APPLICATION OF CHANNEL ESTIMATION TO
UNDERWATER ACOUSTIC COMMUNICATION

by

Brian S. Borowski

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Brian S. Borowski
Brian S. Borowski, Candidate

ADVISORY COMMITTEE

Dan Duchamp, Chairman	6/22/10 Date
Bruce McNair	6/22/10 Date
Charles Suffel	6/22/10 Date
Joseph Mitola III	6/22/10 Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2010

APPLICATION OF CHANNEL ESTIMATION TO UNDERWATER ACOUSTIC COMMUNICATION

ABSTRACT

The underwater channel poses numerous challenges for acoustic communication. Acoustic waves suffer long propagation delay, multipath and fading, limited bandwidth, and potentially high spatial and temporal variability. In addition, there is no typical underwater acoustic channel; every body of water exhibits quantifiably different properties. Consequently, current modems – implemented in hardware with a fixed, conservative set of transmission parameters – are often ill-suited for a particular channel, resulting in performance that is far from optimum. Very little work has been done in the area of channel characterization, especially for waters only several meters deep. As a result, network simulations often make conservative and/or inaccurate assumptions about shallow underwater channels.

In this thesis the Hudson River estuary is characterized as an acoustic communication channel. The analysis reveals that the Hudson is a multipath fading channel (Rician fading over 200 m and Gamma fading over 505 m) with an extremely short coherence time of approximately 50 ms. A subset of the estimation techniques is then employed to develop a network simulation and an adaptive, real-time software modem.

The simulator converts a transmitted packet into a modulated signal and digitally mixes it with the channel estimates to produce a signal that approximates what would have been received after transmission through the physical channel. When simulating a time-invariant channel, the achieved bit error rates are, on average, within 3.34% of those obtained by transmission through the actual channel. The simulator is modular and can easily accommodate new channel estimates, modulation schemes, receiver techniques, and alternate implementations of higher layers in the network stack.

In the software modem each packet is preceded by an acoustic signal that is used for impulse response estimation. The modem then processes the signal in real time and uses the inverse impulse response to equalize the channel, allowing for the transmission of packets at higher data rates with symbols whose duration is less than the multipath spread of the channel. In a time-invariant shallow water test channel, the modem correctly decoded packets at up to 6 kbps. In an AWGN channel, the modem's BER approached the theoretical limit for the given SNR.

Author: Brian S. Borowski

Advisor: Dan Duchamp

Date: June 22, 2010

Department: Computer Science

Degree: Doctor of Philosophy

Acknowledgments

I would like to thank my advisor Dr. Dan Duchamp for his guidance, support, and encouragement throughout my doctoral studies. I am very grateful for the substantial amount of time he set aside for my weekly meetings, which were always very productive and left me thinking about new ideas and alternate approaches to solving problems. Dr. Duchamp's practical approach to research and dedication to implementing working systems are attributes that I will carry on throughout my career. Finally, I wish to express gratitude for his flexibility in allowing me to study this rather broad, interdisciplinary topic.

I would like to express thanks to Prof. Bruce McNair for helping me to understand some of the intricacies of signal processing and linear systems. His insight and previous work experience were instrumental to the success of my projects.

Several others have contributed to my research efforts and overall learning experience. I express my thanks to each of the following:

- Dr. Joseph Mitola III, for providing references and tips for characterizing channels and building digital communication systems.
- Dr. Charles Suffel, for putting me in contact with people who could discuss the interdisciplinary aspects of my research and for his general guidance throughout my doctoral studies.
- Alex Sedunov, Nikolay Sedunov, and Mikhail Tsionskiy, for gathering data in the Hudson River estuary, providing equipment for my office experiments, and helping me get up to speed in analyzing the measurements.
- Dr. Theodoros Kamakaris and Dr. Didem Kivanc-Tureli for sharing their knowledge and ideas about communication systems.

- Dr. Alexander Sutin, for giving me a background in underwater acoustics and the opportunity to study diver detection using passive sonar.
- Dr. Dimitri Donskoy, for several discussions on underwater acoustic propagation.
- Dr. Ionut Florescu, for his help with distribution fitting.
- Dr. Barry Bunin, for showing me the basics of electrical engineering.

I would also like to thank the Maritime Security Laboratory and members of the Stanley Fellowship Committee for providing the funding for my doctoral studies. Without their support none of this would have been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Applications of Underwater Acoustics

One of the earliest references to the existence of underwater acoustics appears in one of Leonardo da Vinci's notebooks [Urick 1996]. In 1490, he wrote, "If you cause your ship to stop, and place the head of a long tube in the water and place the outer extremity to your ear, you will hear ships at a great distance from you." Motivated by the sinking of the Titanic, in 1912 L. F. Richardson filed a patent application with the British Patent Office for echo ranging with underwater acoustics, but he did not implement his proposal [Urick 1996]. Meanwhile, in the United States, R. A. Fessenden designed and built a moving-coil transducer for both submarine signaling and echo ranging which, by 1914, was able to detect an iceberg at a distance of two miles [Urick 1996]. Military applications of sonar were stimulated by the outbreak of the World War I and II, though it wasn't until the latter where echo-ranging sonar was able to effectively combat the German U-boat [Urick 1996]. An underwater telephone, developed in 1945 in the United States for communicating with submarines, was one of the first underwater communication systems [Stojanovic 2003]. Today, underwater acoustics are used for communication in a broad range of applications, mostly sensor-based, including ocean sampling networks, environmental monitoring, undersea explorations, disaster prevention, assisted navigation, speech transmission between divers, distributed tactical surveillance, and mine reconnaissance [Akyildiz 2005; Stojanovic 2003].

The Maritime Security Laboratory (MSL) at Stevens Institute of Technology has been researching port security, with emphasis on detecting underwater threats in the Hudson River. A hypothetical extension to the lab's efforts is to design an underwater acoustic sensor network to aid in detecting divers, surface swimmers, AUVs, and small surface boats. The nodes will gather

and process signals in real time and send the resulting information via acoustic links to a base station with satellite or RF capabilities.

Over the past decade, network systems for similar applications have been deployed. For example, the initial motivation for the Seaweb project was the need for wide-area undersea surveillance in littoral waters by means of a deployable autonomous distributed system (DADS) [Rice 2001]. Seaweb '98 led off a series of annual ocean experiments intended to progressively advance the state of the art in underwater acoustic communications. The goal of Seaweb 2008, the latest of these experiments, is to provide surveillance of the Port of Long Beach [NPS 2008]. The Persistent Littoral Undersea Surveillance Network (PLUSNet) is a multi-institution program sponsored by the Office of Naval Research which aims to provide autonomous detection and tracking of quiet submarines [Grund 2006]. While the network supports satellite or RF links between nodes, acoustic links are reserved for nodes that do not have a surface presence or must maintain depth to carry out a mission. In addition, NATO recognizes the importance of detecting submarines and other small submersibles and has established a research project for Reconnaissance, Surveillance, and Undersea Networks (RSN) [NATO 2008]. Among other goals, the plan calls for "applied research into covert undersea communications and networking and technology research into using LAN-based information architectures."

Though many of today's efforts are directed toward security-based applications, some recent ocean exploration/monitoring projects have made use of underwater communication. Between 1999 and 2002, the Front-Resolving Observational Network with Telemetry (FRONT) study was established to accomplish data telemetry and remote control for a set of widely spaced oceanographic sensors through the use of the Seaweb underwater acoustic network [Rice 2008]. Today, the South Florida Ocean Measurement Center (SFOMC) exists as an ongoing partnership between the Navy and Florida Atlantic University for oceanographic monitoring of the Florida

Straits [Venezia 2003; GulfBase 2008]. One aspect of this project is the development and use of acoustic modems in shallow water for real-time transmission of AUV observations to mobile and fixed bottom receiving and telemetry instrumentation.

1.2 Reasons for Acoustic Communication

While some projects deploy underwater networks for ocean sampling, many others still rely on conventional techniques. There are two standard methods for gathering oceanographic data that do not make use of acoustics. One such approach is to deploy tethered sensors. Although this method results in high throughput with virtually no bit errors, it is limited to short distances in locations where the cables can be placed unobstructed. The other widely used approach is to deploy underwater sensors that record data for a specified amount of time and then are recovered upon completion of the task. With this method there are no bit errors, but there are a significant number of drawbacks [Akyildiz 2005]:

1. The (repeated) deployment and recovery of the instruments can be an expensive, difficult, or dangerous procedure.
2. The data processing cannot be performed in real time.
3. There is no interaction with the device after it is deployed, impeding any fine-tuning or reconfiguration that might be necessary for maintaining functionality.
4. It might be difficult or impossible to detect the failure of an instrument until after it is recovered, possibly resulting in the failure of the entire mission.
5. The amount of data recovered is limited by the storage space on the device itself.

Since both tethered and standalone devices have numerous disadvantages, most underwater sensor networks employ a wireless physical layer with acoustic links. Two other wireless methods, radio frequency (RF) and optical transmission, have several limitations that prevent them from being widely utilized in underwater channels, the most significant being short propagation

distance. While pure water is an insulator, most bodies of water contain dissolved salts and other matter, making them partial conductors. The level of attenuation of radio signals is directly proportional to the conductivity of the water. The attenuation of radio waves in water also rises with an increase in frequency and is proportional to $\sqrt{fs}$, where f is the frequency in Hz, and s is the conductivity of the water in mhos/meter[1]. Because of the salinity levels, attenuation in sea water is very high, and to communicate at any depth, it is necessary to use very low frequencies (long wave radio, 10 – 30 kHz) where attenuation is on the order of 3.5 to 5 dB per meter [Butler 1987].

While Maxwell's equations can be used to predict the propagation of electromagnetic waves traveling in seawater, there have been some papers describing actual measurements of horizontal propagation. Propagation in seawater 76 meters deep at 7 MHz produced a transmission distance of 460 meters [Al-Shamma'a 2004], while propagation at 14 MHz was experimentally shown to produce a transmission distance of only some 10 to 20 feet [Siegel 1973]. In both experiments, propagation exhibited significant signal loss. For frequencies from 0.1 – 20 MHz, the total signal loss over 1 km is severe, ranging from -112 to -166 dB [Al-Shamma'a 2004]. A more recent experiment demonstrated that electromagnetic waves propagated from 2 to 30 meters with a constant transmitter power of 100 mW [Cella 2009].

Because of scattering and absorption, optical systems are also limited to short distances. Scattering reduces signal levels and limits the maximum data rate when multipath stretches the time of a pulse to that of the bit time. Infrared modulation cannot be used underwater, since water is not transparent in that region of the spectrum. Visibility in the Irish Sea, for instance, is typically 0 meters and only 1 – 2 meters at best due to suspended matter in the water [Shaw 2006]. Replacing infrared LEDs with high power blue and green LEDs has been stated to pro-

[1] *mho* is a unit of electrical conductance, equal to one ampere per volt.

duce bandwidths up to 312.5 kbits/sec [Schill 2004]. This result was achieved in a round pool at a distance of approximately 2 meters. The latest demonstrations have proven that optical communication at 1 Gbit/sec through a 2-meter water pipe with up to 36 dB of narrow-beam extinction is possible [Hanson 2008]. Since it is difficult to perform such experiments in the ocean, most underwater optical propagation measurements are performed in a lab tank, leaving the efficacy of such systems in natural environments yet to be explored.

1.3 Principles of Underwater Acoustics Relevant to Communication

Acoustical transmission is more flexible than other approaches, as it can be deployed in a wide variety of configurations, including networks consisting of both mobile and stationary nodes. It is not, however, free of complexity. In fact, certain aspects of underwater acoustic communications are more difficult than those of RF terrestrial networks, especially high propagation delay. In general, underwater acoustic communications are influenced by transmission loss, bubbles, stratification, multipath propagation, Doppler spread, noise, and high propagation delay.

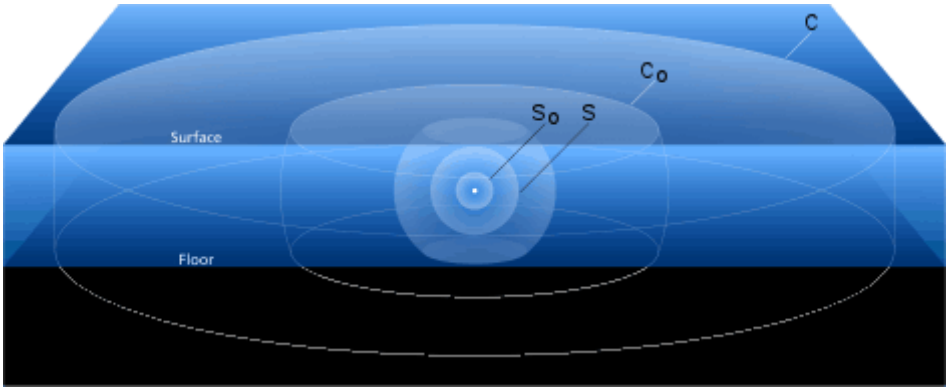

Figure 1-1: Spherical and cylindrical spreading. Sound generated by the sound source, shown as the white dot in the center, begins by spreading out uniformly in all directions. The intensity of the sound decreases rapidly as it spreads out from the sphere with radius s_0 to the one with s. The sound can no longer spread out uniformly once it reaches the surface and floor of the water, and begins to spread out cylindrically, radiating horizontally away from the source. The intensity of the sound decreases more slowly as it spreads from the cylinder with radius c_o to the one with c than when it spreads out from the sphere with radius s_0 to the one with s [URI 2008].

Transmission loss describes the weakening intensity of sound over a distance and is comprised of losses from both spreading and attenuation. Spreading loss is a geometrical effect that represents the weakening of sound as the wave moves outward from the source. It can be further classified as spherical spreading, cylindrical spreading, or a variant with properties somewhere between the two. Spherical spreading is omnidirectional, where the sound intensity decreases with the square of the range. Cylindrical spreading, on the other hand, takes place in horizontal channels, where the pressure of the sound varies inversely with the range [Urick 1996]. Figure 1-1 depicts the differences between the two types of spreading. Attenuation loss encompasses the effects of absorption, scattering, and leakage out of a sound channel [Urick 1996]. Absorption, a true loss of acoustic energy that results from the conversion of that energy into heat, accounts for the majority of attenuation. Marsh and Schulkin's empirical formula for the attenuation coefficient in sea water is often used for frequencies between 3 kHz and 0.5 MHz, while Thorp's formula better describes the attenuation of low frequency sounds, in the range of 100 Hz to 3 kHz [Brekhovskikh 2003]. The attenuation coefficient produced by both formulas is expressed in dB/km for a frequency f in kHz; however, Marsh and Schulkin's formula also requires values for the salinity and hydrostatic pressure of the body of water. Since attenuation increases rapidly with frequency, there exists an upper limit on the frequency used for a link of a given distance in a digital communication system.

Bubbles produced by breaking waves at the surface can influence the propagation of high frequency signals. No bubble-induced losses were discovered for waves produced with wind speeds of 6 m/s or less [Preisig 2006]. However, with faster wind speeds, losses increased as wind speed increased, with 20 dB loss reported for a wind speed of 10 m/s.

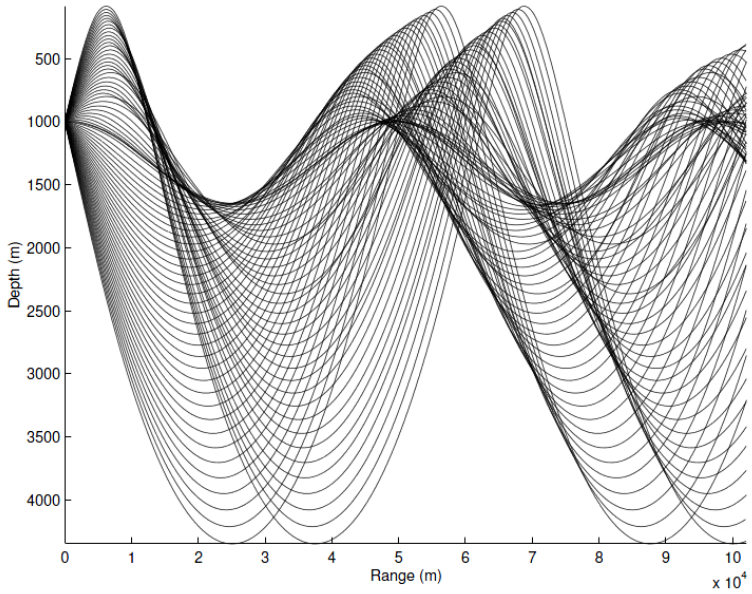

Figure 1-2: Shadow zones (white areas) created by the refraction of waves in deep water.

Stratification, the separation of a body of water into layers of similar densities, can also greatly impact the availability of an acoustic link. Fluctuations in the sound speed within a channel cause the refraction of signals, which in turn lead to shadow zones, areas nearly void of acoustic signal. This shadow zone phenomenon occurs in large bodies of water at a depth of 5000 m and distance of 100 km as well as in shallow regions of about 100 m in depth and 3 km across [Preisig 2006]. Figure 1-2 shows the refraction of waves in a deep water environment with a Munk sound velocity profile[2] [Munk 1974] and the shadow zones created by this propagation pattern.

[2] The sound velocity profile, sometimes called the sound speed profile, plots the velocity of sound as a function of depth. The velocity values are often derived from salinity, temperature, and depth measurements. The Munk profile is a canonical sound velocity profile that illustrates features typical of deep water environments.

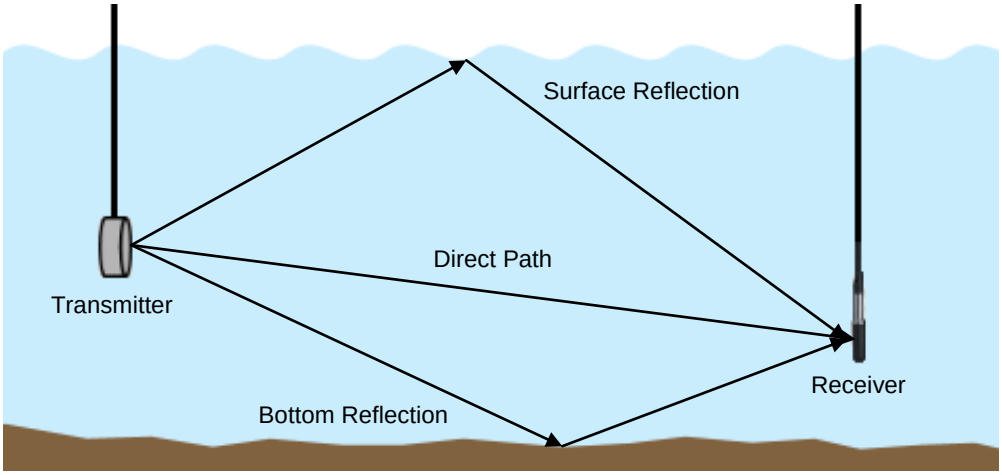

Figure 1-3: Multipath propagation in shallow water consists of the direct path and reflections from the surface and bottom.

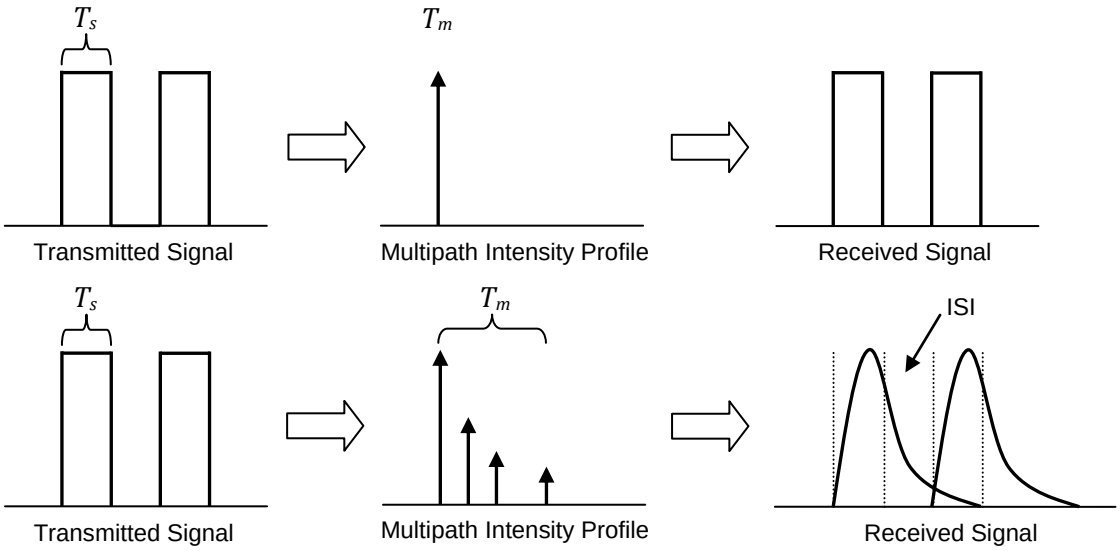

Figure 1-4: Channel-induced intersymbol interference. The top portion shows how a transmitted signal appears at the receiver after passing through a channel where the symbol time T_s is greater than the delay spread of the channel T_m. Since the channel represented here is ideal, it does not modify the transmitted signal in any way. The bottom portion shows how a transmitted signal becomes distorted when T_s is less than T_m. The symbol from one time slot "smears" into the following slot, resulting in channel-induced intersymbol interference.

Multipath propagation in an underwater acoustic network is the phenomenon where the acoustic signal will reach the receiver by two or more paths. It is prevalent in shallow water, where acoustic links are horizontal and spreading is considered to be almost entirely cylindrical, but it is not much of an issue in vertical channels. In shallow environments, some acoustic signals from the transmitter are reflected off the surface of the water and bottom of the channel before reaching the receiver, thus creating multipath propagation (see Figure 1-3). Delay spreads in shallow environments can last for several milliseconds. Unless complex adaptive filters are utilized, the duration of a communication system's symbol time must be greater than the delay spread of the channel in order to avoid intersymbol interference, or ISI (see Figure 1-4).

Shallow water channels exhibit temporal variation due to tides, a moving surface, and fluctuating amounts of water traffic. These changes significantly affect the impulse response of the channel, leading to potentially drastic differences in the estimates obtained over sub-second intervals. These rapid fluctuations in the channel lead to a short coherence time, the duration over which the channel's properties are essentially invariant. When viewed from the from the frequency domain, the channel exhibits a wide Doppler spread. In time-varying environments, receivers need to repeatedly estimate and quantify changes in the environment to keep an acoustic link functional. Such systems typically employ a decision feedback equalizer (DFE) [Stojanovic 2003].

Several papers assume ambient noise in underwater channels is white Gaussian noise; however, this assumption is now known to be incorrect. Though it is difficult to precisely derive a formula for noise, empirical formulas that give the power spectral density of noise caused by turbulence, shipping, waves, and thermal noise do exist [Coates 1989]. These formulas are generalized and do not account for extreme conditions in certain environments. For example, during some of the underwater acoustic experiments conducted in the Hudson River off the campus of

Stevens Institute of Technology, the sounds and physical vibrations produced a pile-driver in a nearby construction site obliterated all signals up to 100 kHz. In addition, in shallow areas with temperate and tropical waters, such as Kaneohe Bay, O'ahu in the Hawaiian Islands, snapping shrimp produce a tremendous amount of noise. The peak-to-peak source level of a snap can reach 185 dB re: 1 μPa, and the frequency spectrum of the snap is extremely broad, with energy beyond 200 kHz [Au 1998]. Thus, the noise present in some of these shallow environments can render underwater acoustic communication impossible at times.

The speed of sound in water is approximately 1500 m/s, which is five orders of magnitude less than the speed of light, or 3×10^8 m/s. This slow speed leads to long propagation delays of about 0.67 km/s and comparatively large motion-induced Doppler shift [Partan 2006]. Some researchers believe that the high delay variance is more detrimental to the design of efficient protocols than the propagation delay, since it precludes accurately estimating round trip time [Akyildiz 2005]. Others state that although the underwater acoustic channel is time-varying, propagation delays can be estimated and are stable enough to use within network protocols [Partan 2006].

1.4 Motivation for Research

The initial goal of this research was to create an adaptive communication system that would allow a set of nodes to be deployed into any body of water; the nodes would then sense channel conditions and adapt. Adaptation would be both initial (immediately upon deployment) and ongoing, hopefully providing increased overall system efficiency compared to non-adaptive systems. It soon became apparent that the goal was overly ambitious because a fully adaptive system would have to be able to change not only physical parameters (such as frequency and modulation technique) but also link-level and routing-level operation. Several other facts also became apparent during the literature search and formative stages of this work, including that (1) there was no satisfactory adaptive experimental platform, (2) there was no satisfactory simulation

platform, and (3) there was limited information about how to characterize an acoustic underwater channel and how to apply results of the characterization to the design of adaptive digital communication. Accordingly, this research set out to address these three shortcomings. The design of a multi-node system, adaptive at all levels, is left as future work.

Understanding how the channel affects signals is critical to the design of an efficient communication system. At the physical layer, thorough analysis of the channel can suggest the type of modulation, symbol duration, required source level, length of packets, and the type and rate of equalization. In this dissertation, data collected during channel sounding experiments and the methods used to process that data become the backbone of the entire effort. The resulting impulse response estimates from both a controlled, time-invariant indoor test environment and the complex, time-variant Hudson River estuary are processed from the time and frequency domains to fully describe the properties of the channels. Analysis of the Hudson is a unique extension to previous work in this area because, at 3 meters, it is shallower than other channels that have been studied. In addition, unlike many publications or even textbooks, this dissertation clearly shows how to compute the various channel characterization functions in MATLAB, describes the limitations of the experimental setup, and offers solutions to mitigate the effects of the limitations, all of which are valuable to the experimentalist.

