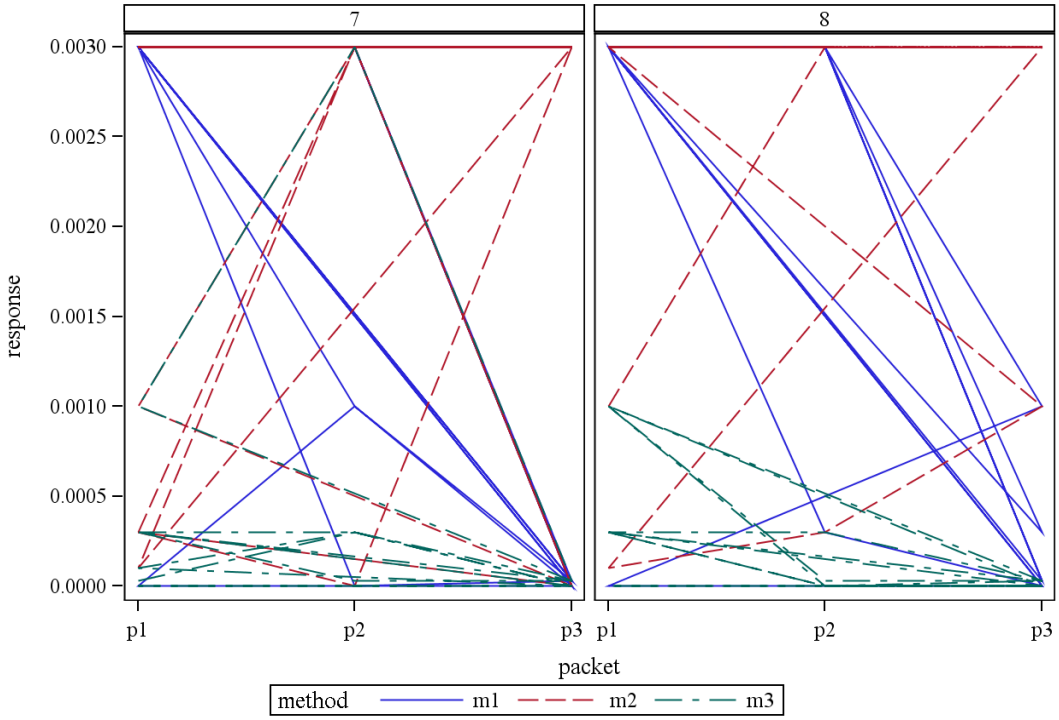

Figure K.21: Analysis of Data by Order

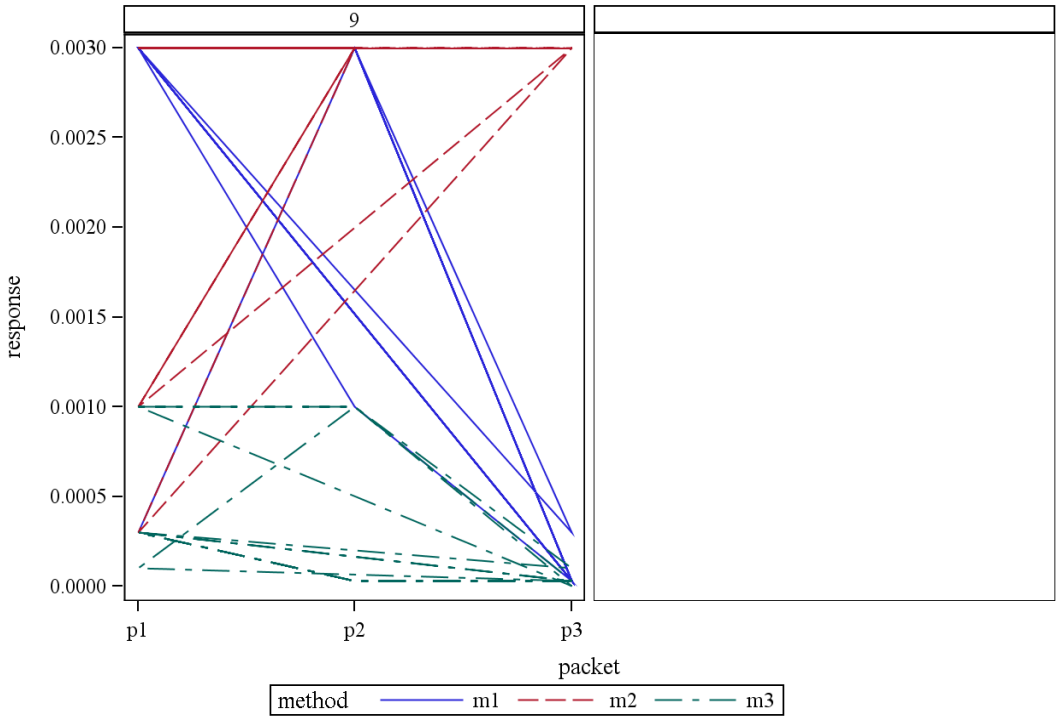

Figure K.22: Analysis of Data by Order

Model Information	
Data Set	WORK.RANK_TEMP
Dependent Variable	responseRank
Covariance Structure	Variance Components
Estimation Method	REML
Residual Variance Method	Profile
Fixed Effects SE Method	Model-Based
Degrees of Freedom Method	Containment

Class Level Information		
Class	**Levels**	**Values**
ORDER	9	1 2 3 4 5 6 7 8 9
SUBJECT	54	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
method	3	m1 m2 m3
packet	3	p1 p2 p3

Dimensions	
Covariance Parameters	2
Columns in X	16
Columns in Z	54
Subjects	1
Max Obs Per Subject	486

Number of Observations	
Number of Observations Read	486
Number of Observations Used	486
Number of Observations Not Used	0

Iteration History			
Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	4112.49240714	
1	1	4063.46623794	0.00000000

Convergence criteria met.

Covariance Parameter Estimates	
Cov Parm	Estimate
SUBJECT	64.3459
Residual	237.05

Fit Statistics	
-2 Res Log Likelihood	4063.5
AIC (smaller is better)	4067.5
AICC (smaller is better)	4067.5
BIC (smaller is better)	4071.4

Type 3 Tests of Fixed Effects				
Effect	Num DF	Den DF	F Value	Pr > F
method	2	424	2.43	0.0891
packet	2	424	5.43	0.0047
method*packet	4	424	4.62	0.0012

Least Squares Means							