Even the most recent efforts in simulating the underwater channel have fundamental limitations. In the World Ocean Simulation System (WOSS), the authors use the BELLHOP model to derive the Signal-to-Interference-plus-Noise Ratio (SINR) from which they compute the bit error rate (BER) for a given modulation [Guerra 2009]. ISI caused by multipath propagation, common in shallow water channels, is not taken into account in this model. Therefore, in this dissertation, a new model based on channel measurements is introduced. Through a mathematical process called convolution, the impulse response estimates obtained from the channel sound-

ing experiment afford a more accurate simulation of how the channel distorts an acoustic signal. The original modulated waveform is "mixed" with the channel and sent to a receiver implemented fully in software, where the actual BER is computed.

The last portion of the research combines the channel estimation techniques and software defined radio aspects of the simulation to produce a pseudo real-time underwater acoustic modem – the Softwater Modem [Borowski 2009]. Unlike existing efforts implemented purely in software [GNU 2010; Sailer 2000], this modem is packet-based and mitigates channel-induced ISI through means of a zero-forcing equalizer. It has been proven to work in an office test environment that exhibits a significant amount of multipath propagation where the other software defined radio (SDR) platforms have failed. Furthermore, the Softwater Modem allows the user to configure numerous parameters including the carrier frequency, symbol rate, packet detection threshold, and number of parity bytes used in the Reed-Solomon error correcting codes.

1.5 Dissertation Outline

The dissertation is divided into the following topics: channel characterization, software modem design and implementation, and simulation of the underwater channel and physical layer. Chapter 2 discusses the characterization of two very different channels, those being the Hudson River estuary adjacent to the Stevens Institute of Technology campus and an office test setup comprised of a plastic tub filled with water. In both cases, successive impulse response estimates are processed to generate the channel's scattering function, from which all other characterization functions are derived – multipath intensity profile, spaced-frequency correlation function, Doppler power spectrum, and spaced-time correlation function. Amplitude fluctuations of the components in the multipath intensity profile are fit to the Rayleigh, Rice, and Nakagami-*m* distributions often used to model fading channels. Values obtained in the analysis of the characterization func-

tions yield estimates of the delay spread and coherence time of the channel as well as the severity of fading, all of which greatly influence the design of a digital communication system.

Chapter 3 discusses how to more accurately implement channel and physical (PHY) layer models in the OMNeT++ discrete-event simulator. In order to simulate a time-variant channel, an impulse response is chosen from the database at random and convolved with the waveform representing the modulated packet. The resulting signal, which now contains channel-induced distortion, is passed to a receiver block, where it is demodulated and its bit BER is computed. The signal processing blocks, exported as a shared library, are performed in MATLAB and called from within the OMNeT++ simulation. Aspects of the simulation's architecture are described in detail. In addition, the experiment used to validate the accuracy of the estimated BERs is expounded.

Chapter 4 presents the design of the Softwater Modem, an acoustic modem fully implemented in software. The modem allows users to easily deploy network applications that make use of the sockets interface in Linux. The system is comprised of three layers of user space applications which pass data among themselves via UDP sockets and the TUN kernel space character device. The modem portion of the system is a Java application built as a series of stages that handle various aspects of signal processing. The details of the algorithms, computational performance, and expected BER in an additive white Gaussian noise (AWGN) channel are presented.

Chapter 5 provides an evaluation of the thesis and a summary of the contributions to the field. The appendix contains the most relevant sections of source code for channel characterization routines, the acoustic modem, and the simulation of the channel and physical layers.

Chapter 2
Shallow Water Channel Characterization

2.1 Introduction

Over the past three decades there has been much research in underwater acoustic communication. While recent years have seen the shift from noncoherent point-to-point communication to developing networks based on coherent reception techniques [Chitre 2008], relatively few papers have focused on the fundamental process of characterizing the underwater acoustic channel. There is no typical underwater channel [Preisig 2006]; each environment possesses different characteristics that will affect the performance of a digital communication system. Therefore, it is necessary to study numerous underwater acoustic channels to gain a quantitative understanding of their properties, as had been done in the RF research community.

The shallow water acoustic communication channel can be classified as a multipath fading channel. It generally exhibits a long multipath delay spread, which can lead to intersymbol interference (ISI) if the spread exceeds the symbol time of the communication system, as shown in Figure 1-4. The channel typically has significant Doppler spread in the frequency domain, or short coherence time when viewed in the time domain. Communication in estuarine environments is often complicated by ambient noise from both shipping vessels and activity on the surrounding land, such as that in a construction site. The depth of water in an estuary is extremely shallow; thus, the effects of surface waves and wind speed on an underwater acoustic signal are more apparent. In addition, estuaries are prone to stratification, the separation of a body of water into layers of similar densities. Fluctuations in the sound speed within a water column cause the refraction of signals, which in turn leads to shadow zones – areas nearly void of acoustic signal, as shown in Figure 1-2.

Measuring and analyzing a channel's parameters is a necessary step for the design of a successful communication system. Moreover, numerous channel measurements are required to build up a database of underwater environments that helps the research community create a model for more realistic simulation of the physical as well as higher layer protocols within a communication system's network stack. Chapter 3 describes how to use the channel measurements from this chapter in an underwater network simulation implemented in OMNeT++ and MATLAB.

2.2 Description of Signals and Functions

The impulse response provides all the information necessary for channel characterization. In order to estimate the channel's impulse response, two signals are needed – a known reference or "sounding" signal that covers the frequency band of interest and the received signal, that is, one that has passed through the channel and has undergone some degradation. The impulse response of a system taken repeatedly over time affords the ability to produce the scattering function, which in turn can be made to show different channel correlation functions via Fourier transforms.

2.2.1 Sounding Signal

Since a unit impulse is an unrealizable signal, engineers choose a practical input signal to the system that will lead to an accurate estimation of the system's impulse response. Several signals are frequently employed, those being a LFM (linear frequency modulated) chirp [Dessalermos 2005], a HFM (hyperbolic frequency modulated) chirp [Michalopoulou 2001], white noise [Schomer 1972], and a DSSS BPSK (direct sequence spread spectrum binary phase shift keying) signal [Chitre 2004; Dessalermos 2005]. While it is debatable which sounding signal is the best, all except for the HFM chirp possess acceptable autocorrelation properties as to closely approx-

imate the Dirac delta function[3]. Autocorrelation that approximates the Dirac delta function is the test of goodness for a sounding signal. In general, the best result is obtained when the sounding signal spans the entire bandwidth from 0 Hz up to the Nyquist[4] frequency (except in the case of the HFM chirp). However, because of the potential ambient noise in lower frequency bands and the lack of low frequency response in most emitters, the starting frequency for the sounding signal is sometimes chosen to be several kHz. It should be noted, though, the increasing the lower frequency will decrease the autocorrelation function's likeness to the Dirac delta function, as seen in Figures 2-1 through 2-5.

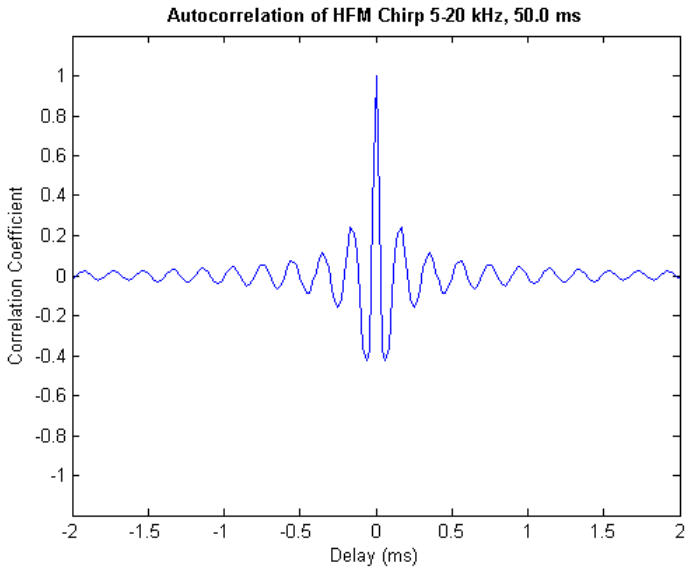

Figure 2-1: Autocorrelation of HFM chirp 5-20 kHz, 50.0 ms.

[3] In the context of signal processing, the Dirac delta function is referred to as the unit impulse, or a signal having infinite amplitude, zero width, and unit area.
[4] The Nyquist frequency is half the sampling frequency of a discrete signal processing system.

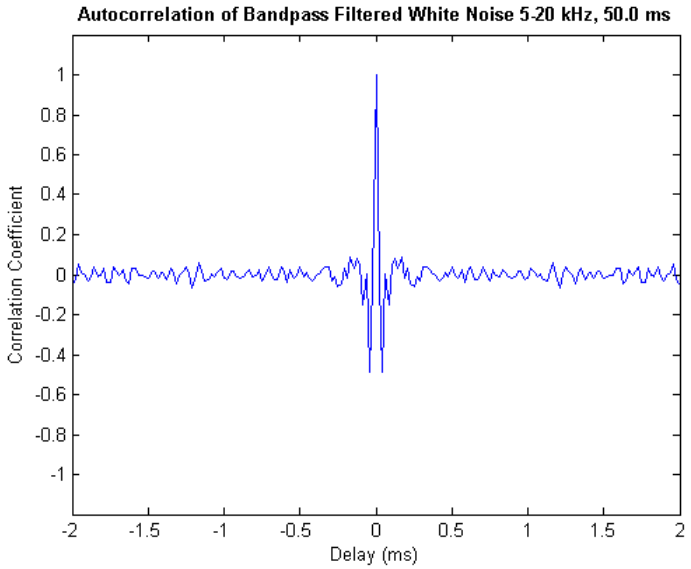

Figure 2-2: Autocorrelation of bandpass filtered white noise 5-20 kHz, 50.0 ms.

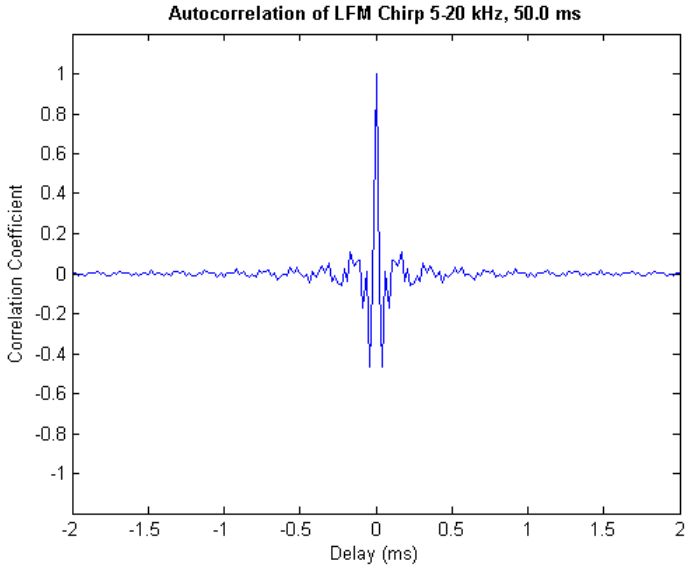

Figure 2-3: Autocorrelation of LFM chirp 5-20 kHz, 50.0 ms.

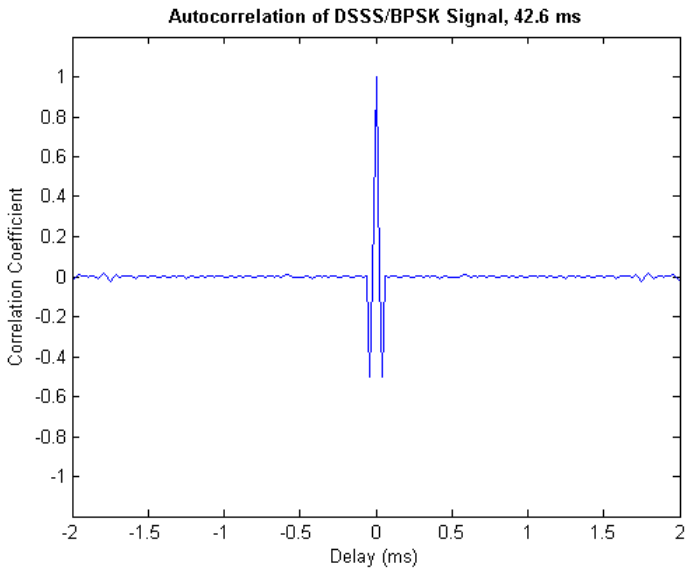

Figure 2-4: Autocorrelation of DSSS/BPSK signal 5-20 kHz, 42.6 ms.

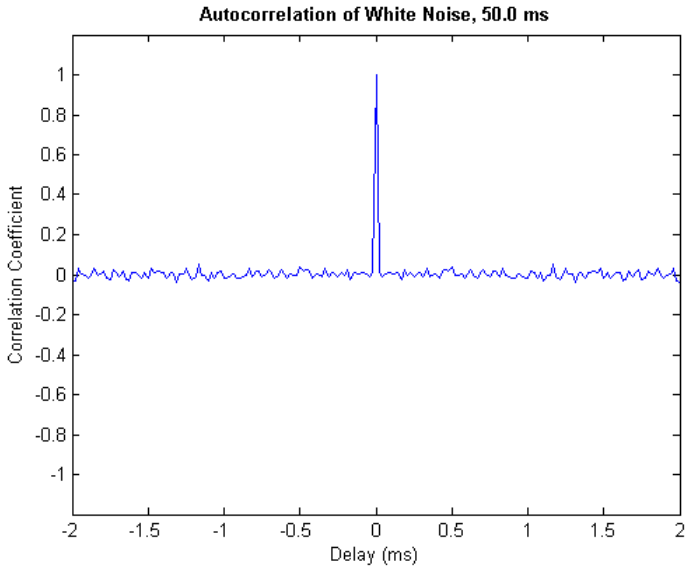

Figure 2-5: Autocorrelation of white noise, 50.0 ms.

Figures 2-1 through 2-5 show the autocorrelation of five sounding signals that have been used by researchers to estimate a channel's impulse response, in increasing order of similarity to

the Dirac delta function. The autocorrelation of the HFM chirp in Figure 2-1 exhibits significant side lobes which will reduce the accuracy of the impulse response estimate. The autocorrelation of the bandpass filtered white noise in Figure 2-3 and LFM chirp signal in Figure 2-4 are roughly the same, with smaller side lobes than the HFM chirp. The LFM chirp signal, however, possesses less fluctuation in the correlation coefficient at lags further from the center. While the autocorrelation function of both types of chirp signals exhibits side lobes, it is stated that in horizontal shallow water, where the channel frequency spreading is low if the transmitter and receiver remain fixed, chirp signals are well known to possess the best delay (range) sensitivity [Cook 1998].

Figure 2-4 shows the autocorrelation of the DSSS/BPSK signal generated from the MSE/AO maximum length sequence of code length 511 found in [Kärkkäinen 2007] with a 12 kHz carrier. The MSE/AO criteria emphasize autocorrelation properties, and the autocorrelation of just the sequence itself is indeed an impulse. However, when mixed with a carrier, the autocorrelation of the resulting BPSK signal possesses one lobe of negative correlation on each side of the impulse.

The noise signal in Figure 2-5 most closely approaches the Dirac delta function. It is supposed to represent white noise, but since the duration of the signal is so short, not all frequency components have been given equal representation, making it improper to fully classify the noise as “white”. The autocorrelation of this signal fluctuates minimally at lags off to the sides, and it does not exhibit any side lobes. Moreover, increasing the length of the signal will further improve the autocorrelation function, as there is more randomness in a longer signal. Figure 2-6 illustrates how a 1-second clip of white noise exhibits a much cleaner autocorrelation function than a 10-ms clip.

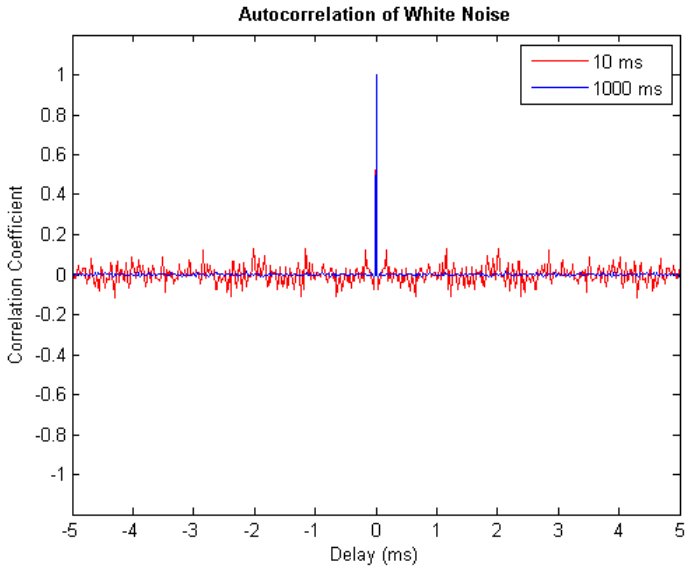

Figure 2-6: Autocorrelation of white noise signals, 10 ms and 1 s.

To make a fair comparison among the autocorrelation functions of all five signals, each signal was designed to be about the same length and use approximately the same bandwidth. The 5-20 kHz frequency band was utilized for two reasons. 5 kHz was used as the lower bound because, as already stated, many emitters are incapable of producing low frequency signals. 20 kHz was used as the upper bound so that a typical PC sound card can be used to play the sounding signals. The generation of the chirp signals is straightforward. See Appendix A for MATLAB code. Generating a DSSS/BPSK digitally requires the designer to pay careful attention to the sampling rate of the playback hardware. In particular, each symbol in the maximum length sequence must be multiplied by an integer number of samples in each sampling period. This implies that the symbol rate must divide the sampling rate, which thus limits the number of symbol rates that can be used. The bandwidth of the main lobe of the resulting BPSK signal is

$$BW(\text{Hz}) = \frac{2}{T_s} = 2 \times R_s, \tag{2.1}$$

where T_s represents the symbol time and R_s is the symbol rate. Each symbol must contain the same number of periods of the carrier wave so that the phase shifts always occur 180° apart. Therefore, the carrier frequency F_c must be a multiple of the symbol rate in the range $1 \leq F_c \leq$ Nyquist frequency.

The DSSS/BPSK signal described in this section was created with a 12 kHz carrier frequency and a symbol rate 12 ksamples/sec. It occupies the entire bandwidth below 24 kHz. The unfiltered white noise also covers the same band. The power spectral density of all five sounding signals is shown in Figure 2-7.

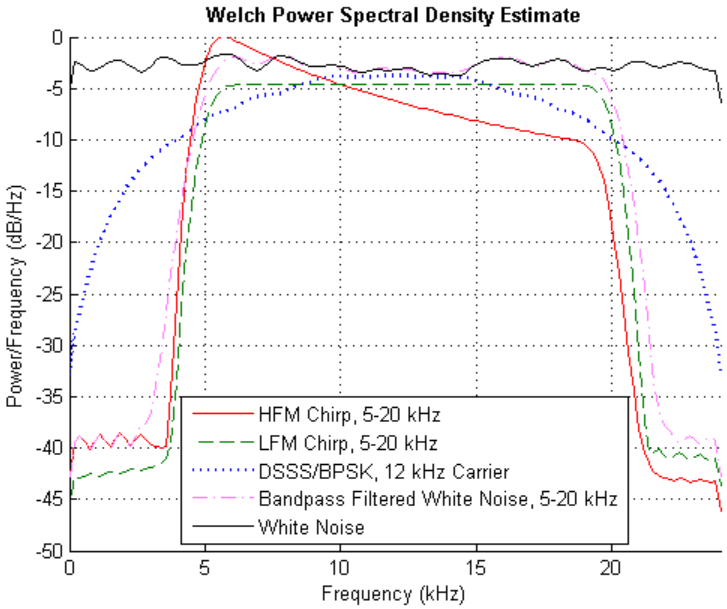

Figure 2-7: Power spectral density of various channel sounding signals.

In order to produce the channel's scattering function with enough resolution (wide range of frequencies), the channel needs to be sounded as often as possible per second. There are, however, some constraints on the length of the sounding signal. The length of the signal must not exceed the coherence time of the channel. This restriction exists because the channel's characteristics need to remain nearly constant over the duration of the sounding in order to capture a single undistorted impulse response estimate. When played consecutively with no silence between

repetitions, the length of the sounding signal must exceed the channel's multipath spread. If there is no silence between soundings, all multipath arrivals must appear within the duration of a sounding so that late arrivals do not overlap with the arrivals appearing from subsequent repetitions. This restriction can be eliminated if a shorter chirp signal can be used; however, it must be followed by a period of silence long enough to allow all multipath propagations to taper off. Though this method produces correct estimates, it results in correlations with reduced SNR (signal-to-noise ratio), as there are less samples in each sounding to work with. Therefore, there is no benefit to using gaps of silence.

Figures 2-8 through 2-10 illustrate why the restrictions on the length of a sounding signal exist. In all three sections, the multipath spread of the channel is 50 ms. Figure 2-8 depicts correct channel sounding with the use of a 60 ms signal that exceeds the 50 ms multipath spread. Figure 2-9 also depicts correct, though not ideal, channel sounding with the use of a 10 ms signal followed by 50 ms of silence that guarantees all arrivals taper off before the subsequent sounding. Figure 2-10 shows what happens when the channel is sounded by a signal that is 20 ms shorter than the multipath spread. The main arrival from the second chirp signal appears before all the multipath propagations from the first taper off. Therefore, cross-correlation produces large peaks that appear in the middle of the intensity profile for the first chirp, resulting in smeared output.

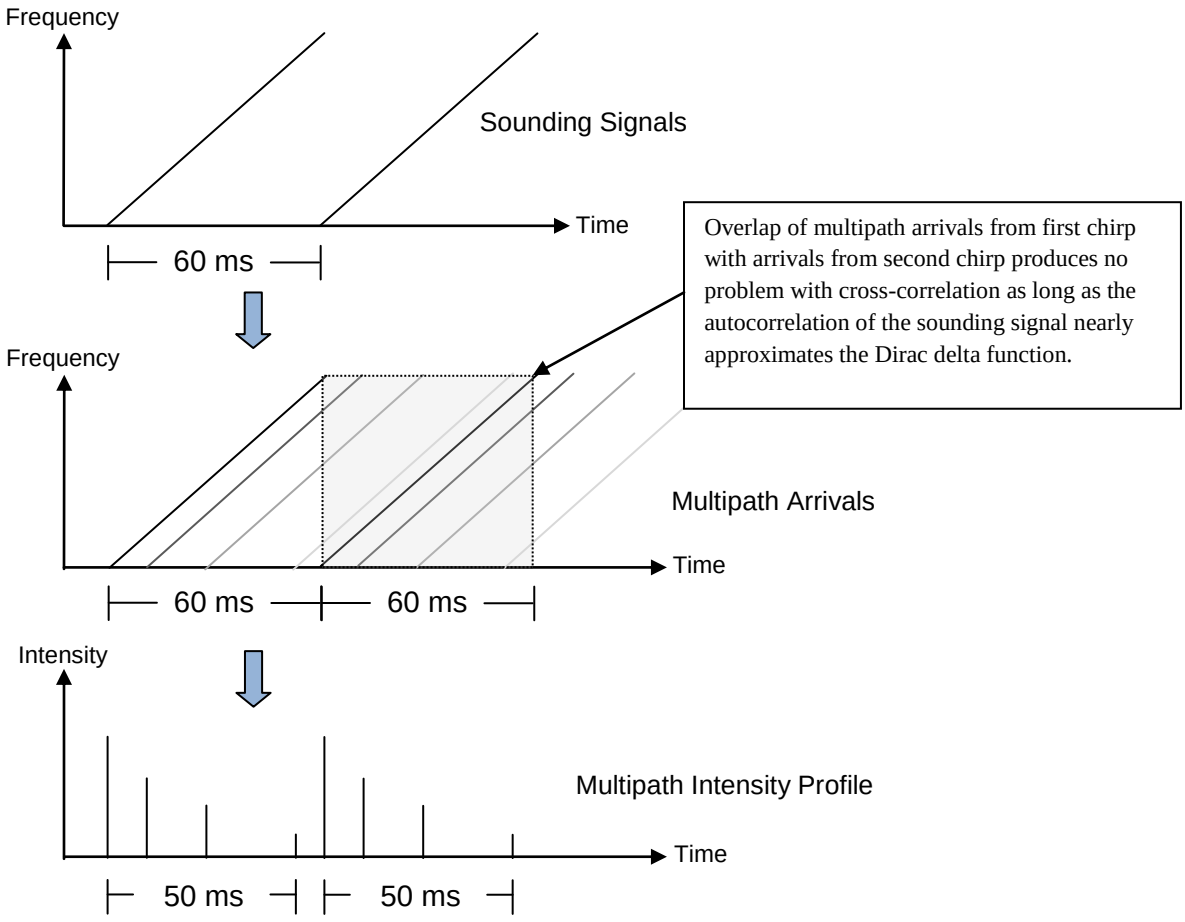

Figure 2-8: Ideal channel sounding. Length of sounding signal (60 ms) > multipath spread (50 ms).

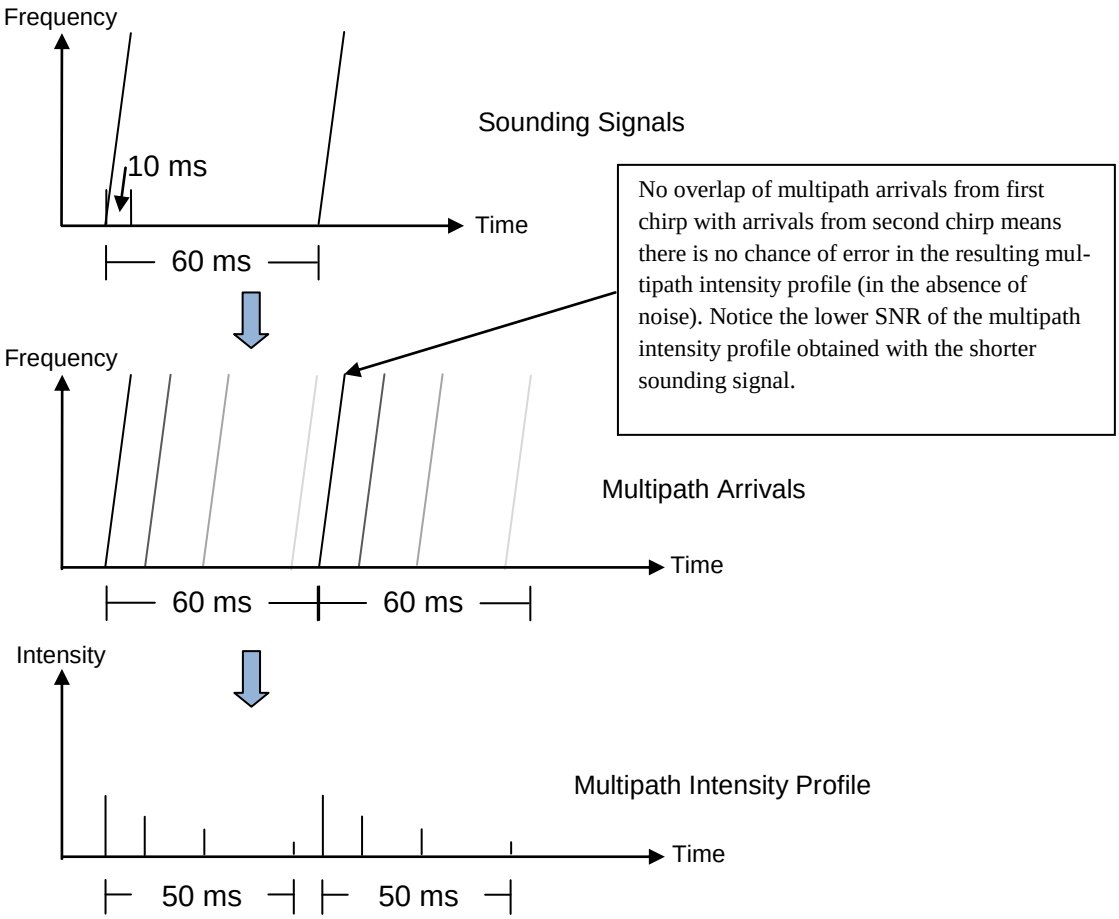

Figure 2-9: Correct channel sounding. Length of sounding signal (10 ms) plus period of silence (50 ms) > multipath spread (50 ms).

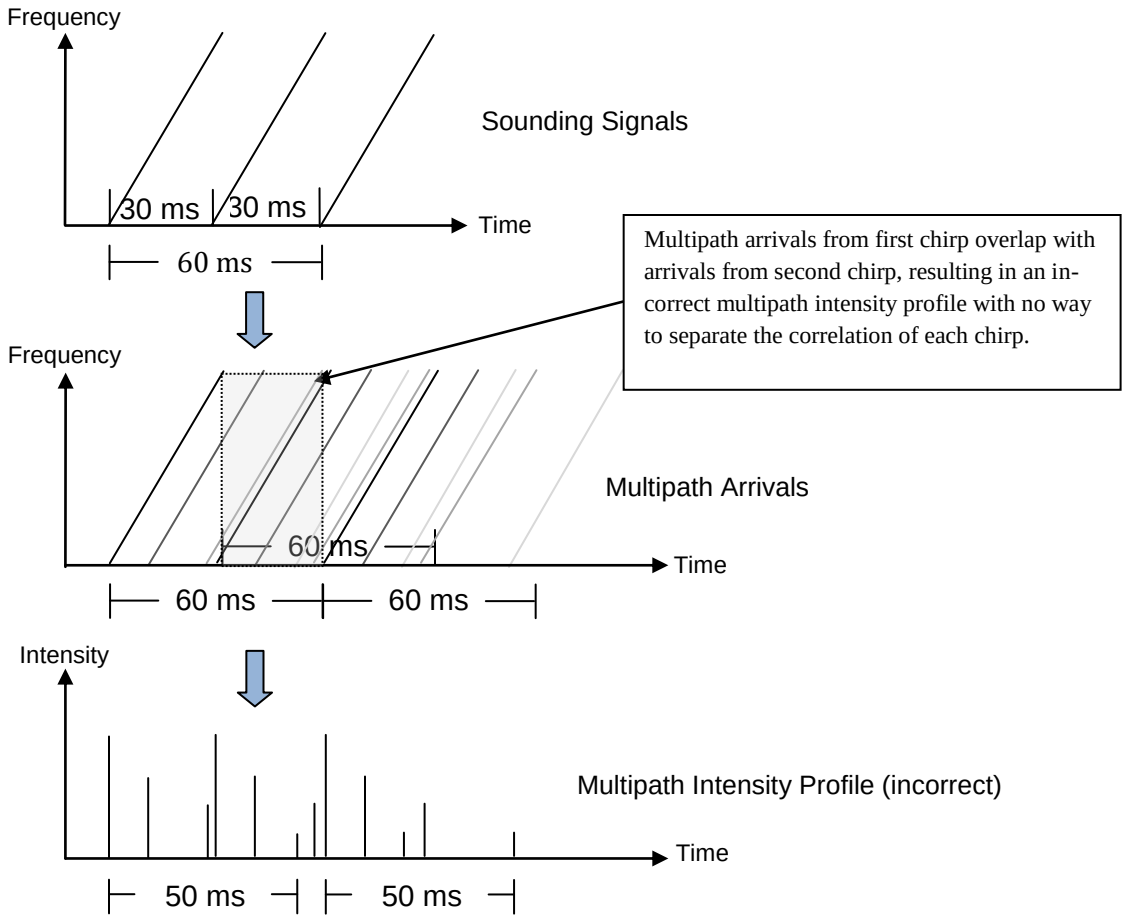

Figure 2-10: Incorrect channel sounding. Length of sounding signal (30 ms) < multipath spread (50 ms).

The analysis of all the sounding signals presented in this section leads to the following conclusions. A long clip of white noise exhibits the best autocorrelation properties. A 50-ms clip still possesses good autocorrelation, but since the emitter cannot produce low frequencies, the white noise signal essentially becomes bandpass-filtered. The differences between the bandpass-filtered white noise of Figure 2-2 and the LFM chirp signal of Figure 2-3 are minor, with the LFM chirp signal being slightly better. In addition, a LFM chirp signal is much less likely to appear randomly in any environment than a clip of white noise. Therefore, the LFM chirp signal was used as the sounding signal in the experiments presented later in this chapter.

2.2.2 Impulse Response

The impulse response $h(t)$ of a linear system is the response when the input to the system is equal to the delta function, or unit impulse. The response of a linear system $y(t)$ to an arbitrary input signal $x(t)$ is found by convolving $x(t)$ with $h(t)$, as in

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{2.2}$$

or

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(t-\tau)h(t)d\tau, \tag{2.3}$$

where τ is the time at which the impulse was applied. Since the system is causal, the lower limit of the integral can be changed to 0, enabling $y(t)$ to be expressed as

$$y(t) = \int_{0}^{\infty} x(t-\tau)h(t)d\tau \tag{2.4}$$

or in discrete form, which is more practical when working with digital communication systems, as

$$y[t] = \sum_{t=0}^{M-1} x[t-\tau]h[t], \tag{2.5}$$

where M is the maximum of the number of samples in signals x and h, and the shorter signal is zero-padded to the length of the longer one.

When the system's input and output signals are known, the process of solving for the impulse response becomes the inverse of convolution, or deconvolution, which actually has no direction definition in the time domain [Riad 1986]. However, in the frequency domain, deconvolution is represented as

$$H(j\omega) = Y(j\omega) / X(j\omega), \tag{2.6}$$

where $X(j\omega)$, $Y(j\omega)$, and $H(j\omega)$ are the frequency-domain forms of $x(t)$, $y(t)$, and $h(t)$.

In practical situations, $x(t)$ is a signal with limited bandwidth, causing $X(j\omega)$ to be close or equal to zero at some frequencies. In this case deconvolution results in an unstable filter. Wiener deconvolution can be used to improve the accuracy of the resulting impulse response if the power spectral density of the noise in the channel is known. Another solution, though computationally expensive, capitalizes on the fact that convolution is the equivalent to matrix multiplication of the form of Ax=b, where A is a matrix that represents the input to the system, b represents the output, and x is the system's impulse response. This solution approaches deconvolution as a pseudoinverse[5] problem. This method experiences problems in the presence of noise, amplifying it proportionally to the inverse of the singular values of A. Performing the pseudoinverse operation with a tolerance close to the smallest singular value produces estimates with an acceptable rate of error.

Because of the complications associated with deconvolution, cross-correlation, a measure of the likeness of two signals, is the method most often used to determine a system's impulse response. In the discrete case, the cross-correlation of signals $x[t]$ and $y[t]$ is defined by

$$r_{xy}[t] = \sum_{n=0}^{M-t-1} x[n+t]y^*[n], \tag{2.7}$$

where M is the maximum of the number of samples in signals x and y, the shorter signal is zero-padded to the length of the longer one, and the superscript asterisk indicates the complex conjugate. Cross-correlation has the obvious benefit of computational simplicity when compared to any method of deconvolution, especially when it is performed in the frequency domain using the fast Fourier transform.

Assuming the channel is wide-sense stationary (WSS) uncorrelated scattering (US)

[5] Pseudoinverse is the standard definition for the inverse of a matrix if the matrix is not square or singular. A square matrix has an equal number of rows and columns. A singular matrix is a square matrix that does not have a matrix inverse. A matrix is singular iff its determinant is 0.

(combined, WSSUS) [Bello 1963], the time-varying complex-valued low-pass impulse response $c(\tau; t)$ of the underwater channel is captured via the following procedure:

1. A sounding signal is repeatedly transmitted through the channel and recorded.
2. The imaginary part of the reference sounding signal is obtained via the Hilbert transform.
3. The received signal is cross-correlated with the complex conjugate of the reference signal.

Additional steps are sometimes necessary during field tests, as seen in Section 2.5.4. Figure 2-11 provides MATLAB code that produces the time-varying impulse response of a channel.

```
%% Computes impulse response for spreads up to 10 ms.

seconds = 0.010;
impulseResponse = zeros(numOfImpulseResponses, seconds*samplingRate);
for i = 1:numOfImpulseResponses
    snip = recordedSignal((i-1)*referenceSamples+1:i*referenceSamples);
    temp = fftshift(xcorr(snip, conj(referenceSignal)));
    impulseResponse(i,:) = temp(1:seconds*samplingRate);
end

% Normalize output.
maxVal = max(max(abs(impulseResponse)));
impulseResponse = impulseResponse / maxVal;
```

Figure 2-11: MATLAB code to produce the time-varying impulse response of a channel.

In summary, successive impulse response estimates of a linear system can be obtained via cross-correlation. The procedure is practical, efficient, and accurate. The impulse response estimates obtained with this method fully describe the channel. These estimates can be processed to provide various statistical views of the channel, presented in subsequent sections of this chapter.

2.2.3 Scattering Function

The scattering function gives the average power output of the channel as a function of time delay τ and Doppler frequency λ and is the basis for computing the remainder of the channel characterization functions described in this chapter. Assuming that $c(\tau; t)$ is a WSS random process, the auto-correlation A of $c(\tau; t)$ is defined as

$$A_c(\tau_1, \tau_2; \Delta t) = E[c^*(\tau_1; t)c(\tau_2; t + \Delta t)], \tag{2.8}$$

where * denotes the complex conjugate. Further assuming uncorrelated scattering – that the attenuation and phase shift of the channel at two separate path delays τ_1 and τ_2 are uncorrelated, the WSS assumption is strengthened to WSSUS, and

$$E[c^*(\tau_1; t)c(\tau_2; t + \Delta t)] = A_c(\tau_1; \Delta t)\delta[\tau_1 - \tau_2] \triangleq A_c(\tau; \Delta t). \tag{2.9}$$

The scattering function is defined as the Fourier transform of $A_c(\tau; \Delta t)$ with respect to the Δt parameter [Goldsmith 2005], as in

$$S_c(\tau;\ \lambda) = \int_{-\infty}^{\infty} A_c(\tau;\ \Delta t)e^{-j2\pi\lambda\Delta t}d\Delta t. \tag{2.10}$$

```
%% Computes scattering function.
tauSamples = length(impulseResponse);
lambdaSamples = numOfImpulseResponses;

scatteringFunction = zeros(tauSamples, lambdaSamples * 2 - 1);
for i = 1:tauSamples
     temp = fftshift(fft(xcorr(impulseResponse(:,i))));
     scatteringFunction(i,:) = temp(end:-1:1);
end

% Normalize output.
maxVal = max(max(abs(scatteringFunction)));
scatteringFunction = scatteringFunction / maxVal;

% Compute range of tau and lambda to represent time delay and Doppler
% frequency for the axes.
[tauSamples lambdaSamples] = size(scatteringFunction(1:end,:));
tau = (0:tauSamples-1) / samplingRate * 1000 - 1;
lowerBound = floor(lambdaSamples / 2);
upperBound = floor(lambdaSamples / 2);
if (mod(lambdaSamples, 2) == 0)
    upperBound = upperBound - 1;
end
f = -lowerBound:upperBound;
lambda = (1 / chirpSignalSeconds / 2) * f / lowerBound;
```

Figure 2-12: MATLAB code to produce the scattering function of a channel.

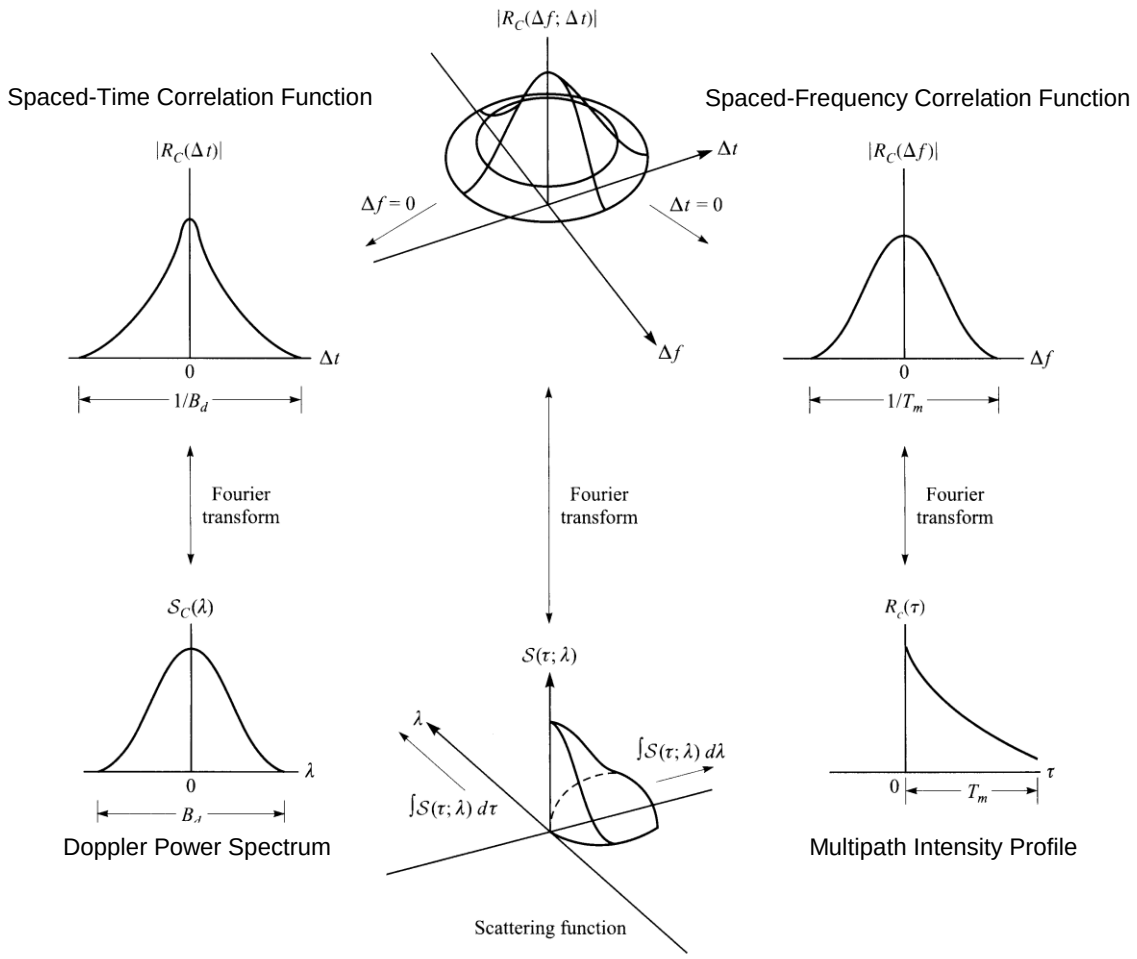

Figure 2-13: Relationships between scattering function and derived correlation functions and power spectra [Proakis 2008].

Figure 2-12 provides MATLAB code that produces the scattering function of a channel. Figure 2-13 shows the relationships between the scattering function and its derived views, which are explained in greater detail in Sections 2.2.4 – 2.2.7.

2.2.4 Multipath Intensity Profile

The multipath intensity profile (MIP) or power delay profile $P(\tau)$ gives the average power output as a function of time delay τ. It is computed by summing the power levels over the λ values of the scattering function, as in

$$P(\tau) = \int S_c(\tau;\, \lambda) d\lambda \,. \tag{2.11}$$

The MIP represents the delay spread of the channel. In general, an underwater multipath channel causes a transmitted pulse to arrive at the receiver as distinct components spread out over time. In digital communication systems without equalization, the length of this delay spread places a lower bound on the duration of a symbol T_s, or an upper bound on the data rate of the system, that must be used in order to avoid channel-induced ISI. Figure 2-14 provides MATLAB code to compute the MIP from the scattering function obtained from the code in Figure 2-12.

```
%% Computes multipath intensity profile.
mip = sum(abs(scatteringFunction'));

% Compute range of tau in milliseconds to represent time delay.
len = length(mip);
tau = (0:len-1) * 1000/samplingRate - 1;

% Normalize output.
mip = mip / max(mip);
```

Figure 2-14: MATLAB code to produce the multipath intensity profile of a channel.

The mean excess delay and rms delay spreads of the MIP [Goldsmith 2005] are defined as

$$mean\ excess\ delay\ \mu_{T_m} = \frac{\int \tau P(\tau) d\tau}{\int P(\tau) d\tau} \tag{2.12}$$

and

$$rms\ delay\ \sigma_{T_m} = \sqrt{\frac{\int \left(\tau - \mu_{T_m}\right)^2 P(\tau) d\tau}{\int P(\tau) d\tau}}, \tag{2.13}$$

where τ is the time delay of the multipath component within the MIP $P(\tau)$. The threshold level might be set to -20 dB for computation of the mean excess delay and rms delay spreads, while the maximum excess delay is typically computed for multipath components within 10 dB of the maximum level [Rappaport 2002].

2.2.5 Spaced-Frequency Correlation Function

The Fourier transform of the MIP yields the spaced-frequency correlation function (SFCF), which provides a measure of the frequency coherence of the channel. This function indicates the coherence bandwidth of the channel, which is a statistical measure of the range of frequencies over which the channel passes all spectral components with approximately equal gain and linear phase [Sklar 2001]. The SFCF essentially provides the same information as the MIP, except that the SFCF describes the view from the frequency domain. If the data rate of the communication system requires more bandwidth than the coherence bandwidth of the channel, frequency-selective fading, another name for channel-induced ISI, would occur. On the other hand, if the modulation bandwidth is less than the coherence bandwidth, frequency-nonselective or flat fading would occur. Figure 2-15 provides the MATLAB code to produce the SFCF from the MIP of a channel.

```
%% Computes spaced-frequency correlation function.
sfcf = abs(fftshift(fft(mip)));

% Normalize output.
sfcf = sfcf / max(sfcf);

% Compute range of frequencies for x-axis.
numPoints = length(sfcf);
lowerBound = floor(numPoints / 2);
upperBound = lowerBound;
if (mod(lambdaSamples, 2) == 0)
    upperBound = upperBound - 1;
end
f = -lowerBound:upperBound;
freq = (nyquistFreq / 2) * f / lowerBound;
```

Figure 2-15: MATLAB code to produce the spaced-frequency correlation function of a channel.

2.2.6 Doppler Power Spectrum

The Doppler power spectrum provides the signal intensity as a function of the Doppler frequency λ. It is found by summing the power of spectral components over the time delay τ of the scattering function, as seen in

$$P(\lambda) = \int S_c(\tau;\ \lambda)d\tau\,. \tag{2.14}$$

The range of frequencies over which the Doppler power spectrum is essentially nonzero is known as the Doppler spread of the channel. By replacing τ with λ, Equations (2.12) and (2.13) can be reapplied to calculate the Doppler shift and spread. The method, described in [Dessalermos 2005], provides the average and rms delay spreads in Hz, as in

$$shift_{\mathrm{Hz}} = \frac{\int \lambda P(\lambda)d\lambda}{\int P(\lambda)d\lambda} \tag{2.15}$$

and

$$spread_{\mathrm{Hz}} = \sqrt{\frac{\int (\lambda - shift)^2 P(\lambda)d\lambda}{\int P(\lambda)d\lambda}}, \tag{2.16}$$

where λ is the Doppler frequency in Hz and $P(\lambda)$ is the power of the spectral component at frequency λ. Figure 2-16 provides the MATLAB code to produce the Doppler power spectrum of a channel.

```
%% Computes Doppler power spectrum.
dps = sum(abs(scatteringFunction));

% Compute range of frequencies for x-axis.
numPoints = length(dps);
lowerBound = floor(numPoints / 2);
upperBound = lowerBound;
if (mod(lambdaSamples, 2) == 0)
    upperBound = upperBound - 1;
end
f = -lowerBound:upperBound;
lambda = (1 / chirpSignalSeconds / 2) * f / lowerBound;

% Calculate Doppler shift and spread.
sumValue = sum(dps);
overallShift = sum(lambda .* dps) / sumValue;
overallSpread = sqrt(sum((lambda - overallShift).^2 .* dps) / sumValue);

% Smooth and normalize Doppler power spectrum before plotting.
dps = smooth(dps, 3);
dps = dps / max(dps);
```

Figure 2-16: MATLAB code to produce the Doppler power spectrum of a channel.

If the Doppler spread $f_d > W$, where W is the bandwidth required for modulation, the channel is referred to as fast fading, since the channel's conditions change within the duration of a single symbol. Such channels typically require noncoherent or differentially coherent modulation [Sklar 2001]. If $W > f_d$, the channel is referred to as slow fading. If the baseband signal bandwidth is much greater than f_d, the effects of Doppler spread at the receiver are negligible [Rappaport 2002].

2.2.7 Spaced-Time Correlation Function

The Fourier transform of the Doppler power spectrum provides the spaced-time correlation function (STCF), which specifies the extent to which there is correlation between the channel's response to two sinusoids sent at different times. It provides the channel's coherence time T_c, a measure of the expected time duration over which the channel's response is essentially invariant [Sklar 2001]. The STCF presents the same data as the Doppler power spectrum, except that is described from the time domain. If $T_s > T_c$, fast fading degradation occurs. If $T_c > T_s$, the channel exhibits slow fading. Figure 2-17 provides the MATLAB code to produce the STCF from the Doppler power spectrum of a channel.

```
%% Computes spaced-time correlation function.
stcf = abs(fftshift(fft(dps)));

% Normalize output.
stcf = stcf / max(stcf);

% Compute range of times for x-axis.
numPoints = length(stcf);
lowerBound = floor(numPoints / 2);
upperBound = lowerBound;
if (mod(lambdaSamples, 2) == 0)
    upperBound = upperBound - 1;
end
t = -lowerBound:upperBound;
time = (totalRecordedSeconds / 2) * t / lowerBound;
```

Figure 2-17: MATLAB code to produce the spaced-time correlation function of a channel.

2.3 Fading Distributions

In shallow water channels, a transmitted signal will arrive at the receiver at slightly different times via multiple paths. If a single pulse is transmitted over a multipath fading channel, it will appear as a pulse train at the receiver, with each pulse in the train corresponding to a distinct multipath component [Goldsmith 2005] which may or may not include the line-of-sight component. In wireless channels, the amplitudes, phase shifts, and number of multipath components vary if pulses are transmitted from a moving transmitter. Thus, they combine at the receiver either constructively or destructively to form a resultant signal that can exhibit significant fluctuations in amplitude and phase. Furthermore, (wind-induced) surface waves, current, and changes in the salinity and temperature profile contribute to the time-varying nature of a shallow water channel, which can also lead to amplitude and phase fluctuations of the received signal, even with stationary apparatus.