Effect	method	packet	Estimate	Standard Error	DF	t Value	Pr > |t|
method	m1		14.4198	1.6294	424	8.85	<.0001
method	m2		13.2222	1.6294	424	8.11	<.0001
method	m3		16.9198	1.6294	424	10.38	<.0001
packet		p1	17.4259	1.6294	424	10.69	<.0001
packet		p2	15.2963	1.6294	424	9.39	<.0001
packet		p3	11.8395	1.6294	424	7.27	<.0001
method*packet	m1	p1	13.2963	2.3625	424	5.63	<.0001
method*packet	m1	p2	14.5185	2.3625	424	6.15	<.0001
method*packet	m1	p3	15.4444	2.3625	424	6.54	<.0001
method*packet	m2	p1	20.1667	2.3625	424	8.54	<.0001
method*packet	m2	p2	10.5185	2.3625	424	4.45	<.0001
method*packet	m2	p3	8.9815	2.3625	424	3.80	0.0002
method*packet	m3	p1	18.8148	2.3625	424	7.96	<.0001
method*packet	m3	p2	20.8519	2.3625	424	8.83	<.0001
method*packet	m3	p3	11.0926	2.3625	424	4.70	<.0001

Differences of Least Squares Means											
Effect	**method**	**packet**	**_method**	**_packet**	**Estimate**	**Standard Error**	**DF**	**t Value**	**Pr > |t|**	**Adjustment**	**Adj P**
method	m1		m2		1.1975	1.7107	424	0.70	0.4843	Tukey-Kramer	0.7636
method	m1		m3		-2.5000	1.7107	424	-1.46	0.1447	Tukey-Kramer	0.3107
method	m2		m3		-3.6975	1.7107	424	-2.16	0.0312	Tukey-Kramer	0.0791
packet		p1		p2	2.1296	1.7107	424	1.24	0.2139	Tukey-Kramer	0.4274
packet		p1		p3	5.5864	1.7107	424	3.27	0.0012	Tukey-Kramer	0.0034
packet		p2		p3	3.4568	1.7107	424	2.02	0.0439	Tukey-Kramer	0.1085