There are three distributions commonly used to statistically model the fading channel, namely Rayleigh [Strutt 1880], Rician [Rice 1944, 1945], and Nakagami-*m* [Nakagami 1960]. Rayleigh fading occurs when there are many objects in the environment that scatter the signal before it arrives at the receiver. When the phase of the *n*th multipath component $\Phi_n(t)$ is uniformly distributed, the in-phase and quadrature components $r_I(t)$ and $r_Q(t)$ are both zero mean Gaussian random variables. If the variance is assumed to be σ^2 for both r_I and r_Q, the signal envelope

$$z(t) = |r(t)| = \sqrt{r_I^2(t) + r_Q^2(t)} \tag{2.17}$$

is Rayleigh distributed, as given by the pdf

$$p_Z(z) = \frac{z}{\sigma^2} e^{\left(\frac{-z^2}{2\sigma^2}\right)}, \qquad z \geq 0, \tag{2.18}$$

where $2\sigma^2$ is the average received power of the signal [Goldsmith 2005].

If the channel has a fixed line-of-sight (LOS) component, $r_I(t)$ and $r_Q(t)$ are not zero-mean variables. In this case, the received signal is comprised of the superposition of a complex Gaussian component and a LOS component. The signal envelope is then shown to have a Rician distribution, given by

$$p_Z(z) = I_0\left(\frac{zs}{\sigma^2}\right)\frac{z}{\sigma^2}e^{-\left(\frac{z^2+s^2}{2\sigma^2}\right)}, \qquad z \geq 0, \tag{2.19}$$

where I_0 is the zero-order modified Bessel function of the first kind, $2\sigma^2$ is the average power in the non-LOS multipath component, and s^2 is the power in the LOS component [Goldsmith 2005].

The Rician *K*-factor is defined as the ratio of signal power in dominant component over the (local-mean) scattered power [Linnartz 2009a], as in

$$K = \frac{s^2}{2\sigma^2}. \tag{2.20}$$

When $K = 0$ Rayleigh fading is present, and when $K = \infty$ the channel exhibits no fading. Thus, a small *K* implies severe fading, and a large *K* implies relatively mild fading [Goldsmith 2005].

Since some experimental data cannot be described by the Rayleigh and Rician distributions, the more general Nakagami-*m* distribution was developed. The Nakagami-*m* distribution is given by

$$p_Z(z) = \frac{2m^m z^{2m-1}}{\Gamma(m)\omega^m}e^{\left(\frac{-mz^2}{\omega}\right)}, \qquad m \geq 0.5, \tag{2.21}$$

where ω is the average received power and Γ(·) is the Gamma function. The Nakagami-*m* distribution can model a range of fading channels from one-sided Gaussian fading ($m = 1/2$, worst-case fading) to nonfading ($m = \infty$). $m = 1$ is a special case, equivalent to Rayleigh fading [Nakagami 1960]. Some papers state that the Rician distribution can be closely approximated by the Nakagami-*m* distribution when $m > 1$; however, other references including [Linnartz 2009] and [Yacoub 2005] argue against that claim.

In practice, the fading characteristics of the channel can be determined in two ways. For narrowband[6] fading, a single sinusoid can be transmitted through the channel. The received signal should be run through a bandpass filter before computing its envelope. The amplitude values of the envelope can then be fit to various distributions to determine the best match. For wideband fading, the fluctuation in the amplitudes of a multipath component at time delay τ in the complex-valued impulse response can be fit to the various distributions. It is possible for each multipath component to exhibit a different fading distribution.

2.4 Underwater Channel – Office Test Environment

Channel characterization is not a simple procedure. Typically it involves taking two boats, each equipped with power supplies, computers, an emitter and/or hydrophones, and a crew of scientists, out to a body of water to gather data. While this procedure is necessary for thorough characterization of a real underwater channel, it is cost-prohibitive for constructing and tweaking the channel-sounding experiment. Therefore, a simple underwater channel can be created out of a large tub filled with tap water and used for experiment design and prototyping. While it may not exhibit the time-varying complexities found in real shallow water channels, it does possess a significant amount of multipath propagation due to the close proximity of the hard walls of the tub.

In the Lieb building at Stevens Institute of Technology, a testbed has been created with the following components: Rubbermaid® storage tub, OCEANEARS DRS-4 emitter [OCEANEARS 2010], OCEANEARS DRS-2 hydrophone, custom-built fixed-gain amplifier, Lenovo T60p laptop, and an M-Audio Transit USB recorder. Only one laptop was used in the experiment, which simultaneously played and recorded the sounding signals. To ensure there was no internal leakage of the playback signal in the microphone input, the T60p's onboard sound card

[6] Narrowband refers to a situation where the bandwidth of the transmitted signal is less than the channel's coherence bandwidth.

was used solely for transmission while the external M-Audio device was used for recording. The sounding signals were played in Winamp 5.52 and recorded with Adobe Audition 3.0. The OCEANEARS devices were affixed to the bottom of the table with duct tape and oriented so that the large, flat surface of each transducer was facing the device on the opposite side of the tub. The two transducers were separated by about 14 inches. Figure 2-18 depicts the layout of the office test environment.

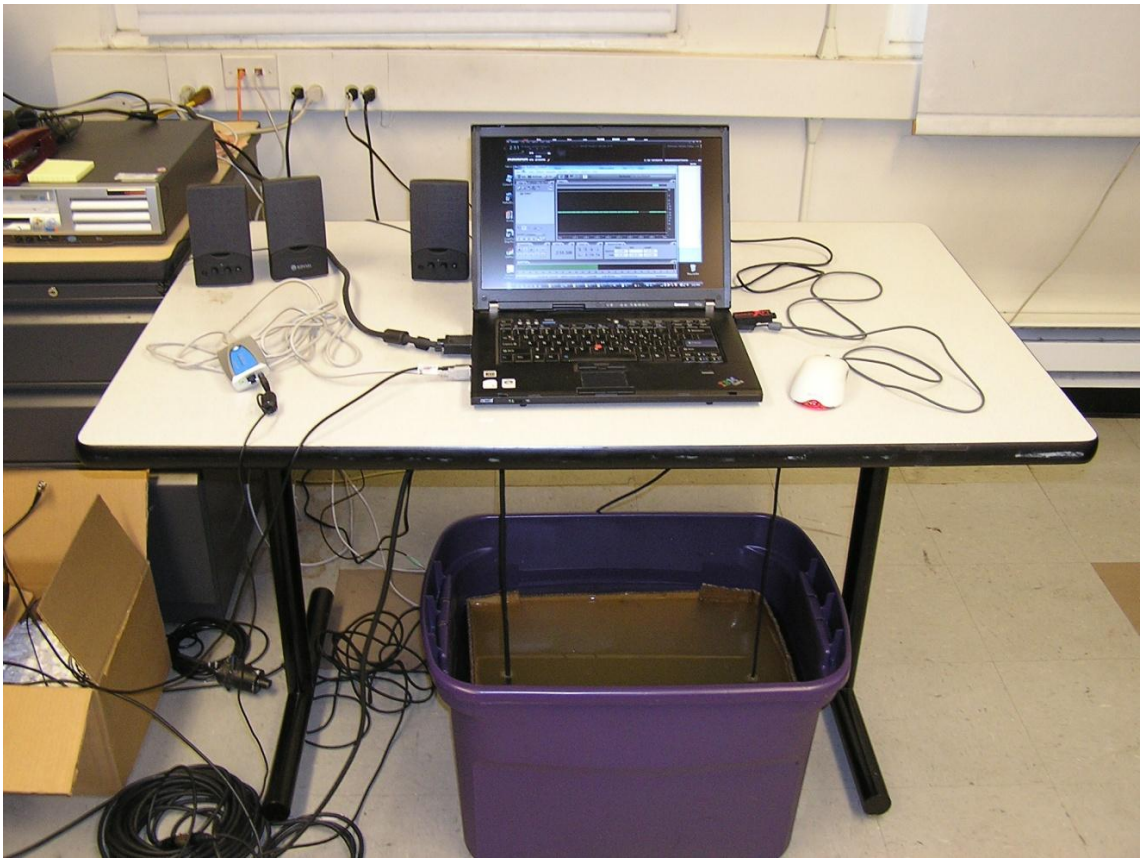

Figure 2-18: Underwater channel testbed inside office.

2.4.1 Sounding Signal

The sampling rate was set to 48 kHz for all experiments conducted in the office. A 50-ms LFM chirp from 0 – 24 kHz was repeatedly used to sound the channel. Figure 2-19 shows the autocorrelation function of the sounding signal in dB scale.

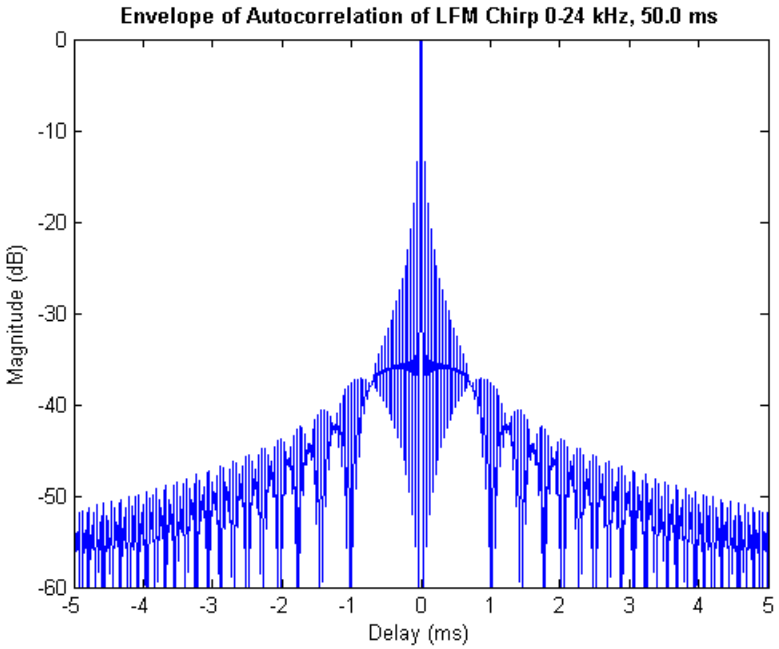

Figure 2-19: Envelope of autocorrelation of LFM Chirp 0-24 kHz, 50.0 ms.

2.4.2 Initial Tub Configuration

Figure 2-20: Initial configuration of underwater channel in office setup.

The tub, which is about 20.5 inches wide and 15.5 inches deep, was initially filled with approximately 9.5 inches of water, as seen in Figure 2-20. The device on the left is the OCEANEARS DRS-4 emitter; the one on the right is the OCEANEARS DRS-2 hydrophone.

2.4.2.1 Impulse Response

The impulse response of the tub over a span of 5 seconds appears in Figure 2-21. Each second on the graph contains 20 impulse response estimates. Since the channel is time-invariant, all impulse response estimates are nearly identical. Thus, when the collection of estimates is viewed as a scaled image, it appears as a series of vertical columns. Columns in red represent strong correlation at a specific time delay, while columns in blue represent low correlation. Values between the extremes are displayed in various colors of the spectrum present in the color bar to the right of the graph.

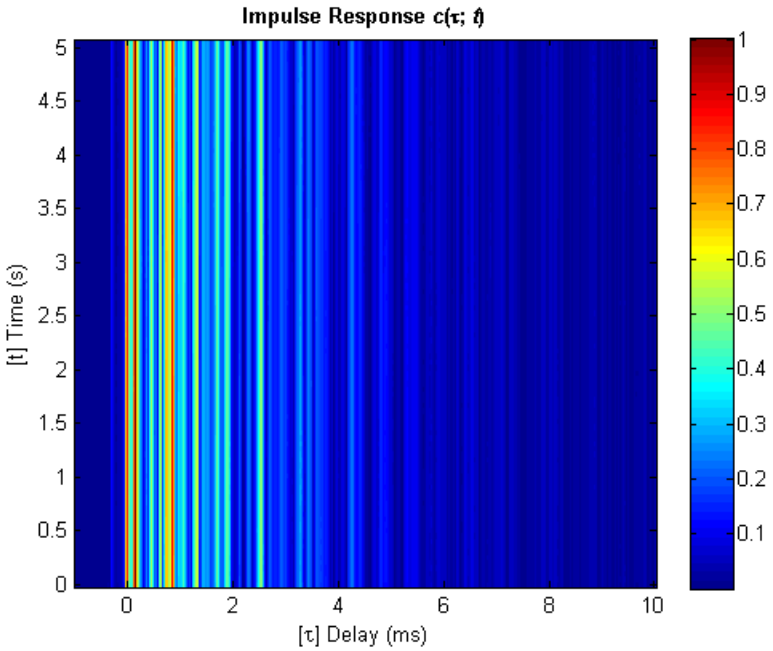

Figure 2-21: Successive impulse response estimates of office test tub.

2.4.2.2 Scattering Function

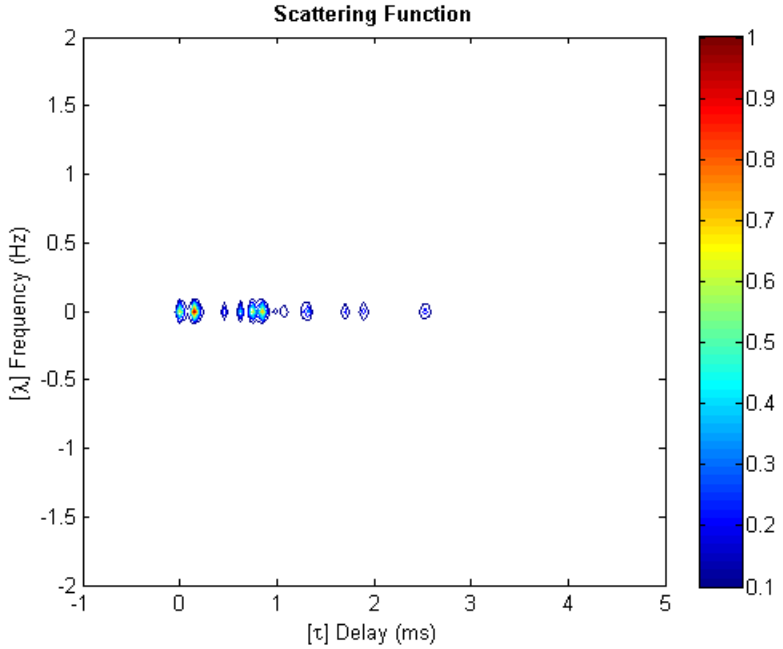

Figure 2-22: Scattering function of office test tub.

The scattering function depicts significant multipath arrivals occurring for approximately 2.5 ms. There is no Doppler shift, since the devices were fixed, and no measureable spreading in the frequency domain.

2.4.2.3 Multipath Intensity Profile

The delay spread of the channel appeared rather unconventional with many strong multi-path components. While it is possible for an arrival of lesser intensity to precede the one with the strongest amplitude, it is not common to find repeating sequences of weaker and then stronger arrivals, like those shown in the multipath intensity profile in Figure 2-23. This phenomenon is caused by the acoustic waves being reflected off the walls of the tub. Moreover, since the tub is constructed out of hard plastic, very little acoustic energy is absorbed. Thus, the waves were re-flected several times before losing intensity.

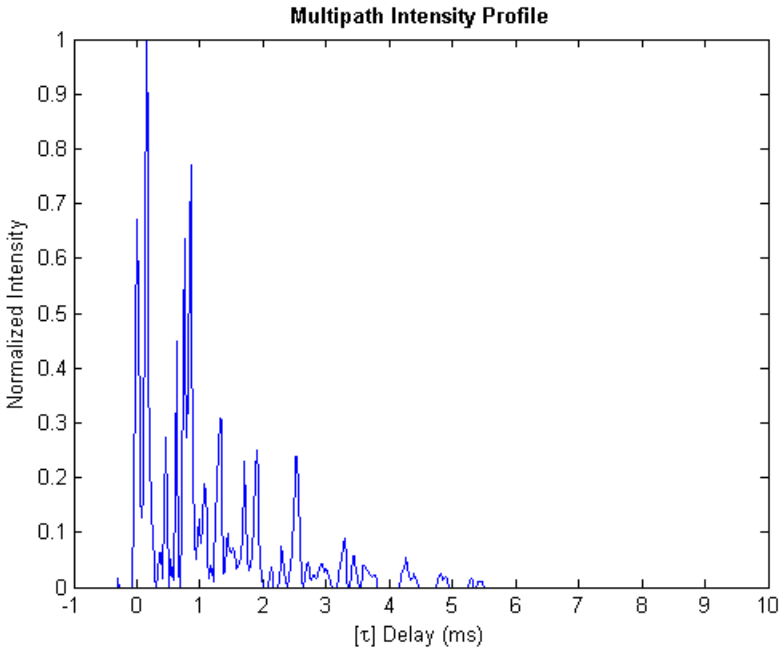

Figure 2-23: Multipath intensity profile of office test tub.

Table 2-1: Delay Spread (ms) of Multipath Intensity Profile Computed with -20 dB Threshold

Mean Excess Delay	RMS Delay Spread	Maximum Excess Delay (-10 dB)
1.1513	1.0945	2.6042

Table 2-2: Doppler Shift and Spread (Hz) of Strong Multipath Arrivals

	Time Delay (ms)	Intensity	Shift	Spread
Arrival 1	0.000	0.6719	0.0000	0.0000
Arrival 2	0.146	1.0000	0.0000	0.0000
Arrival 3	0.458	0.2734	0.0000	0.0000
Arrival 4	0.625	0.4477	0.0000	0.0000
Arrival 5	0.750	0.6346	0.0000	0.0000
Arrival 6	0.854	0.7692	0.0000	0.0000
Arrival 7	1.313	0.3098	0.0000	0.0000
Arrival 8	1.708	0.2284	0.0000	0.0000
Arrival 9	1.896	0.2502	0.0000	0.0000
Arrival 10	2.542	0.2393	0.0000	0.0000

2.4.2.4 Spaced-Frequency Correlation Function

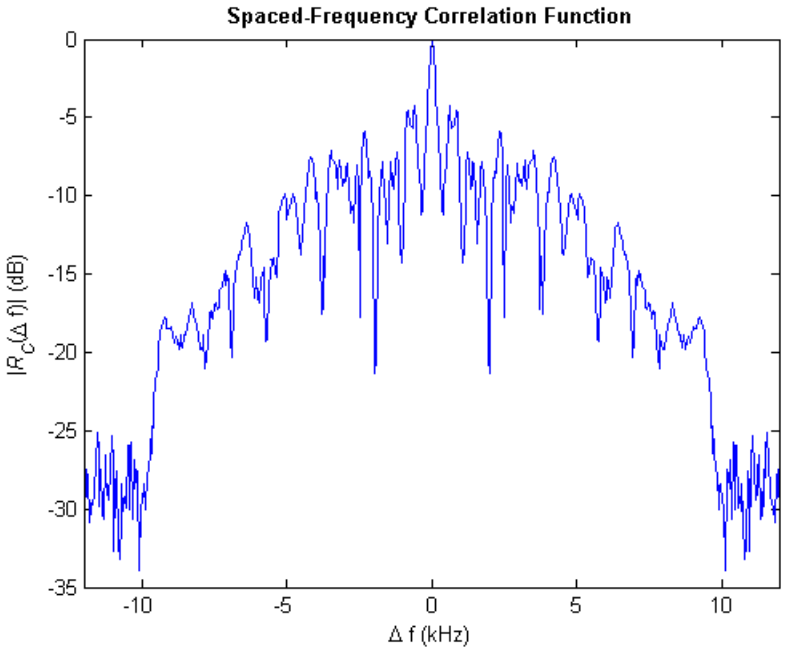

Figure 2-24: Spaced-frequency correlation function of office test tub.

Table 2-3: Coherence Bandwidth (Hz)

-3 dB	-6 dB	-10 dB
181	363	544

2.4.2.5 Doppler Power Spectrum

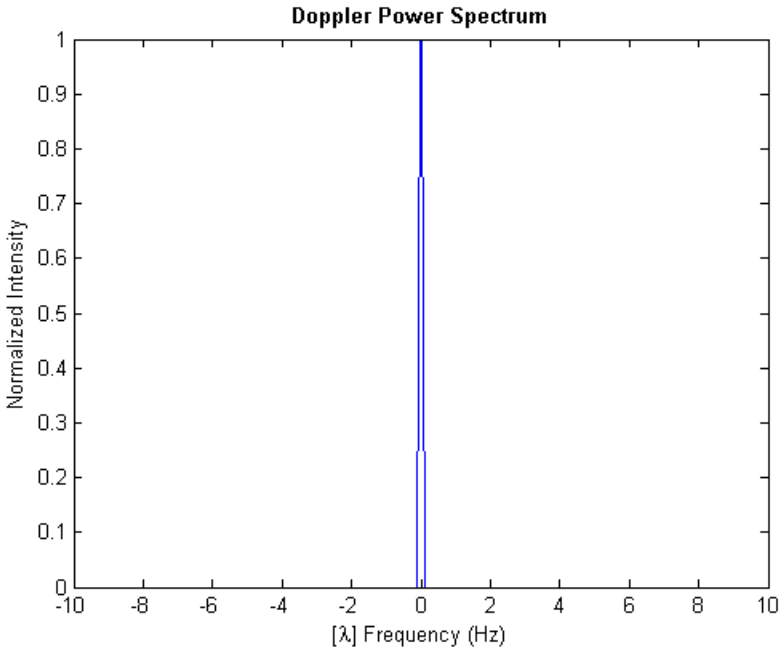

Figure 2-25: Doppler power spectrum of office test tub.

Table 2-4: Overall Doppler Shift and Spread (Hz)

Shift	Spread
0.000	0.000

2.4.2.6 Spaced-Time Correlation Function

Table 2-5: Coherence Time (ms)

0.5 (-3 dB)	0.25 (-6 dB)	0.1 (-10 dB)
∞	∞	∞

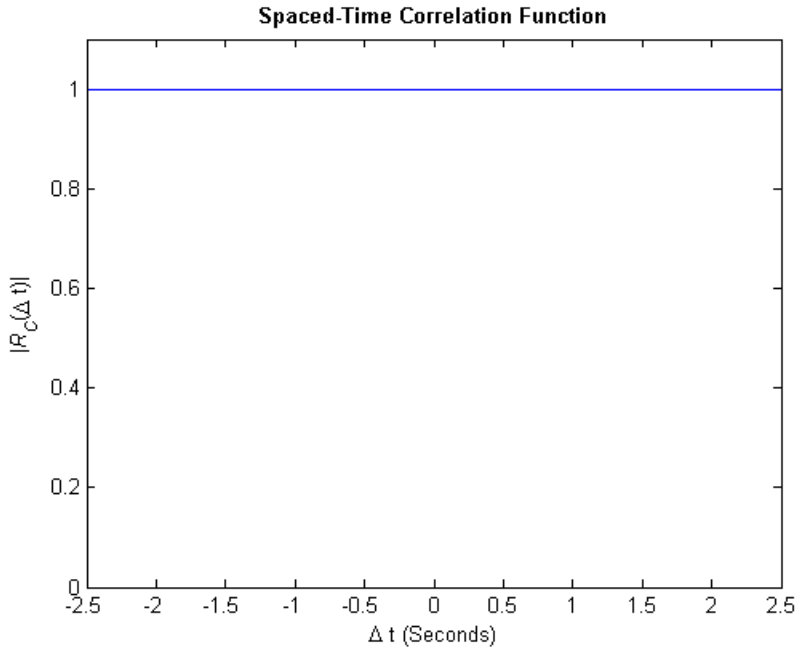

Figure 2-26: Spaced-time correlation function of office test tub.

2.4.2.7 Analysis and Implications for Communication

The analysis of the channel sounding data, presented in Sections 2.4.2.1 through 2.4.2.6, describes a channel that is time-invariant yet exhibits significant multipath propagation. A channel that is truly as time-invariant as a tub does not occur in nature, so this type of environment should be used only to test underwater communications systems against a channel with significant delay spread. Consequently, a zero-forcing equalizer can be employed. This type of equalizer works by inverting the estimated channel response and applying the result to the incoming signal stream. More complex adaptive filters can be used, but this particular channel won't provide any interesting test cases.

The amount of delay spread is directly proportional to the duration of a symbol T_s used in a communication system that does not employ an equalizer. Some authors define T_s in terms of the maximum excess delay T_m [Sklar 2001], while others use the rms delay spread [Rappaport

2002]. Assuming T_m is used, if $T_m > T_s$, the channel exhibits frequency-selective fading, which results in channel-induced ISI. In this case, the communication system will need to perform equalization to mitigate the distortion. If $T_s > T_m$, the channel exhibits flat fading, which does not result in ISI. T_m = 2.6042 ms, reported in Table 2-1, implies the tub system can transmit up to about 384 symbols per second and still avoid ISI. If the rms delay spread σ_{T_m} is used in the calculation, when $T_s \gg \sigma_{T_m}$ the channel induces only negligible ISI. Assuming T_s is within an order of magnitude of σ_{T_m}, $T_s \gg \sigma_{T_m}$ implies $T_s > 10\sigma_{T_m}$, and there will be some ISI which, depending on the system, may or may not significantly degrade performance [Goldsmith 2005]. Using the rms delay spread σ_{T_m},

$$\frac{\sigma_{T_m}}{T_s} \leq 0.1$$

$$T_s \geq \frac{\sigma_{T_m}}{0.1}$$

$$T_s \geq \frac{1.0945 \text{ ms}}{0.1}$$

$$T_s \geq 10.945 \text{ ms}$$

$$R_s = \frac{1}{T_s} = 0.0914 \times 10^3 \text{sps} = 91 \text{ sps},$$

which is about one-fourth the value computed using the maximum excess delay. Regardless of which method is used, it is clear that the delay spread of the channel significantly decreases the effective data rate of this channel to a maximum of at most a few hundred symbols per second.