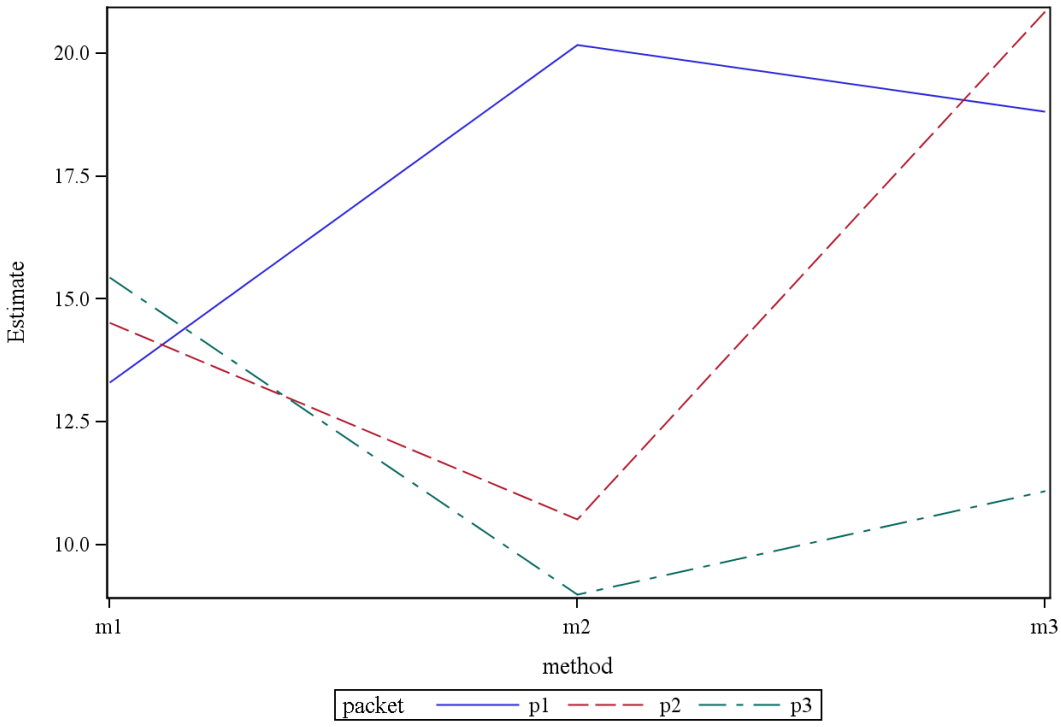

Figure K.23: Analysis of Ranked Response Data

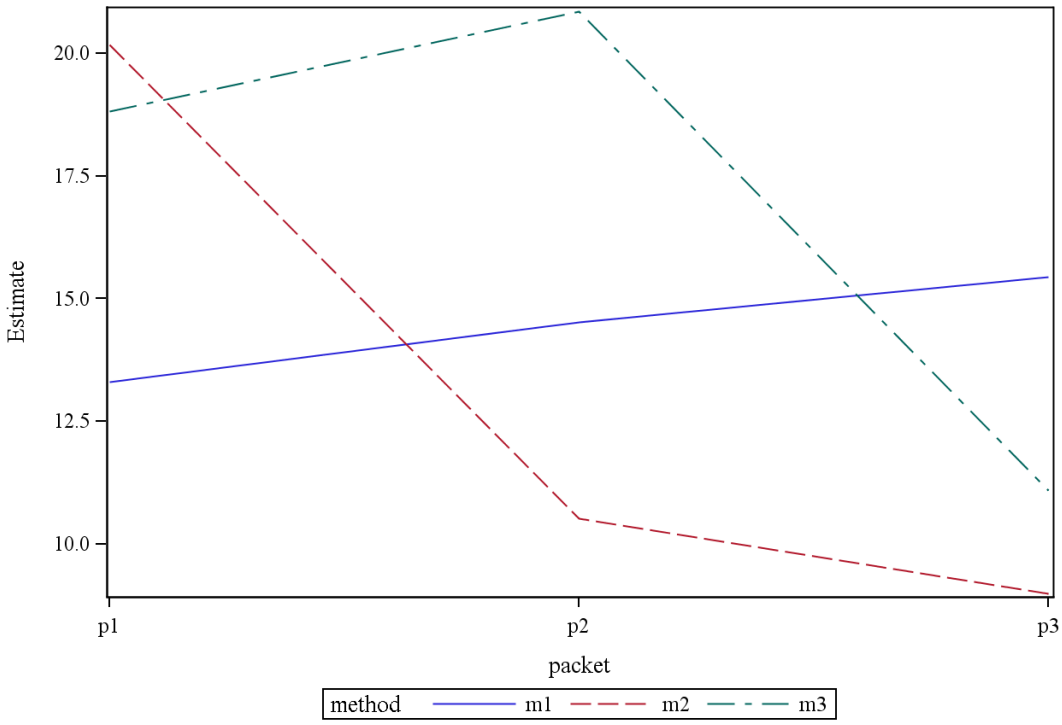

Figure K.24: Analysis of Data by Order

BIBLIOGRAPHY

[1] S. A. Gelfand, "Measurement Principles and the Nature of Hearing," in *Essentials of Audiology*, 3rd ed. New York: Thieme, 2009, ch. 3, Central and Temporal Masking, pp. 103

[2] N. Aldrich, "The Ear," in *Digital Audio Explained For The Audio Engineer*, 2nd ed. Fort Wayne: Sweetwater Sound, 2004, ch. 6, Cochlear Filters, pp. 74-76

[3] D. Salomon, *Data Compression: The Complete Reference*, 4th ed. Springer, 2006

[4] C. R. Johnson, Jr., W. A. Sethares. And A. Klein, "Coding and Decoding," in *Software Receiver Design: Build Your Own Digital Communications System in Five Easy Steps*, 2008, ch. 2, A telecommunication System, pp. 23-24

[5] S. Lin, D. Costello, *Error Control Coding: Fundamentals and Applications*, 2nd ed. Prentice Hall, 2004

[6] J. Baylis, "Hamming's Solution," in *Error-Correcting Codes: A Mathematical Introduction*, Florida: CRC, 1998, ch. 2, Reducing the Price

[7] W. C. Huffman, V. Pless, "Hamming Codes," in *Fundamentals of Error-Correcting Codes*, United Kingdom: Cambridge, 2003, ch. 1, Basic Concepts of Linear Codes, pp. 29-31