Time-spreading can also be viewed from the frequency domain. The SFCF yields the coherence bandwidth *f* of the channel. If $W > f$, where W is the bandwidth required for modulation, the channel imposes frequency-selective degradation. If $f > W$, the channel exhibits flat fading. No universal relationship exists between the coherence bandwidth and delay spread [Sklar 2001; Rappaport 2002], since the exact relationship is a function of specific channel impulse res-

ponses and applied signals [Rappaport 2002]. It is also important to note that while the rms delay spread calculation may predict some symbol rate, the actual bandwidth required by a certain type of modulation at that symbol rate may exceed the coherence bandwidth of the channel. Therefore, the achievable symbol rate for modulation within the given bandwidth may be even less than predicted.

2.4.2.8 Details of Calculation

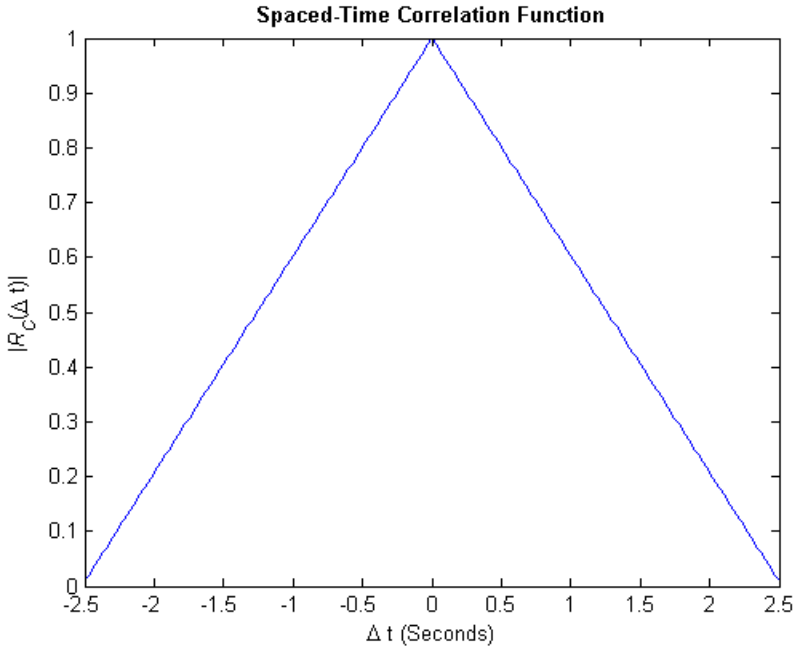

Figure 2-27: First attempt at spaced-time correlation function of office test tub.

The first attempt at computing the STCF of the office test tub produced the result shown in Figure 2-27. It reveals a -3 dB coherence time of approximately 2.5 seconds, which is far shorter than the theoretical value of infinity expected in a time-invariant system. From viewing the successive impulse response estimates in Figure 2-21, it appears that the amplitudes of each component in the impulse response remain constant over the entire duration of the test. Therefore, one should expect the Doppler power spectrum to be a delta function, with all the power occurring at the 0 Hz lambda frequency. The Fourier transform of such a delta function would

yield a flat STCF, with correlation values of 1 across the entire Δt x-axis. However, the original results contradict these expectations.

In order to determine where the model breaks down, it is necessary to view the output of each step in a controlled experiment. The magnitude values of the largest component of the impulse response can be plotted to verify that they are approximately equal. Since the impulse response is complex-valued, the magnitude of each sample is given by

$$magnitude(x) = \sqrt{x_r^2 + x_i^2}, \tag{2.22}$$

where x_r denotes the real component and x_i the imaginary component of the complex number x. The mean value of the normalized magnitude over the 5-second test is 0.998 with a standard deviation of 0.001. Thus, the fluctuations are indeed negligible, confirming a time-invariant channel.

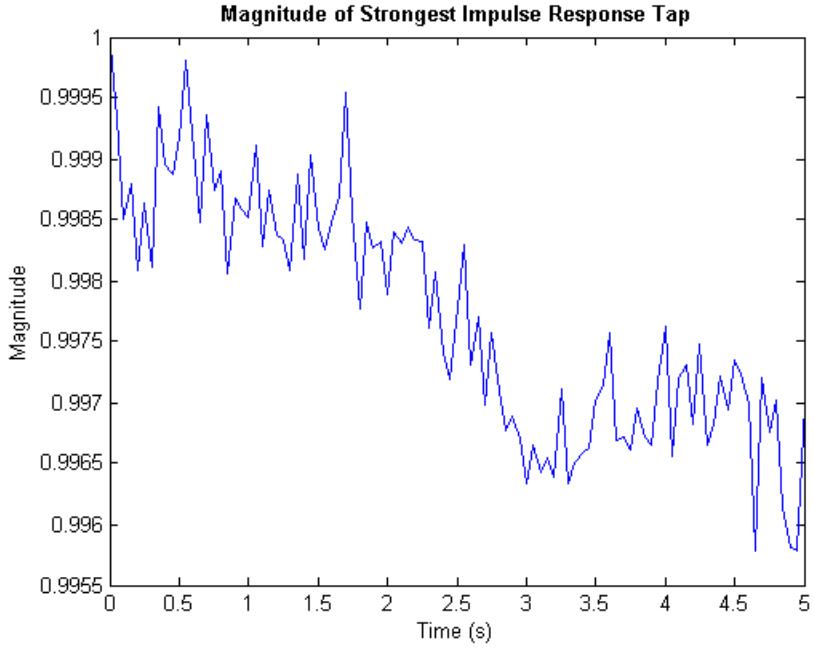

Figure 2-28: Magnitude of the strongest impulse response tap over time.

The next step in the verification process is to analyze the output of the scattering function at the iteration of the loop pertaining to the strongest impulse response tap. The algorithm in-

volves taking for the Fourier transform of the autocorrelation of the strongest impulse response tap (across all 5 seconds of measurements). The result produces the Doppler power spectrum for a given multipath component. Figure 2-29 shows the Doppler power spectrum of the strongest multipath arrival in the tub. Note the small side lobes at the bottom of the spike. They are an artifact of computing the FFT with a rectangular window, which has poor sideband attenuation of only -13 dB [NI 2010c]. Figure 2-29 depicts the Doppler power spectrum of the strongest multipath arrival in linear scale; Figure 2-30 depicts it in dB scale.

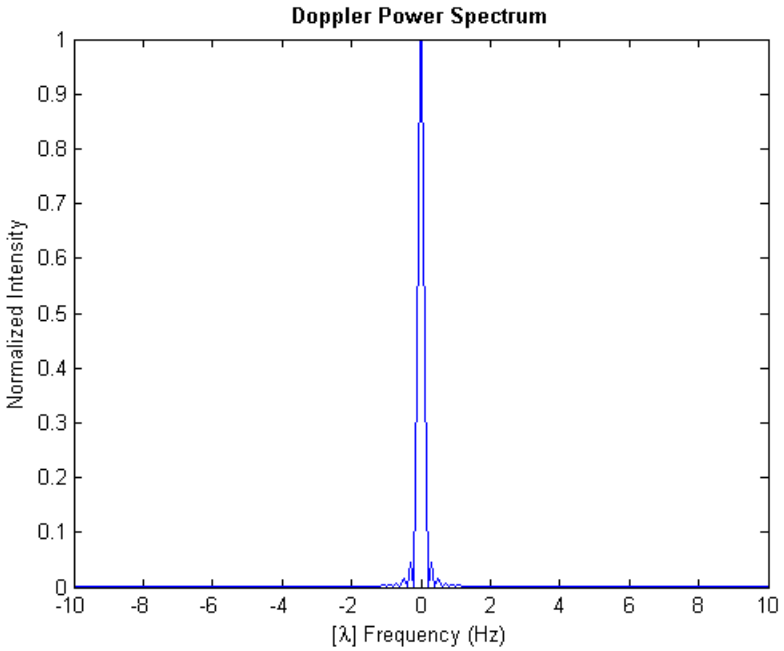

Figure 2-29: Doppler power spectrum of strongest multipath arrival.

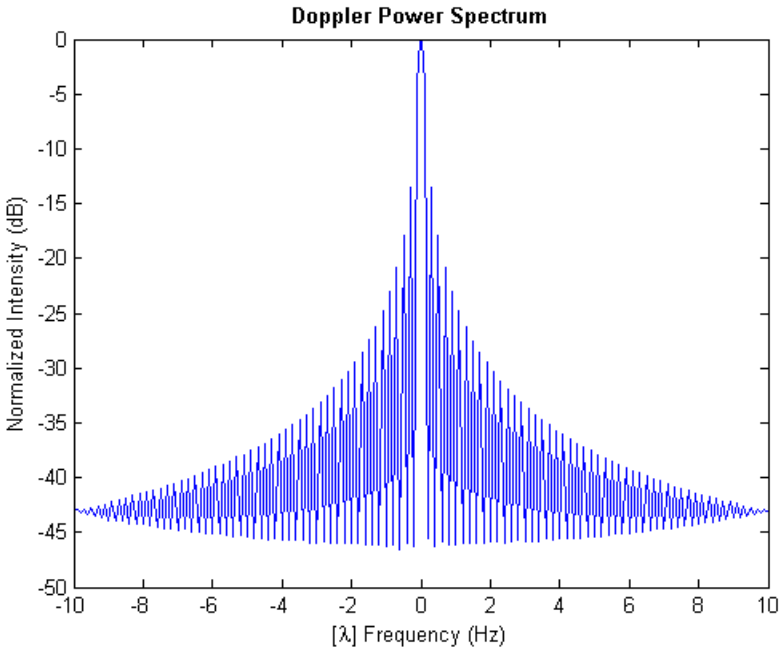

Figure 2-30: Doppler power spectrum of strongest multipath arrival in dB scale.

It is not useful to apply a smoothing window to the signal before taking its Fourier transform. While windowing affords greater side lobe attenuation, it increases the width of the main lobe, thus decreasing the frequency resolution. The increase in the width of the main lobe with attenuated side lobes has a similar effect on the resulting STCF as a narrow main lobe with significant side lobes – they yield a coherence time that is shorter than expected. In the case of this time-invariant channel, it proves useful to simply eliminate the side lobes by zeroing out any part of the scattering function that is less than 13 dB of the intensity of the strongest component.

The main point in describing this observation and all the intermediate calculations is to show that the WSSUS model used to describe the properties of a channel is just that – a model. While artifacts of the computational methods are often not described in literature, they can have significant effects on the results. The derived characterization functions and predictions about symbol rates must be viewed as rough approximations.

2.4.3 Modified Tub Configuration

Figure 2-31: Final configuration of underwater channel in office setup. The walls of the tub were lined with cardboard and the bottom was covered with sand to reduce multipath propagation.

Because of the extreme amount of multipath propagation present in the initial configuration of the underwater channel in the office testbed, a second attempt was required to produce an artificial channel that more realistically simulates natural bodies of water. In order to reduce the number and intensity of reflections, the walls of the tub were lined with cardboard and the bottom was covered with sand. Figure 2-31 depicts the altered configuration of the tub, which was used for all subsequent measurements in this chapter and the communication experiments discussed in Chapter 3.

2.4.3.1 Impulse Response

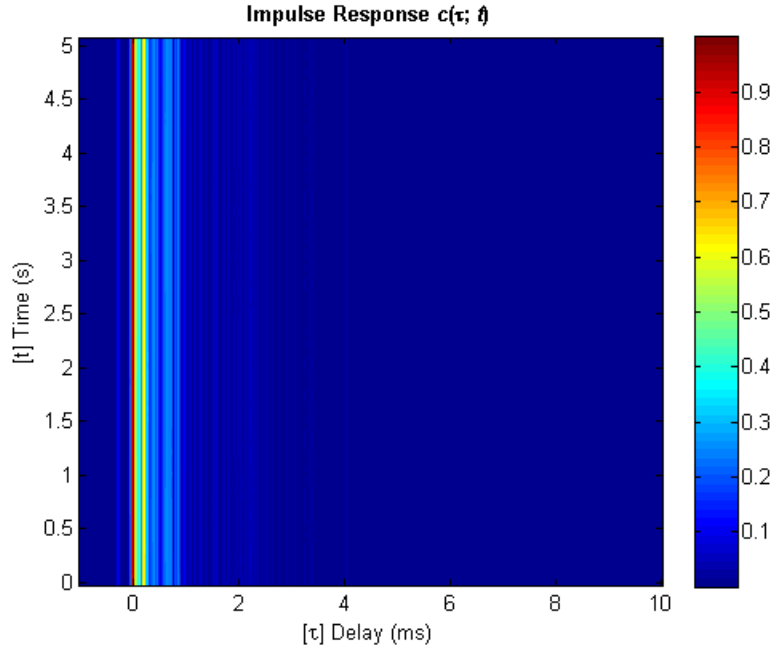

Figure 2-32: Successive impulse response estimates of modified office test tub.

2.4.3.2 Scattering Function

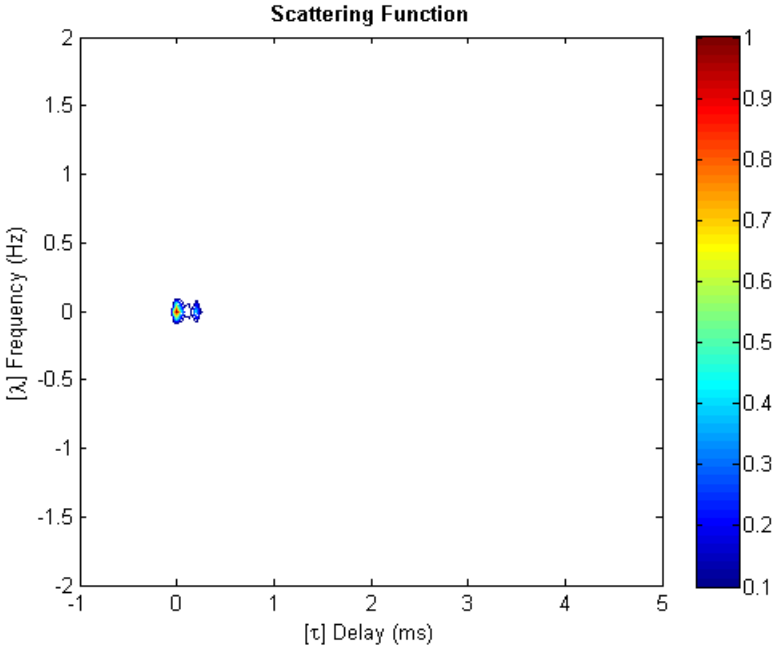

Figure 2-33: Scattering function of modified office test tub.

2.4.3.3 Multipath Intensity Profile

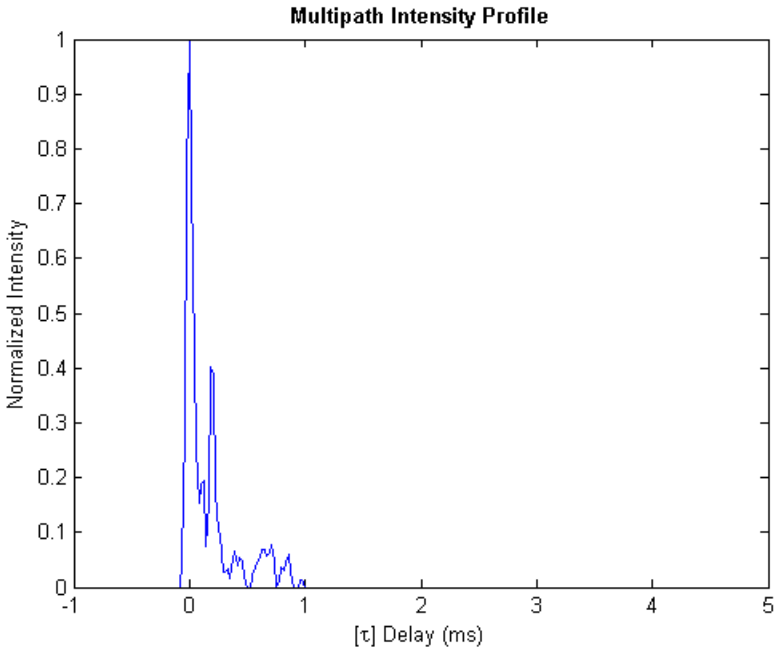

Figure 2-34: Multipath intensity profile of modified office test tub.

Table 2-6: Delay Spread (ms) of Multipath Intensity Profile Computed with -20 dB Threshold

Mean Excess Delay	RMS Delay Spread	Maximum Excess Delay (-10 dB)
0.1609	0.2327	0.2917

Table 2-7: Doppler Shift and Spread (Hz) of Strong Multipath Arrivals

	Time Delay (ms)	Intensity	Shift	Spread
Arrival 1	0.000	1.0000	0.0000	0.0000
Arrival 2	0.188	0.4026	0.0000	0.0000

2.4.3.4 Spaced-Frequency Correlation Function

Table 2-8: Coherence Bandwidth (Hz)

-3 dB	-6 dB	-10 dB
1633	6170	11251

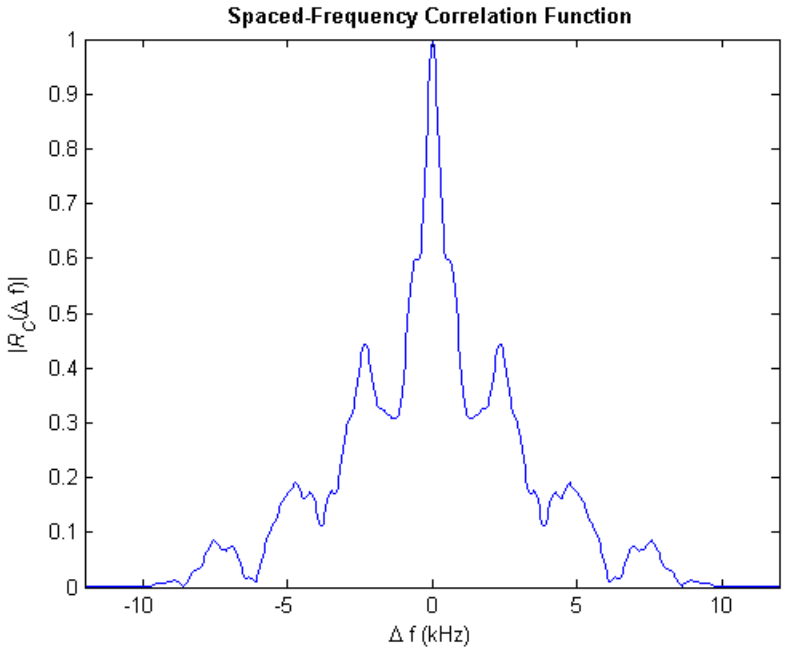

Figure 2-35: Spaced-frequency correlation function of modified office test tub.

2.4.3.5 Doppler Power Spectrum

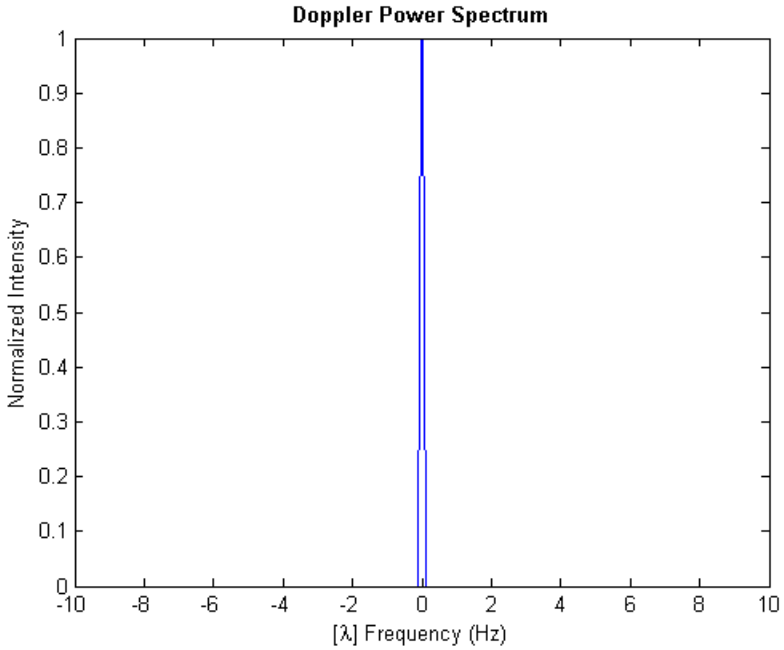

Figure 2-36: Doppler power spectrum of modified office test tub.

Table 2-9: Overall Doppler Shift and Spread (Hz)

Shift	Spread
0.000	0.000

2.4.3.6 Spaced-Time Correlation Function

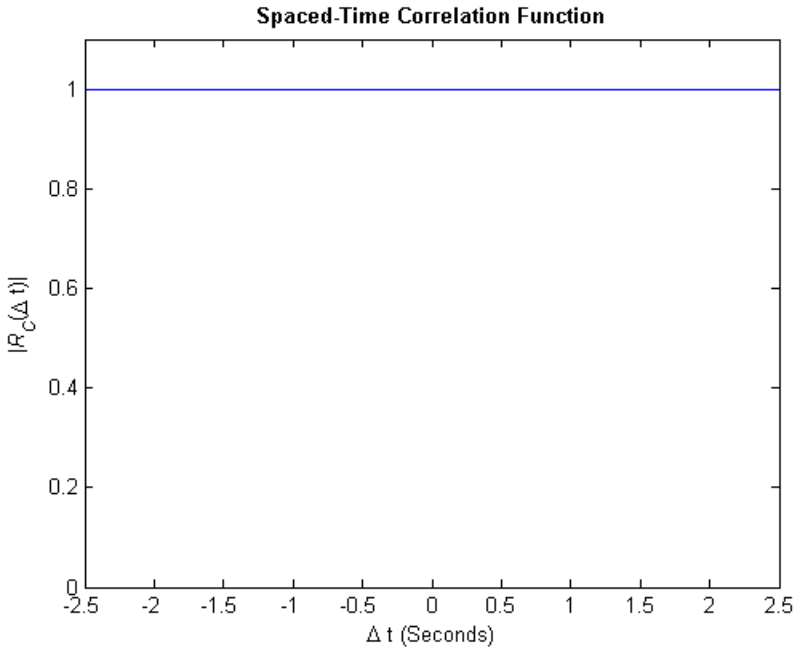

Figure 2-37: Spaced-time correlation function of modified office test tub.

2.4.3.7 Analysis and Implications for Communication

The modified office test tub possesses a much more realistic multipath intensity profile. There is one main arrival, followed by several reflections, only one of which is strong. There are no longer groups of weaker arrivals followed by a strong component. As a result of the reduced delay spread, the channel permits the transmission of signals with higher data rates. Using the maximum excess delay as the lower bound for the duration of a symbol, the system can transmit up to about 3428 symbols per second and still avoid ISI. The rms delay spread approach produces a much smaller rate of only approximately 430 symbols per second.

2.5 Underwater Channel – Hudson River Estuary

2.5.1 Experiment

Figure 2-38: Test site for channel sounding experiment.

The Maritime Security Laboratory (MSL) at Stevens Institute of Technology conducted a field test on August 21, 2008, in the Hudson River estuary adjacent to its campus, as shown in Figure 2-38. Two boats were used, one for emitting signals – the Phoenix – and the other for recording them – the Savitsky. The Phoenix was also equipped with a hydrophone to record the emitted signals at a distance of 1 meter from the transducer for later reference. The computer system on the Phoenix used the NI USB-6221 [NI 2010b] data acquisition (DAQ) board; the system on the Savitsky utilized the NI PCI-6123 [NI 2010a]. All reference and recorded signals were created at 200 ksamples/second. A custom ceramic transducer emitted the signals, while ITC-6050C hydrophones [ITC 2010] were used for reception.

The channel was about 3 meters deep, and sounding experiments were performed at distances of 505 and 200 meters. The emitter was placed 1 meter below the surface; hydrophones

were placed 60 cm from the bottom. The boats were anchored and motors turned off while data were gathered. Each test consisted of taking CTD[7] measurements prior to channel sounding, recording 30 seconds of ambient noise, playing a comb signal containing 5 sinusoidal components – 35, 45, 60, 75, and 85 kHz – for 1 minute, and repeatedly emitting a 50-ms LFM chirp signal spanning 20-100 kHz for 30 seconds.

2.5.2 Sounding Signal

Analysis of prior measurements in the Hudson River estuary revealed significant noise below 20 kHz and a very short coherence time, which facilitated the choice of sounding signal for this test. As stated in Section 2.2.1, the LFM chirp signal is a good choice for a sounding signal, since it possesses good autocorrelation properties as to closely approximate the Dirac delta function. It is known that the best autocorrelation function for a LFM chirp signal is obtained when chirping from 0 Hz up to the Nyquist frequency. However, because of the Hudson's noise in lower frequency bands and the emitter's lack of low frequency response, 20 kHz was chosen as the starting frequency for the chirp signal.

2.5.3 Environmental Conditions

The field test in the Hudson River estuary was conducted on the afternoon of August 21, 2008. Environmental conditions were recorded at the Castle Point Buoy, at 40.74348° latitude and -74.02263° latitude, and downloaded upon the completion of the experiment from the website for the Urban Ocean Observatory at the Center for Maritime Systems [NYHOPS 2009]. The 505-meter test was started at 5:14 P.M. At that time, the temperature was 76°F with 55% relative humidity, the wind speed was about 10 knots, and the wind direction was about 159° [WIS 2009]. Medwin's expression for sound velocity in meters/second

[7] A CTD – an acronym for Conductivity, Temperature, and Depth – is a sensor used to determine essential physical properties of sea water.

$$c = 1449.2 + 4.6T - 5.5 \times 10^{-2}T^2 + 2.9 \times 10^{-4}T^3 + (1.34 - 10^{-2}T)(S - 35) + 1.6 \times 10^{-2}D, \quad (2.23)$$

where D is the depth in meters, S is the salinity in parts per thousand (ppt), and T is the temperature in degrees Celsius [Urick 1996], was applied to the CTD measurements taken at the start of the test. Other expressions for sound velocity exist; however, the salinity values obtained in the test fall outside the range of acceptable input values for these expressions. Figure 2-39 shows the temperature, salinity, and derived sound velocity of the water column in the 505-meter test. Since the numeric difference between practical salinity unit (psu) and ppt is small, psu values were used in place of ppt values while generating this graph.

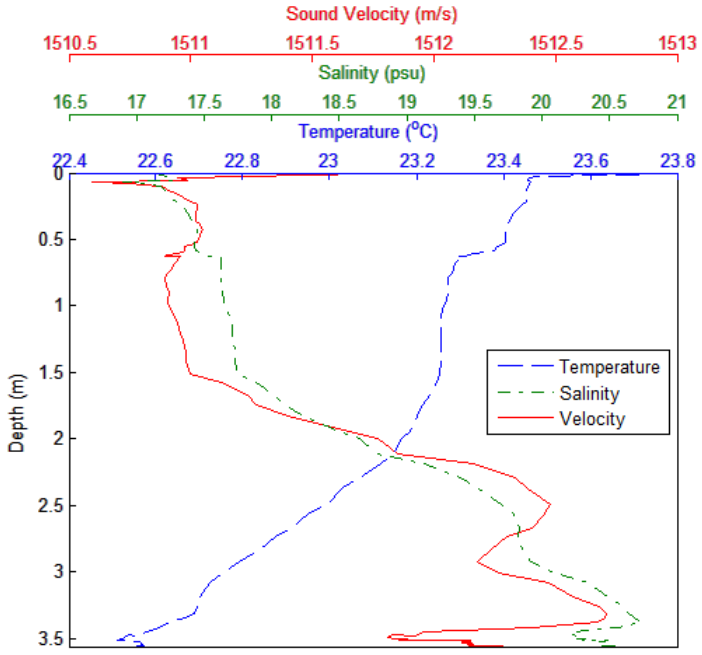

Figure 2-39: Sound velocity profile for 505-meter channel.

The 200-meter test was started at 6:04 P.M. At that time, the temperature was 75°F with 57% relative humidity, the wind speed was about 8 knots, and the wind direction was still about 159°. Following the same procedure as before, the CTD measurements resulted in the sound velocity profile shown in Figure 2-40.

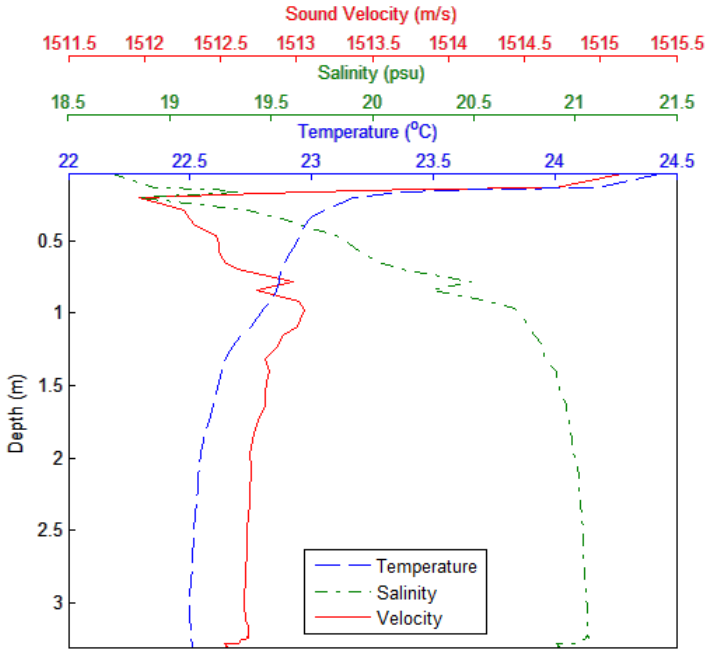

Figure 2-40: Sound velocity profile for 200-meter channel.

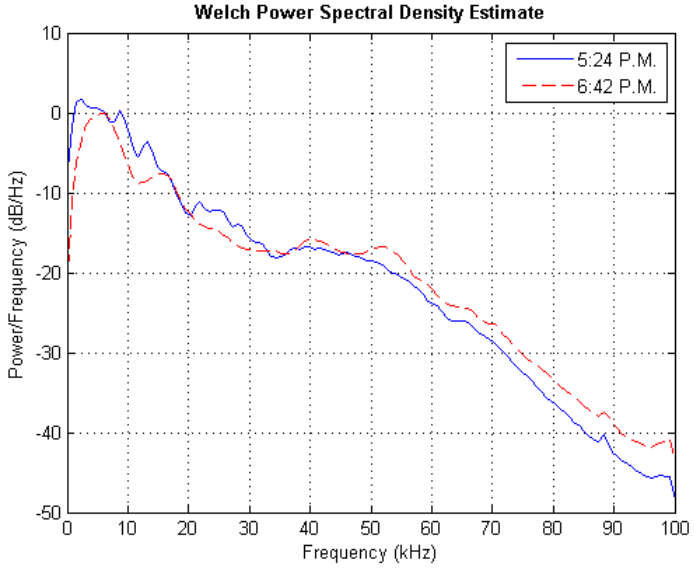

Figure 2-41: PSD of ambient noise in Hudson River estuary.

Ambient noise was recorded for 30 seconds during both tests before any signals were emitted. The power spectral density (PSD) of noise was estimated via a Welch periodogram technique based on a 256-point FFT together with a Hanning window and no overlap. Figure

2-41 shows the PSD of noise in the Hudson River estuary captured at 5:24 and 6:42 P.M.

2.5.4 Time-Variant Impulse Response

The time-varying complex-valued low-pass impulse response $c(\tau; t)$ of the underwater channel was captured via same procedure as outlined in Section 2.2.2 but with two additional steps. The complete method is as follows:

1. The 50-ms chirp signals were recorded 1 meter from the emitter and either 200 or 505 meters away (depending on the test) where the Savitsky was anchored.
2. The received signal and 1-meter reference signal were run through a 10^{th} order high-pass Butterworth filter at 20 kHz to eliminate out-of-band noise.
3. One chirp was extracted from the 1-meter reference signal, accurate to the sample.
4. The imaginary part of the reference chirp signal was obtained via the Hilbert transform.
5. The received signal was cross-correlated with the complex conjugate of the reference chip signal.

There are several experimental issues worth describing in more detail. The standard technique is to apply matched filtering to the received signal with the original waveform sent to the emitter. While this approach results in the best autocorrelation function for a given chirp signal, it unfairly distributes weight to frequencies that were not emitted with equal amplitudes, as in the case when the frequency response of the emitter is not flat. In this situation, the derived impulse response estimates contain higher levels of noise correlation.

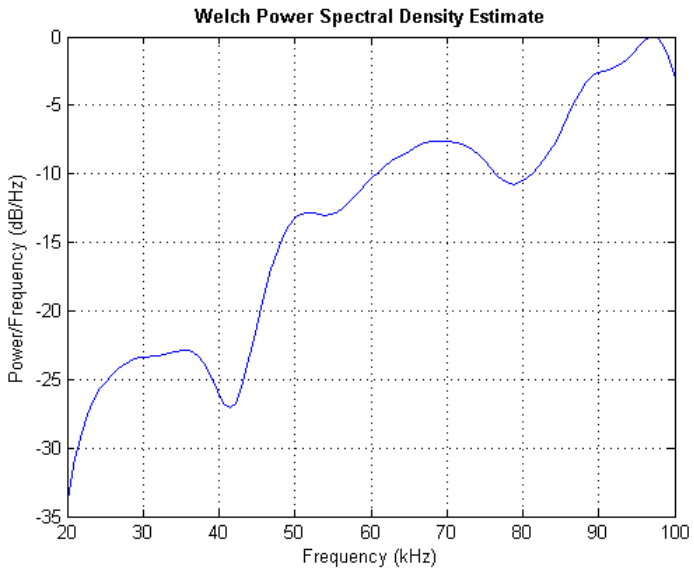

Figure 2-42: PSD of chirp signal at 1m (frequency response of emitter).

The custom transducer used in this experiment does not exhibit a flat response over the 20-100 kHz band. Figure 2-42 shows the PSD of the chirp signal that it emitted. The envelope of the autocorrelation function of the original chirp waveform with a flat response has a narrow main lobe and negligible side lobes, as shown in Figure 2-43. On the other hand, as shown in Figure 2-44, the autocorrelation function of the chirp signal produced by the custom transducer has a wider main lobe and stronger side lobes, averaging about -40 dB across the 10 ms spread. However, since the signal was received with high SNR (19 dB < SNR < 58 dB), the distortion introduced by the side lobes on the impulse response estimates was minimal.

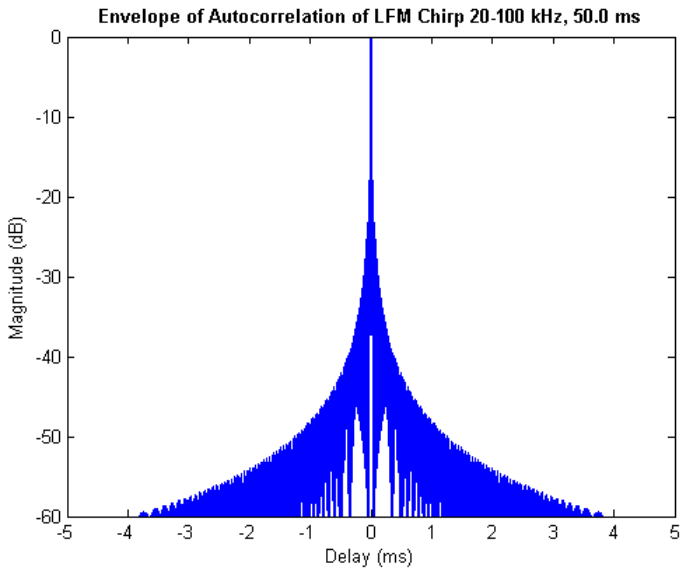

Figure 2-43: Envelope of original chirp waveform autocorrelation function.

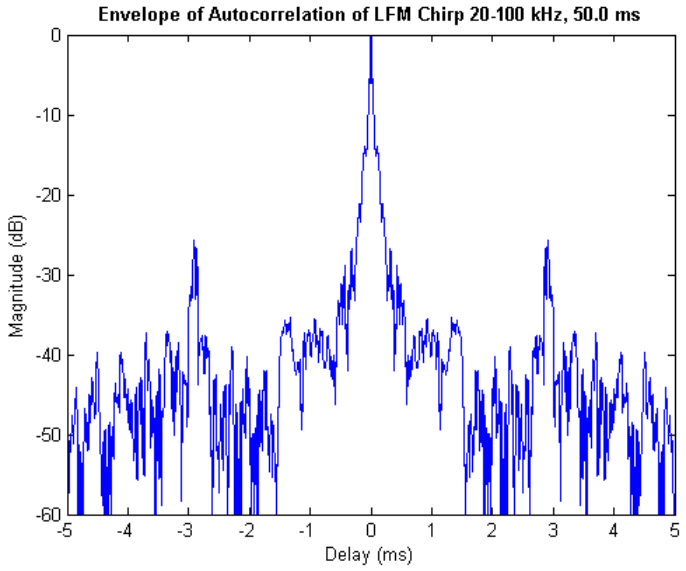

Figure 2-44: Envelope of emitted chirp waveform autocorrelation function.

Figure 2-45 shows 30 seconds of impulse response estimates $c(\tau; t)$ of the Hudson River estuary at the 200-meter distance. Fading is present, but it appears that there are one or possibly two line-of-sight arrivals. Figure 2-46 shows the time evolution of impulse response estimates observed over the 505-meter channel. While there is significant fluctuation in the amount of cor-

relation, the strongest correlation always occurs within a 0.5-ms window. Three multipath components are present, though each has been subjected to periods of deep fading.

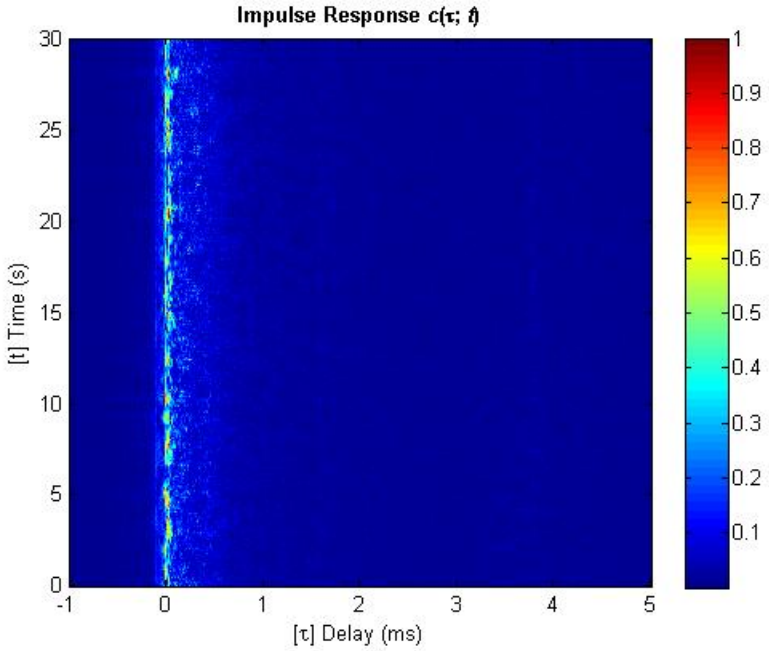

Figure 2-45: Successive time-variant impulse response estimates of Hudson at 200m.

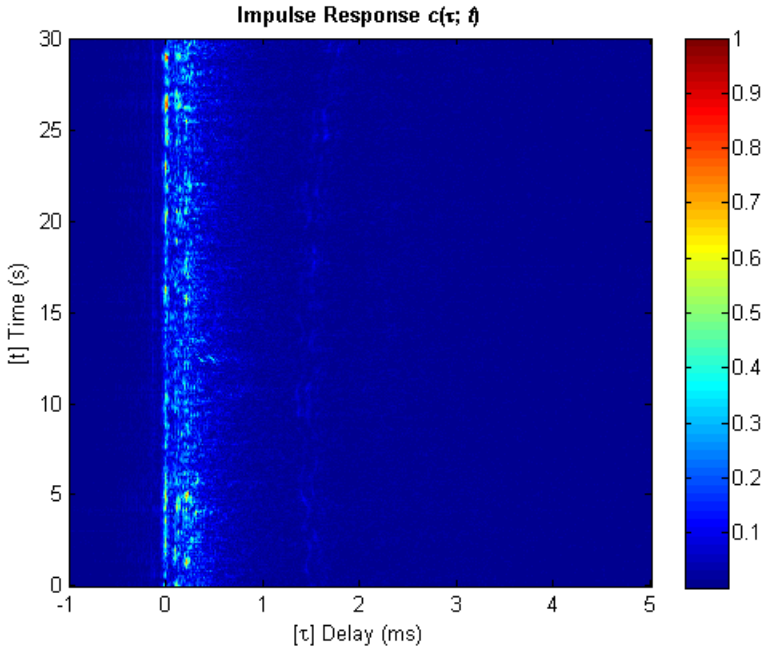

Figure 2-46: Successive time-variant impulse response estimates of Hudson at 505m.

2.5.5 Scattering Function

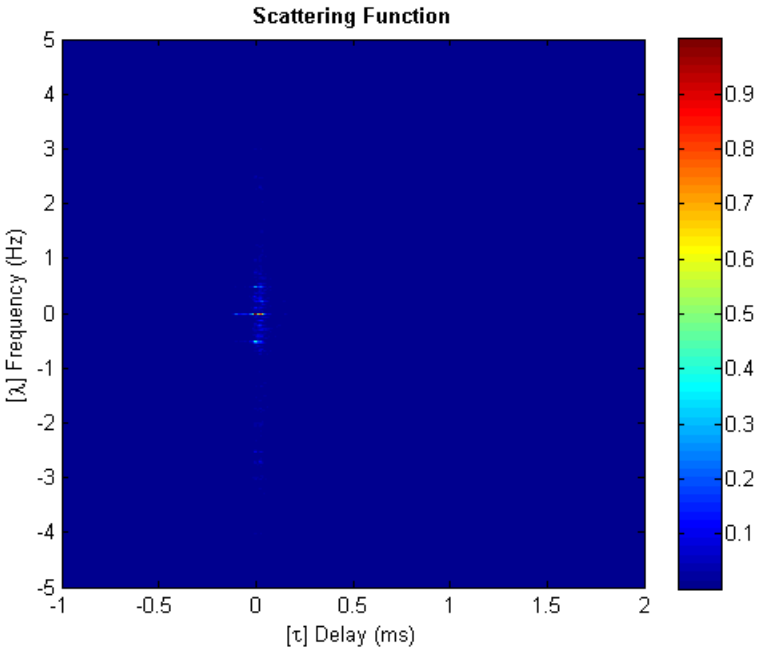

Figure 2-47: Scattering function of Hudson at 200m.

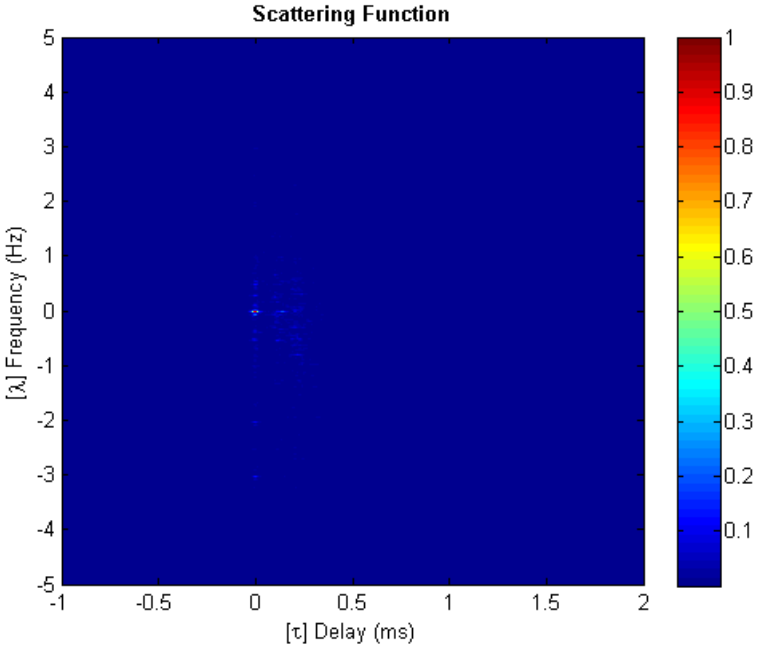

Figure 2-48: Scattering function of Hudson at 505m.

2.5.6 Multipath Intensity Profile

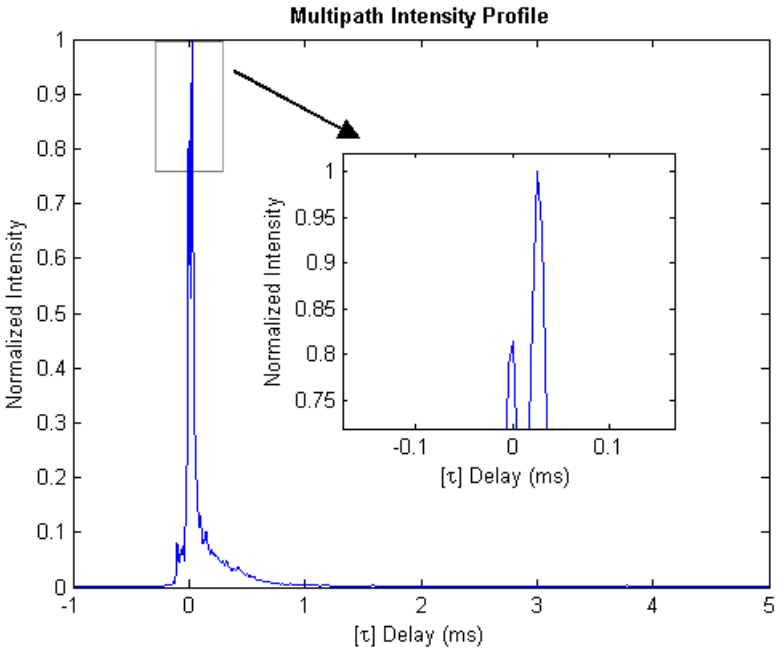

Figure 2-49: Multipath intensity profile of Hudson at 200m.

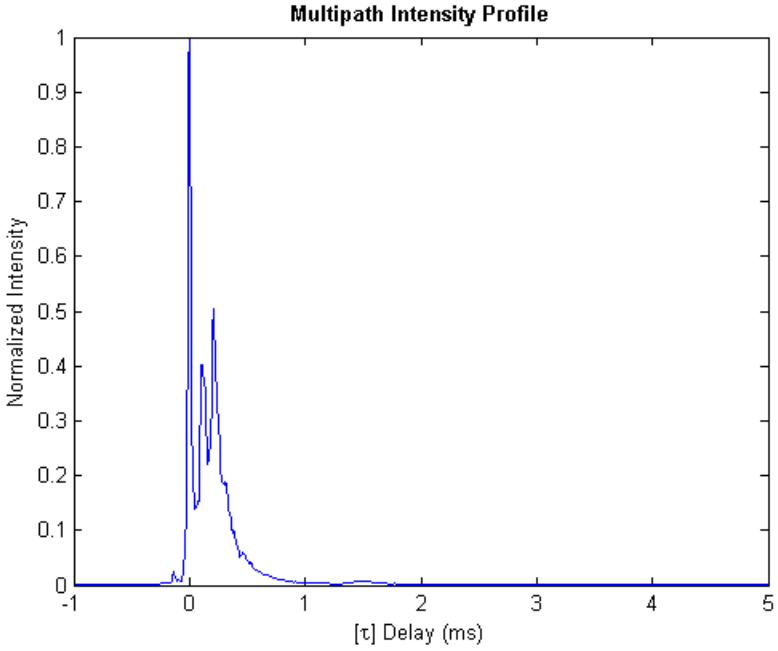

Figure 2-50: Multipath intensity profile of Hudson at 505m.

The MIP of the 200-meter channel, seen in Figure 2-49, appears to have only one arrival. However, upon enlarging the region around the peak, it is evident that there are two distinct arrivals separated by 25 μs. The MIP of the 505-meter channel, shown in Figure 2-50, reveals three distinct arrivals spanning slightly less than 0.5 ms.

Table 2-10: Delay Spread (ms) of Multipath Intensity Profile Computed with -20 dB Threshold

	Mean Excess Delay	RMS Delay Spread	Maximum Excess Delay (-10 dB)
200m	0.0907	0.1478	0.1800
505m	0.1789	0.1636	0.4150

Table 2-11: Doppler Shift and Spread (Hz) of Strong Multipath Arrivals

	200m				505m			
	Time (ms)	Intensity	Shift	Spread	Time (ms)	Intensity	Shift	Spread
Arrival 1	0.000	0.8136	-0.1945	2.6790	0.000	1.0000	-0.3642	2.8315
Arrival 2	0.025	1.0000	-0.2588	2.6948	0.105	0.4033	-0.3667	3.0616
Arrival 3	–	–	–	–	0.205	0.5041	-0.4556	3.0057

2.5.7 Spaced-Frequency Correlation Function

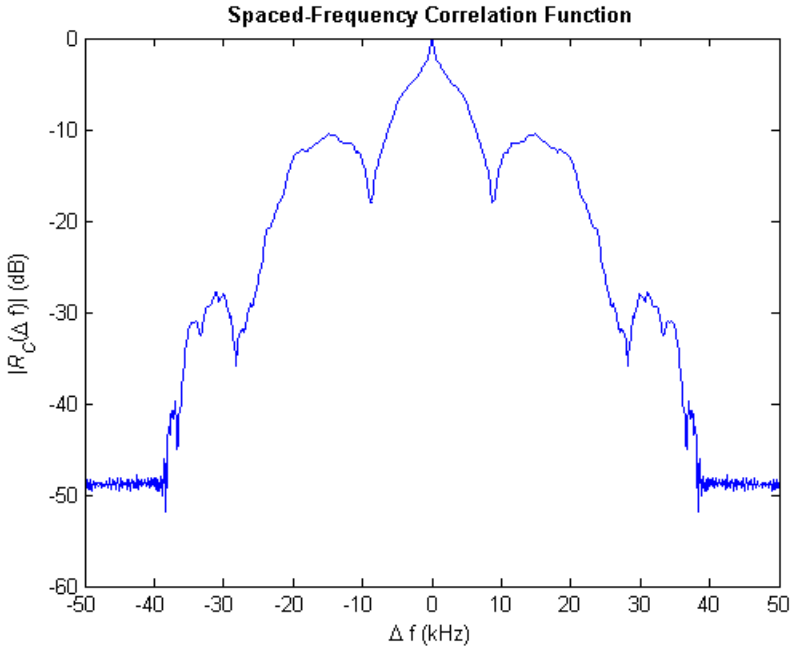

Figure 2-51: Spaced-frequency correlation function of Hudson at 200m.