[8] Q. Quach. (2007, Nov. 3). Matlab GUI Tutorial – Button Types and Button Group [Online]. Available: http://blinkdagger.com/matlab/matlab-gui-tutorial-buttons-button-group

[9] R. Narasimhan. (2006, Jan. 19). *GUI Output* [Online]. Available: http://www.mathworks.com/matlabcentral/newsreader/view_thread/113779

[10] D. R. Oran, and William VerSteeg, "Monitoring and Correcting Upstream Packet Loss," U.S. Patent 370390, Sept. 30, 2010

[11] C. Perkins, and O. Hodson. (1998, June). Options for Repair of Streaming Media [Online]. Available: https://tools.ietf.org/html/rfc2354

[12] T. K. Chua, and D.C. Pheanis. (2005). Perceptual Audio Quality Analysis of VOIP Loss-Recovery Techniques [Online]. Available: http://www.actapress.com/Abstract.aspx?paperId=22846

[13] D. Florencio, Philip A. Chou, and Li-Wei He, “Real-Time Jitter Control and Packet-Loss Concealment in an Audio Signal,” US Patent 20090304032, Sept. 30, 2010

[14] D. Hardman. (2003, April 23). Noise and Voice Quality in VoIP Environments [Online]. Available: http://cp.literature.agilent.com/litweb/pdf/5988-9345EN.pdf

[15] Mohamed, Rubino, and Varela, “A Method for Quantitative Evaluation of Audio Quality over Packet Networks and its Comparison with Existing Techniques,” May, 2004.