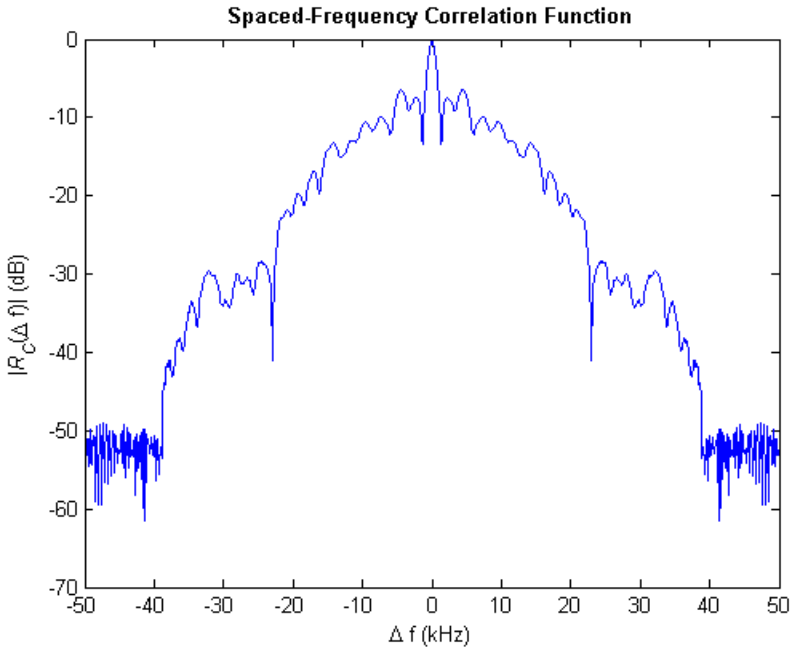

Figure 2-52: Spaced-frequency correlation function of Hudson at 505m.

Table 2-12: Coherence Bandwidth (Hz)

	-3 dB	-6 dB	-10 dB
200m	2331	8160	12490
505m	1166	1665	2165

Since the multipath spread of the channel is longer at 505 meters than it is at 200 meters, the coherence bandwidth of the 505-meter channel is less than that of the 200-meter channel. In fact, for the correlation between two sinusoids to remain within 3 dB of each other (correlation $\geq$ 0.5), the coherence bandwidth is reduced by half when going from 200 meters to 505 meters. As an aside, it should be noted that the correlation tapers off in last 10 kHz on each side of the graph in both graphs. This property is expected, since the channel was sounded over 80 kHz of the 100 kHz made possible with the 200 kHz sampling rate. Thus, there is no data for the remaining 20 kHz.

2.5.8 Doppler Power Spectrum

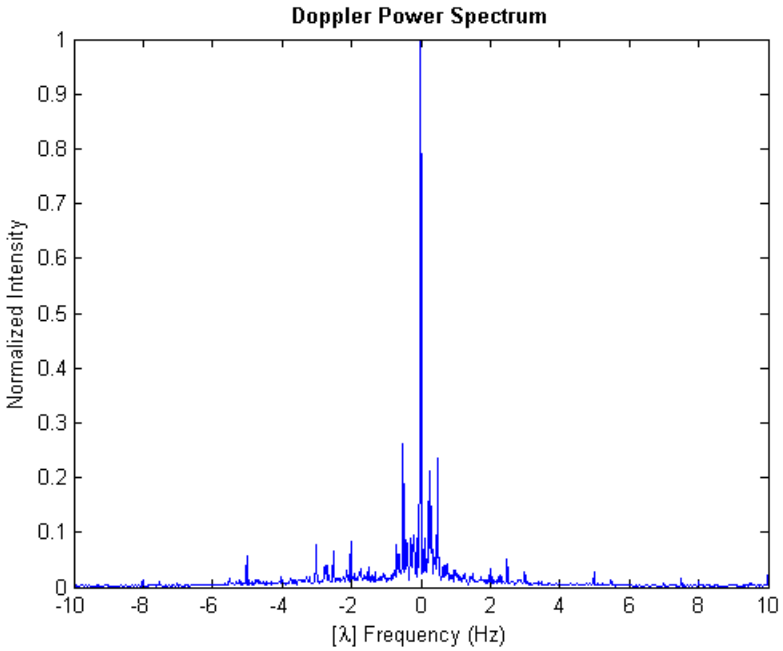

Figure 2-53: Doppler power spectrum of Hudson at 200m.

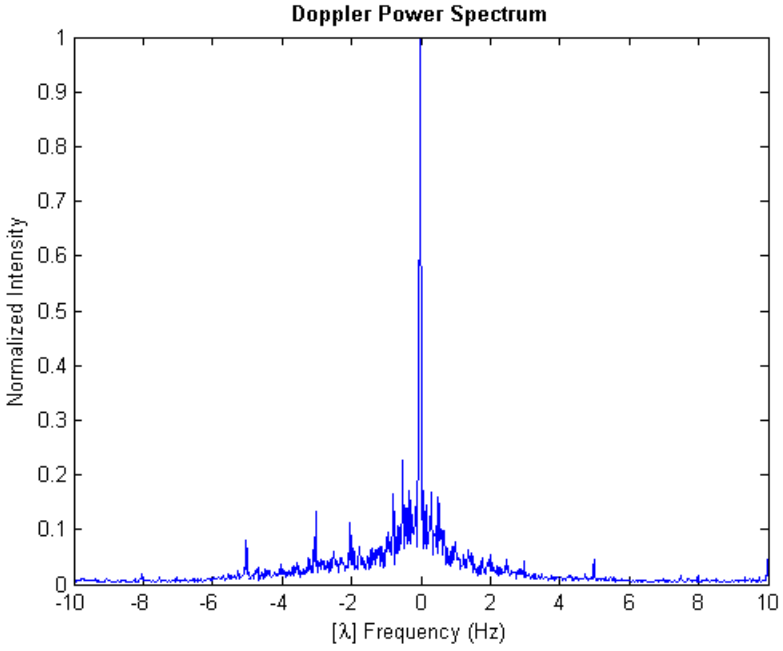

Figure 2-54: Doppler power spectrum of Hudson at 505m.

Table 2-13: Overall Doppler Shift and Spread (Hz)

	Shift	Spread
200m	-0.2357	3.3231
505m	-0.3381	3.3843

There is nothing remarkable about the Doppler power spectrum at either distance. The majority of the power is virtually centered on the 0 Hz lambda frequency, with some power distributed in the negative frequency range.

2.5.9 Spaced-Time Correlation Function

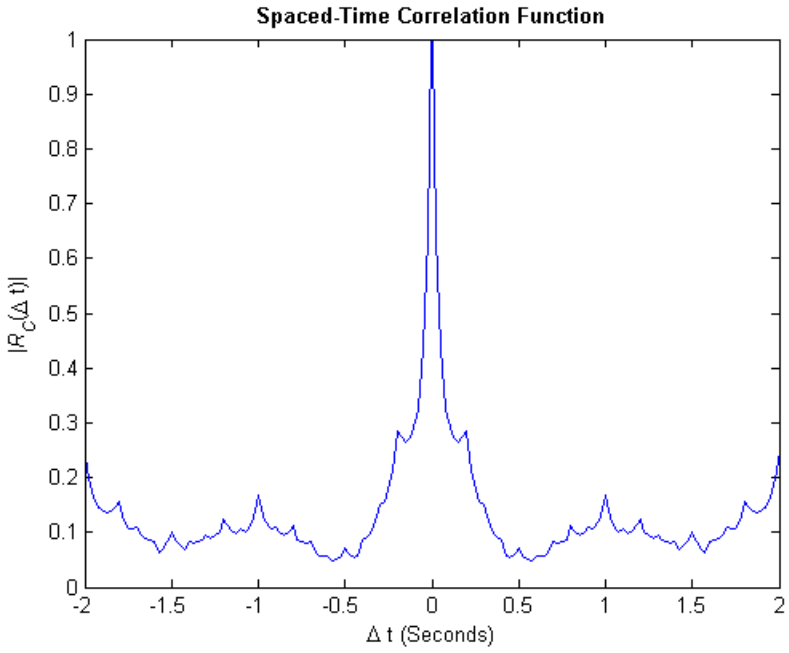

Figure 2-55: Spaced-time correlation function of Hudson at 200m.

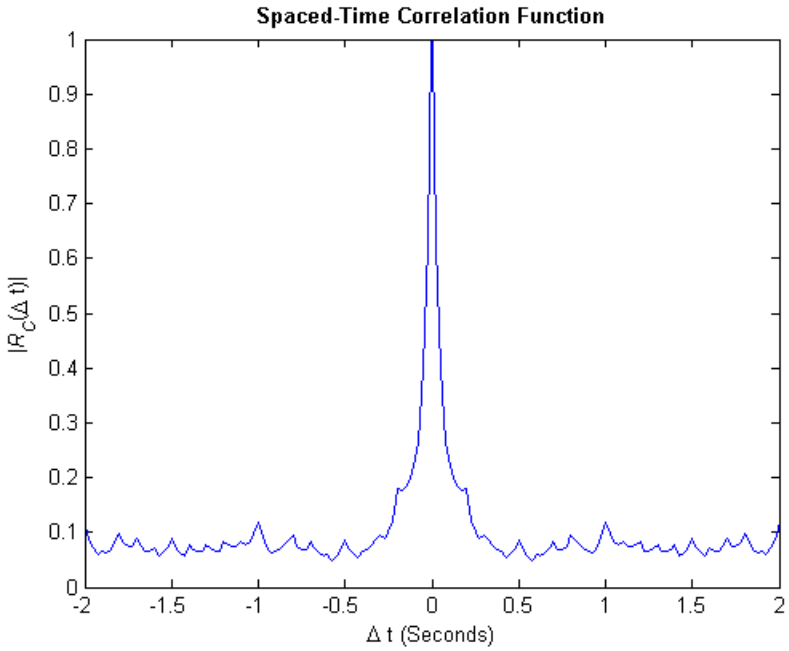

Figure 2-56: Spaced-time correlation function of Hudson at 505m.

Table 2-14: Coherence Time (ms)

	0.5 (-3dB)	0.25 (-6dB)	0.1 (-10 dB)
200m	50	400	699
505m	50	150	500

The coherence time of the channel at either distance is extremely short. In fact, the duration over which two sinusoids remain within 3 dB of each other (correlation ≥ 0.5) is only 50 ms.

2.5.10 Fading Characteristics

The comb signal containing 5 sinusoids – 35, 45, 60, 75, and 85 kHz, where the frequencies were chosen so that no harmonics overlap – has been analyzed to determine the type of narrowband fading that is present in the Hudson River estuary via the following procedure:

1) A 10^{th} order band-pass Butterworth filter with a passband of 2 kHz was applied to each of the tones.
2) The analytic signal x was obtained from the filtered data via the Hilbert transform.
3) The envelope of the signal was computed by taking its magnitude, as in Equation (2.22).

4) The fading envelope was normalized to zero mean.

Figure 2-57 shows the channel-induced amplitude fluctuations of the five sinusoidal components in the comb signal as well as the strongest component of the successive impulse response estimates. The fluctuations in the received signal level are huge, especially in the lower frequency bands, varying more the 80 dB during in a 10-second interval. The fluctuations in the amplitude of the wideband signal are less severe, with differences of 20 dB over 30 seconds. Since the channel was sounded 20 times per second, 30 seconds yields 600 impulse response estimates and, therefore, 600 amplitude values. Even though 30 seconds is three times greater than the analyzed section of the comb signal, it must be understood that the 10-second comb signal contains 2 million amplitude values and, hence, affords much greater resolution.

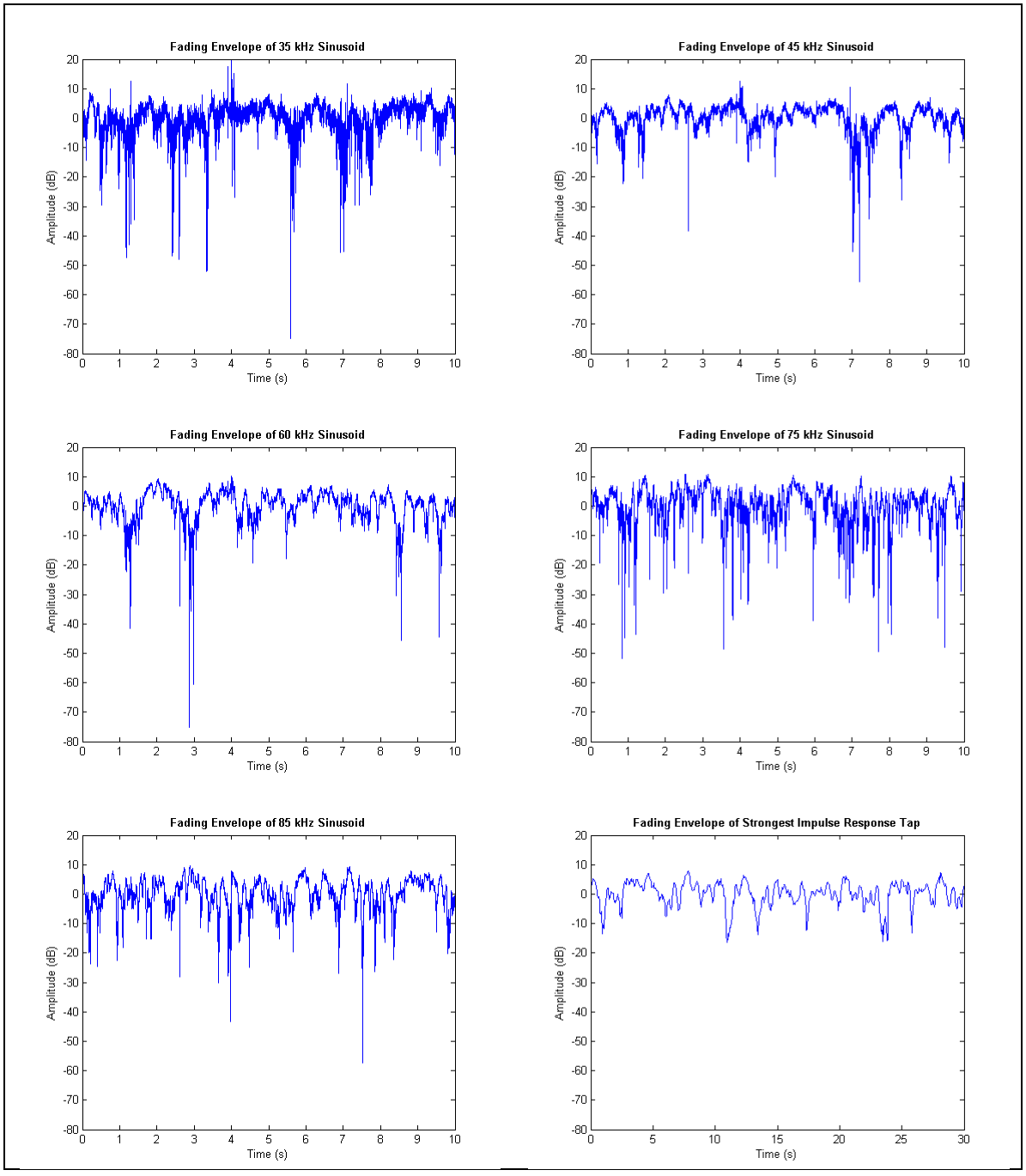

Figure 2-57: Fading envelopes in Hudson at 200m.

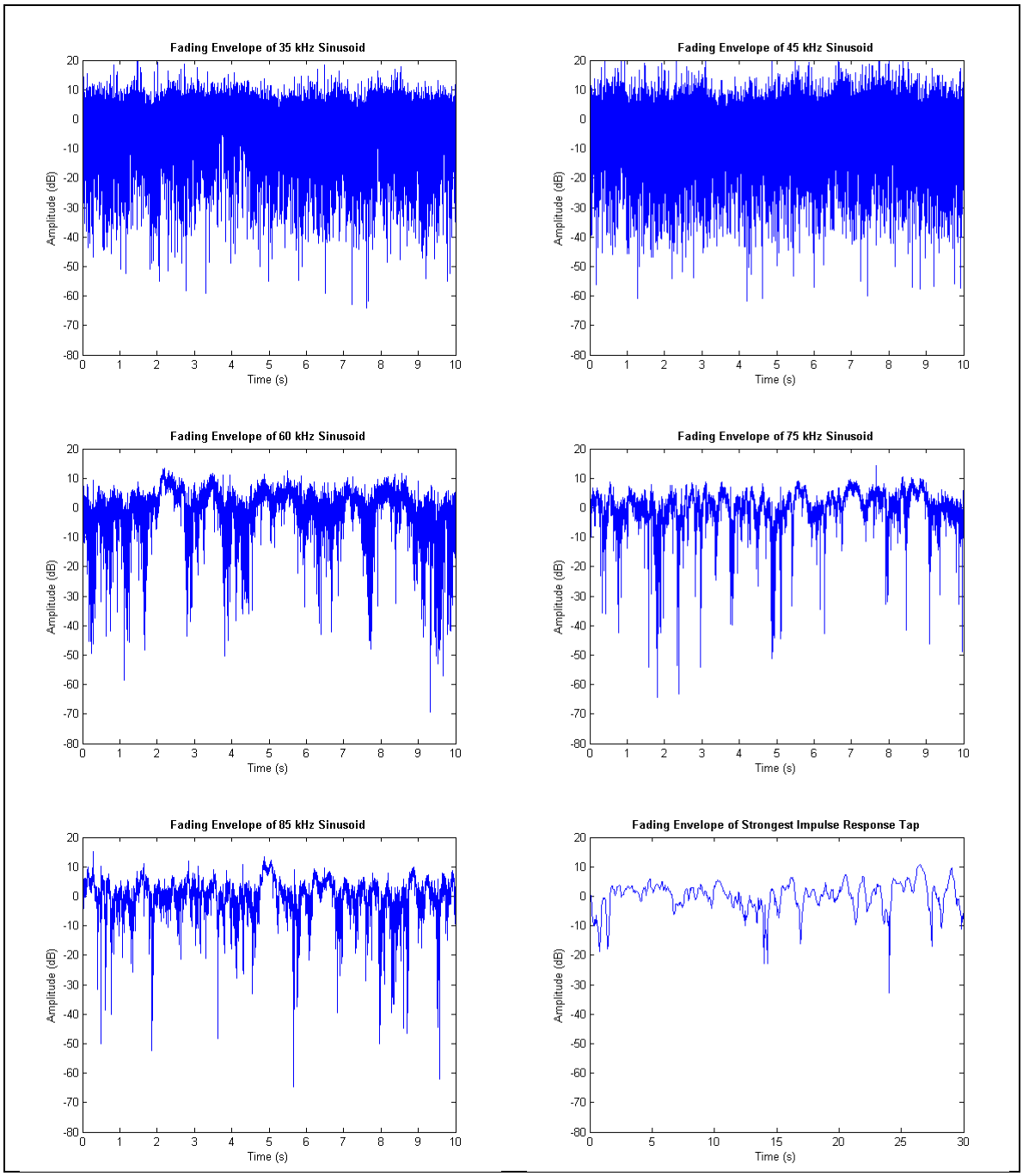

Figure 2-58: Fading envelopes in Hudson at 505m.

Figure 2-58 shows the amplitude fluctuations over the 505-meter channel. The severity of the fading is much stronger at the increased distance, especially with the 35 kHz and 45 kHz

sinusoids. Even the wideband signal exhibits fluctuations of some 40 dB, although as in the 200-meter channel, it still fared far better than any narrowband component.

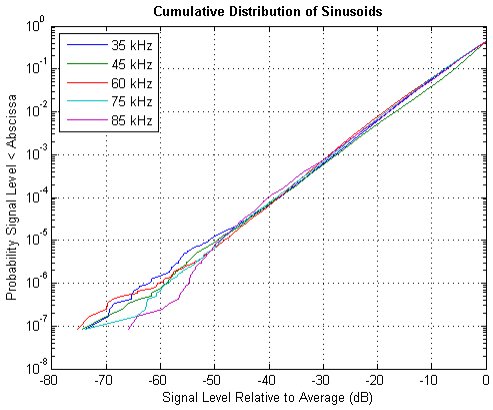

Figure 2-59: CDF for fading measurements at 200m.

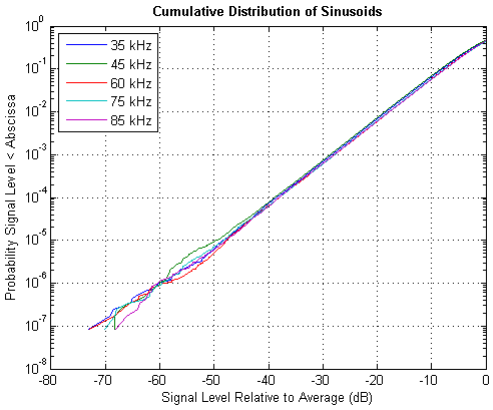

Figure 2-60: CDF for fading measurements at 505m.

Figures 2-59 and 2-60 show the cumulative distribution of amplitude levels of each of the 5 sinusoids taken over the full minute of data. It appears that at both distances the channel affected all the sinusoids equally, with approximately a tenfold decrease in the probability for every 10 dB decrease in the signal level, at least for down to -50 dB relative to the mean level. It should be noted, though, that the confidence intervals for points below -50 dB become significantly reduced, as there are relatively few low-amplitude samples in the 1-minute recording.

Maximum likelihood estimation was used to fit the data to the Rayleigh, Rice, and Nakagami-*m* distributions, which are commonly used to describe fading channels, and to other less likely potential distributions, including gamma, beta, and lognormal. The goodness of fit was tested with three different metrics. When working with the comb signal, the histogram of signal levels was divided into 100 bins. The histogram of the strongest impulse response tap, however, was divided into only 40 bins because there were significantly fewer data points. P is the probability distribution of the measurements; Q is the probability distribution of the fit. The first metric is Kullback-Leibler divergence [Kullback 1951], D_{KL}, defined as

$$D_{KL}(P \parallel Q) = \sum_i P(i) log_2 \frac{P(i)}{Q(i)}. \quad (2.24)$$

The second metric is the Bhattacharyya distance [Bhattacharyya 1943], D_B, defined as

$$D_B(P, Q) = -log_2(BC), \quad (2.25)$$

where *BC* is the Bhattacharyya coefficient, $\sum_i \sqrt{P(i)Q(i)}$. The third and final metric, D_{CRM}, is one based on the Bhattacharyya coefficient, proposed by Comaniciu, Ramesh, and Meer [Comaniciu 2003], defined as

$$D_{CRM}(P, Q) = \sqrt{1 - BC}. \quad (2.26)$$

Figures 2-61 and 2-62 show the histograms of the measurements and the curves corresponding to each of the six distributions for the 200-meter and 505-meter channels, respectively. In each figure, the first five subplots depict the probability distribution functions that correspond to the fading of each narrowband sinusoidal component in the comb signal. The last subplot shows the fading of strongest component of the successive impulse response estimates.

Tables 2-15 through 2-26 provide the values of the three metrics as well as the distribution-specific parameters obtained while fitting the distributions to the data. For each of the three metrics, lower values indicate less divergence from the actual data. In the tables, the best match for each metric is highlighted in yellow.

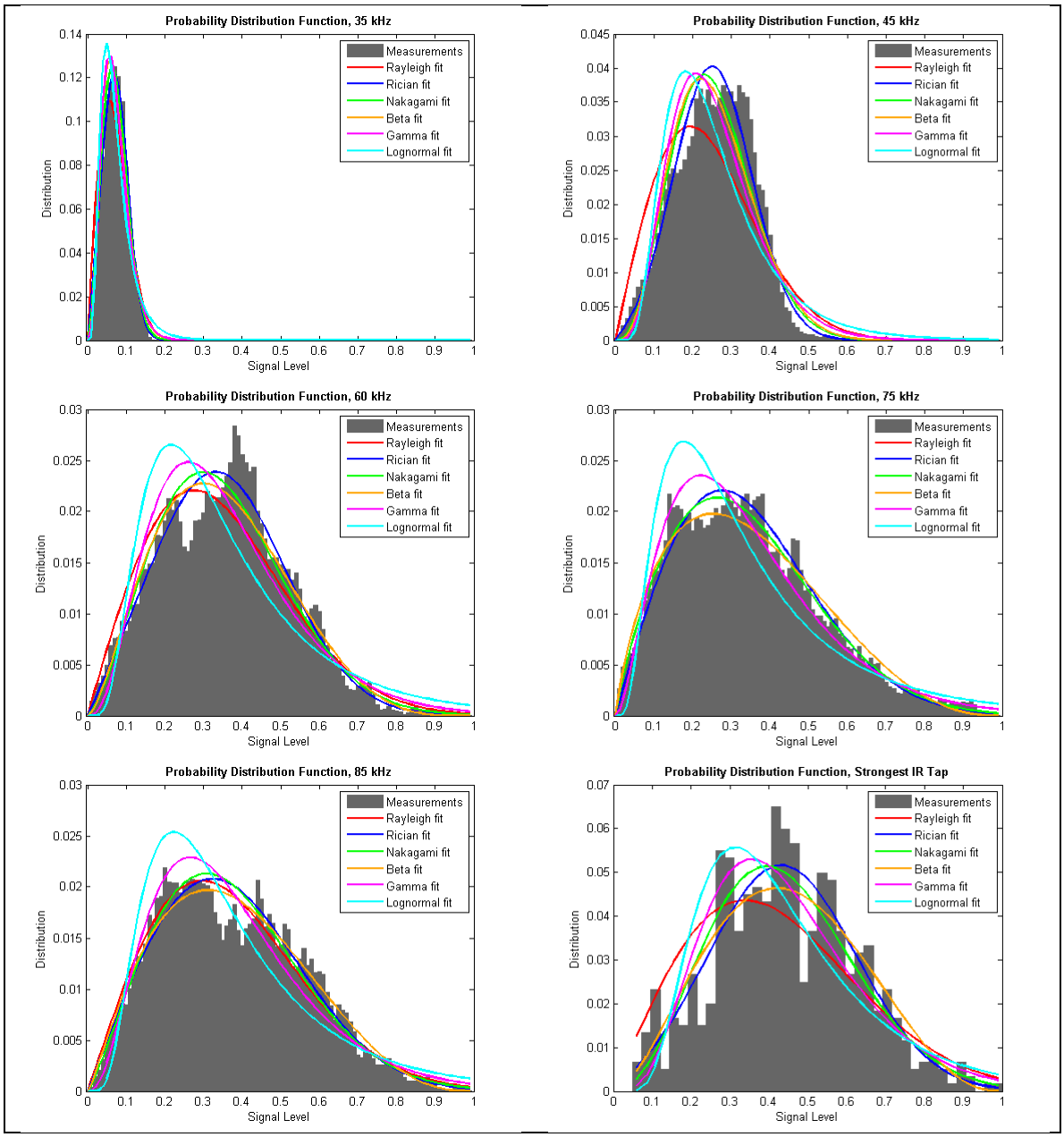

Figure 2-61: PDF of measurements and fits at 200m.

Table 2-15: Goodness of Fits, 200m, 35 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0964	0.0138	0.0974	[alpha = 4.2381, beta = 54.8056]
Gamma	0.1028	0.0161	0.1054	[alpha = 4.5298, beta = 0.0159]
Lognormal	0.4209	0.0434	0.1722	[mu = -2.7483, sigma = 0.5269]
Nakagami-*m*	0.0497	0.0051	0.0591	[m = 1.3867, omega = 0.0061]
Rayleigh	0.0774	0.0183	0.1123	[sigma = 0.0554]
Rice	0.0395	0.0026	0.0423	[s = 0.0602, sigma = 0.0354, K = 1.4411]

Table 2-16: Goodness of Fits, 200m, 45 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0771	0.0189	0.1142	[alpha = 4.2516, beta = 12.4813]
Gamma	0.1273	0.0325	0.1492	[alpha = 5.3128, beta = 0.0481]
Lognormal	0.2731	0.0653	0.2104	[mu = -1.4621, sigma = 0.4930]
Nakagami-*m*	0.0582	0.0147	0.1008	[m = 1.6674, omega = 0.0746]
Rayleigh	0.1547	0.0447	0.1746	[sigma = 0.1932]
Rice	0.0232	0.0058	0.0632	[s = 0.2300, sigma = 0.1042, K = 2.4372]

Table 2-17: Goodness of Fits, 200m, 60 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0284	0.0066	0.0675	[alpha = 2.6273, beta = 4.8210]
Gamma	0.0897	0.0226	0.1248	[alpha = 3.7575, beta = 0.0944]
Lognormal	0.2330	0.0470	0.1791	[mu = -1.1756, sigma = 0.5977]
Nakagami-*m*	0.0352	0.0090	0.0789	[m = 1.2156, omega = 0.1516]
Rayleigh	0.0494	0.0137	0.0973	[sigma = 0.2753]
Rice	0.0186	0.0047	0.0569	[s = 0.2883, sigma = 0.1850, K = 1.2142]

Table 2-18: Goodness of Fits, 200m, 75 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0243	0.0054	0.0612	[alpha = 1.8600, beta = 3.5367]
Gamma	0.0299	0.0066	0.0677	[alpha = 2.8129, beta = 0.1215]
Lognormal	0.1345	0.0219	0.1227	[mu = -1.2620, sigma = 0.6885]
Nakagami-*m*	0.0110	0.0027	0.0430	[m = 0.9149, omega = 0.1516]
Rayleigh	0.0162	0.0037	0.0509	[sigma = 0.2753]
Rice	0.0162	0.0037	0.0509	[s = 0.0004, sigma = 0.2753, K = 0.0000]

Table 2-19: Goodness of Fits, 200m, 85 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0185	0.0041	0.0536	[alpha = 2.1983, beta = 3.6467]
Gamma	0.0385	0.0095	0.0809	[alpha = 3.4408, beta = 0.1089]
Lognormal	0.1118	0.0229	0.1255	[mu = -1.1343, sigma = 0.6063]
Nakagami-*m*	0.0178	0.0045	0.0560	[m = 1.0919, omega = 0.1746]
Rayleigh	0.0198	0.0051	0.0596	[sigma = 0.2955]
Rice	0.0167	0.0043	0.0543	[s = 0.2565, sigma = 0.2332, K = 0.6050]

Table 2-20: Goodness of Fits, 200m, Strongest Impulse Response Tap

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0917	0.0222	0.1237	[alpha = 2.8064, beta = 3.4825]
Gamma	0.1186	0.0307	0.1450	[alpha = 4.9964, beta = 0.0890]
Lognormal	0.1940	0.0475	0.1800	[mu = -0.9139, sigma = 0.4981]
Nakagami-*m*	0.0828	0.0220	0.1229	[m = 1.5415, omega = 0.2300]
Rayleigh	0.1280	0.0364	0.1579	[sigma = 0.3391]
Rice	0.0688	0.0182	0.1119	[s = 0.3904, sigma = 0.1969, K = 1.9654]

The Rician distribution is the best match for most of the test cases at 200 meters. In the narrowband trials, it is the consistently the best fit for the fading of the 35, 45, and 60 kHz tones. Moreover, the Rician distribution is also the best fit for wideband fading according to all three goodness of fit metrics. At 75 kHz, the channel exhibits Nakagami-*m* fading, where $m = 0.9149 < 1.0$ indicates the fading is more severe than Rayleigh. At 85 kHz, the fading appears to be Rician, at least according to Kullback-Leibler divergence. The other two metrics give the Beta distribution a narrow lead over the Rician distribution, although $m = 1.0919 > 1.0$ in the Nakagami-*m* distribution indicates that the fading is at least less severe than Rayleigh. (Recall that not all sources agree that Nakagami-*m* can model the Rician distribution.)

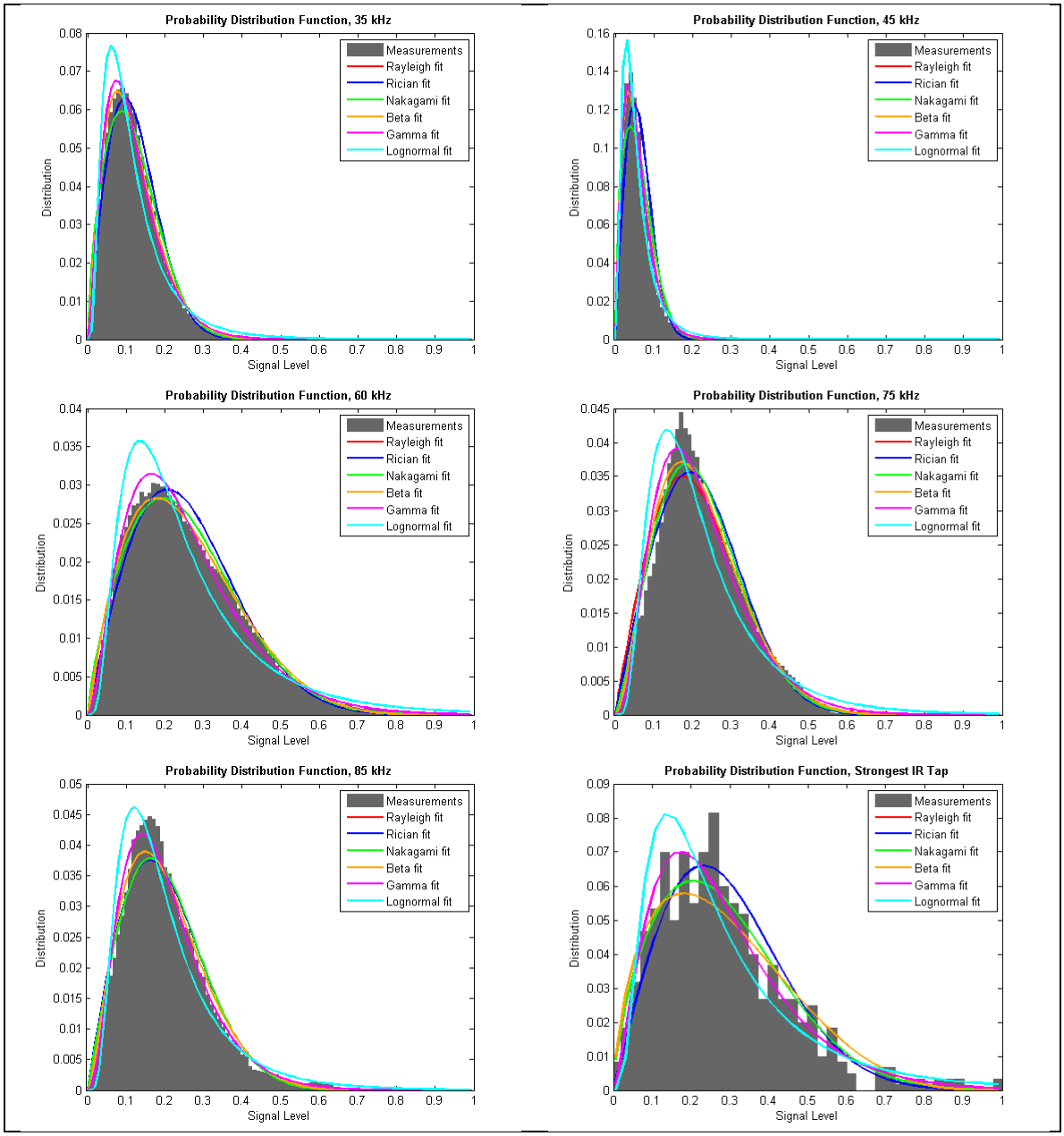

Figure 2-62: PDF of measurements and fits at 505m.

Table 2-21: Goodness of Fits, 505m, 35 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0287	0.0034	0.0486	[alpha = 2.4336, beta = 18.3310]
Gamma	0.0260	0.0034	0.0484	[alpha = 2.7512, beta = 0.0425]
Lognormal	0.2025	0.0193	0.1154	[mu = -2.3389, sigma = 0.6772]
Nakagami-*m*	0.0337	0.0047	0.0569	[m = 0.8529, omega = 0.0185]
Rayleigh	0.0517	0.0066	0.0677	[sigma = 0.0963]
Rice	0.0517	0.0066	0.0677	[s = 0.0001, sigma = 0.0963, K = 0.0000]

Table 2-22: Goodness of Fits, 505m, 45 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0727	0.0089	0.0782	[alpha = 2.2509, beta = 36.9124]
Gamma	0.0689	0.0081	0.0747	[alpha = 2.4133, beta = 0.0237]
Lognormal	0.4863	0.0138	0.0975	[mu = -3.0819, sigma = 0.7095]
Nakagami-*m*	0.0939	0.0127	0.0936	[m = 0.7174, omega = 0.0049]
Rayleigh	0.1947	0.0240	0.1284	[sigma = 0.0494]
Rice	0.1947	0.0240	0.1284	[s = 0.0000, sigma = 0.0494, K = 0.0000]

Table 2-23: Goodness of Fits, 505m, 60 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0092	0.0017	0.0347	[alpha = 2.1391, beta = 6.2493]
Gamma	0.0123	0.0024	0.0407	[alpha = 2.8385, beta = 0.0894]
Lognormal	0.1090	0.0178	0.1108	[mu = -1.5573, sigma = 0.6685]
Nakagami-*m*	0.0066	0.0016	0.0332	[m = 0.8998, omega = 0.0850]
Rayleigh	0.0128	0.0029	0.0450	[sigma = 0.2062]
Rice	0.0128	0.0029	0.0450	[s = 0.0002, sigma = 0.2062, K = 0.0000]

Table 2-24: Goodness of Fits, 505m, 75 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0266	0.0054	0.0614	[alpha = 2.9233, beta = 10.3616]
Gamma	0.0523	0.0124	0.0924	[alpha = 3.6043, beta = 0.0612]
Lognormal	0.2121	0.0402	0.1658	[mu = -1.6565, sigma = 0.6007]
Nakagami-*m*	0.0172	0.0040	0.0524	[m = 1.1366, omega = 0.0601]
Rayleigh	0.0232	0.0062	0.0656	[sigma = 0.1733]
Rice	0.0169	0.0040	0.0526	[s = 0.1564, sigma = 0.1334, K = 0.6878]

Table 2-25: Goodness of Fits, 505m, 85 kHz Sinusoid

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.0221	0.0045	0.0555	[alpha = 2.6744, beta = 10.5769]
Gamma	0.0174	0.0034	0.0489	[alpha = 3.3415, beta = 0.0602]
Lognormal	0.1103	0.0182	0.1121	[mu = -1.7606, sigma = 0.6046]
Nakagami-*m*	0.0265	0.0058	0.0634	[m = 1.0163, omega = 0.0521]
Rayleigh	0.0265	0.0059	0.0638	[sigma = 0.1614]
Rice	0.0265	0.0059	0.0638	[s = 0.0001, sigma = 0.1614, K = 0.0000]

Table 2-26: Goodness of Fits, 505m, Strongest Impulse Response Tap

	D_{KL}	D_B	D_{CRM}	Parameters
Beta	0.1272	0.0294	0.1421	[alpha = 1.7154, beta = 4.2336]
Gamma	0.0670	0.0222	0.1235	[alpha = 2.5826, beta = 0.1087]
Lognormal	0.1678	0.0376	0.1604	[mu = -1.4761, sigma = 0.7098]
Nakagami-*m*	0.0737	0.0222	0.1235	[m = 0.8219, omega = 0.1072]
Rayleigh	0.0951	0.0247	0.1304	[sigma = 0.2316]
Rice	0.0951	0.0247	0.1304	[s = 0.0001, sigma = 0.2316, K = 0.0000]

At 505 meters, the fading becomes noticeably more severe. The Gamma distribution is the best fit for the fading of the individual 35, 45, and 85 kHz tones as well as for wideband fading. At 60 kHz, the best fit is with the Nakagami-*m* distribution, where $m = 0.8998 < 1.0$ implies fading worse than Rayleigh. At 75 kHz, it's a tossup between Rician and Nakagami-*m*, where $m = 1.1366$. Interestingly, the fading at 75 kHz is less severe at 505 meters than it is at 200 meters.

2.5.11 Analysis and Implications for Communication

The maximum excess delay of the 200-meter channel is less than half that of the 505-meter channel, as reported in Table 2-10. Using the maximum excess delay as the lower bound for the duration of a symbol, the system can avoid ISI while transmitting up to about 5555 and 2410 symbols per second at 200 and 505 meters, respectively. The difference between the rms delay spread of the 200- and 505-meter channels is significantly less than the difference between the maximum excess delay values. Therefore, the estimated symbol rates based on the rms delay

spreads are expected to be closer together. At 200 meters, the predicted symbol rate is 677, while at 505 meters that number drops to 611 symbols per second.

The -3 dB coherence time of the Hudson at either distance is 50 ms. Unless the communication system is operating at a mere 20 symbols per second, the duration of a symbol will be significantly shorter than the coherence time. Thus, the channel is said to be one of slow fading. At 200 meters, the Hudson exhibits primarily Ricean fading. At 505 meters, the fading becomes more severe than Rayleigh, as the Gamma distribution is the best fit for most of the tests.

Given the channel's fading characteristics, any amplitude-based modulation technique should be avoided as its performance is expected to degrade under these conditions. If the required data rate results in frequency-selective fading across the channel, adaptive equalization, spread spectrum (either direct-sequence or frequency-hopping), OFDM (orthogonal frequency-division multiplexing), or pilot signal techniques can be considered [Sklar 2001]. Diversity techniques and the use of error-correcting codes can be exploited to reduce errors when the channel is in a deep fade. The most straightforward approach is to employ frequency diversity, that is, to transmit the same information on multiple carriers, where the separation between carriers equals or exceeds the channel's coherence bandwidth [Proakis 2008]. If computational complexity is not an issue, the optimum demodulator for use in a fading multipath channel, called the RAKE receiver [Proakis 2008], can be employed. The RAKE receiver decodes separate multipath components and combines the output of all the correlators, thus providing higher SNR at the decision stage.

The best approach for determining which of the aforementioned techniques works best is to conduct a field test in the channel of interest. Pregenerated signals can be transmitted through the channel, recorded, and returned to the lab for offline processing. These signals should include at least the following:

1) *M*-ary FSK, as in the Benthos modems [Benthos 2010].
2) PSK and QPSK (and perhaps even 8-PSK) in several frequency bands and at several symbol rates that clearly result in frequency-selective fading for some trials and flat fading for others.
3) Direct-sequence spread-spectrum on top of PSK (DSSS/BPSK).
4) Frequency-hopping (FH) PSK and FH-FSK, as in the Micro-Modem [Freitag 2005].
5) OFDM with PSK and/or QPSK modulated carriers.
6) Any of 1-5 preceded by a pilot signal (LFM chirp or other signal with good autocorrelation properties) in the relevant frequency band that can be used to estimate the channel.
7) Any of 1-5 with error-correcting codes such as Reed-Solomon or Turbo codes.

The time-invariant office test environment is too simplistic for testing these different techniques. The most straightforward approach of using a pilot signal to estimate the channel and applying a zero-forcing equalizer to the received signal actually works quite well, allowing for data rates of up to 6 kbps with binary FSK. The SNR in the tub is very high and the time-invariant characteristic implies that the channel's characteristics will remain the same over the length of a symbol, packet, and even packet train. Testing the complicated Hudson River estuary would be a more interesting project. In fact, it was part of the research plan until funding ran out. Therefore, since further experiments could not be conducted, related work on PHY layer techniques is presented here with a high-level discussion of expected results.

M-ary FSK works by transmitting *M* tones, where *M* is a multiple of two. Using more tones allows the modem to pack more bits in a symbol. With binary FSK, one tone represents a '0' and the other a '1'. With 4-FSK, each of the four tones can represent two bits at once – '00', '01', '10', and '11'. 8-FSK can transmit three bits per symbol, and so on. The advantage of this technique for higher values of *M* is that the symbol duration can exceed the multipath spread of

the channel and, hence, avoid ISI while offering the same data rate as a lower *M* value with faster signaling. Early work by a collaboration of Datasonics (later acquired by Benthos), Delphi Communications Systems, and Naval Command, Control & Ocean Surveillance Center, RDT&E Division (NRaD) describes a variation of *M*-ary FSK implemented in the type-A telesonar modem [Scussel 1997]. The previous ATM-850 modem uses 1-of-4 MFSK, where one tone in a group of four encodes two bits at a time. Independent groups of four tones are transmitted simultaneously to transmit up to 16 bits during the symbol time of 12.8 ms. The new modem has a symbol time of 25 ms, allowing for finer resolution in the frequency domain, with tones spaced by only 40 Hz. As a result, the type-A modem can use 128 frequencies in the same bandwidth, transmitting up to 60 bits in one symbol frame. In general, *M*-ary FSK is considered to be mature, reliable technology that works well in shallow water environments with multipath propagation.

The WHOI (Woods Hole Oceanographic Institution) introduced a version of the Micro-Modem in 2005 that supports both FH-FSK and PSK data transmission. While the default data rate on this modem is listed at 80 bps [Freitag 2005], FH-FSK can transmit at higher speeds. In a single-user system, frequency hopping is exploited by allowing the channel to clear between successive transmissions in the same frequency bin, thus eliminating ISI at the receiver [Parrish 2007]. The duration of a symbol can be shorter than multipath spread, allowing for higher data rates, as long as the amount of time between each repeated use of a frequency bin exceeds the delay spread. As an added advantage, frequency hopping can be exploited in a multi-user environment in order to share the available bandwidth by allowing a number of users to transmit simultaneously on different hopping patterns [Parrish 2007].

The PSK data transmission on the Micro-Modem offers much higher data rates of 300-5000 bps [Freitag 2005]. Micro-Modems equipped with the floating-point coprocessor can re-

ceive the PSK transmissions via a DFE. By 2008 the Micro-Modem had been enhanced with QPSK signaling and Reed-Solomon codes [Freitag 2008]. The implementation of the DFE is similar to what Stojanovic described in her original work [Stojanovic 1993, 1994] except that the filter coefficients are updated with the adaptive step-size LMS[8] algorithm instead of RLS[9] and there is only one phase-locked loop[10] (PLL) applied after the feedforward channels are combined instead of one for each hydrophone channel. The modem was tested in Narragansett Bay, very close to the University of Rhode Island Graduate School of Oceanography facility in the March RACE 2008 experiment and again in a shallow area of the Atlantic Ocean south of Cape Cod in Massachusetts in the October SPACE 2008 experiment. The water depth was very shallow in both cases – 10 meters for RACE 2008 and less than 15 meters for SPACE 2008. The receiver arrays contained 12 hydrophone elements, 4 of which were selected to maximize the aperture. Under this experimental setup, which resulted in approximately 5000 bps burst throughput in the actual modem, it was observed that packets were decoded correctly when symbol SNR levels greater than 7 dB were obtained at the output of the equalizer. Thus, it seems that high data rate phase-coherent communication is indeed possible in shallow water environments when coupled with the complexity of a DFE, error-correcting codes, and a receiver array.

Sozer et al. described a communication system based on direct-sequence spread spectrum signaling with a RAKE filter at the receiver [Sozer 1999]. In an experiment performed in the Baltic Sea in March 1999, the authors ran two trials in a 3 km channel, where the second was performed with 12 dB less output power than the first. They estimated the multipath spread of the channel to be about 2.5 ms, for which 6 RAKE receiver taps were required, and found no errors out of 200 bits in either trial.

[8] The least mean squares (LMS) algorithm is used in adaptive filters to find the filter coefficients that correspond to producing the least mean squares of the error signal (difference between the desired and the actual signal).
[9] The recursive least squares (RLS) algorithm is used in adaptive filters to find the filter coefficients that correspond to recursively producing the least squares (minimum of the sum of the absolute squared) of the error signal.
[10] A phase-locked loop, implemented in either hardware or software, tracks changes to the incoming signal's phase.

Yang and Yang analyzed direct-sequence spread-spectrum signals collected from the TREX04 experiment to determine the BER as a function of the input SNR for a single receiver [Yang 2008]. The experiment was conducted in April 2004, off the coast of New Jersey, south-west of the Hudson Canyon. The DSSS signals were centered at 17 kHz and had a bandwidth of 4 kHz. The transmitted symbols were spread with an m-sequence of 511 chips using BPSK modulation. The symbol duration was 127.8 ms, much longer than the channel's multipath delay of 25 ms, which translated to a data rate of approximately 8 bps. A total of 1160 packets of data were generated by adding ambient noise data collected at sea to the signal data (in post-processing) to create signals with different input SNR, some as low as -15 dB. The authors analyzed two methods, both of which used a time-updated channel impulse response estimate as a matched filter to mitigate the multipath-induced interferences. The first method required an independent estimate of the time-varying channel impulse response function; the second method used the channel impulse response estimated from the previous symbol as the matched filter. The first method produced an average BER $<10^{-2}$ for input-SNR as low as -12 dB, while the second method yielded similar performance for input-SNR as low as -8 dB. Thus, DSSS can be useful in situations where noise is dominant or covert passing of short messages is required. It is clear, though, that the data rate will be significantly compromised.

Stojanovic and Freitag demonstrated promising results for a multi-user system with direct-sequence CDMA[11] [Stojanovic 2006]. Performance of two different receivers based on symbol decision feedback (SDF) and chip hypothesis feedback (CHF) was demonstrated in a four-user scenario, using experimental data obtained over a 2-km shallow-water channel. At a chip rate of 19.2 kilochips per second (kc/s) with QPSK modulation, excellent results were achieved at

[11] Code division multiple access, or CDMA, employs spread-spectrum technology and a coding scheme (where each transmitter is assigned a code that is orthogonal to the others so that the cross-correlation of any two codes is close to zero) to allow multiple users to be multiplexed over the same physical channel.

an aggregate data rate of up to 10 kbps. However, the results were obtained with a stationary system, and the received power for each of the users was equal, a condition which will almost never be the case in practice.

Stojanovic proposed a low complexity adaptive algorithm for detecting OFDM signals in Doppler-distorted time-variant multipath channels [Stojanovic 2006a]. The receiver performs MMSE (minimum mean square error) combining of signals received across an array, using adaptive channel estimation. Nonuniform Doppler compensation across subbands is performed using a single adaptively estimated parameter representing the Doppler rate. A field test was conducted in September 2005 in Buzzards Bay, Massachusetts, to measure the performance of the algorithm. The channel was 12 meters deep and spanned 2.5 km. QPSK modulation with a varying number of carriers was used, occupying 24 kHz of acoustic bandwidth. No decision errors occurred with up to 1024 carriers, yielding an overall bit rate of 30 kbps.

A group from Scripps Institution of Oceanography presented OFDM performance results from the KAM08 experiment, conducted off the western side of Kauai, Hawaii, in June 2008 [Kang 2009]. The experiment was performed in 110-meter deep shallow water over a distance of 4 km. The signal was sent from one transmitter to receiver array of 16 elements. The authors compared three channel estimation algorithms and found that sparse channel estimation using Orthogonal Matching Pursuit (OMP) [Pati 1993] performed the best for both single and multiple receiver configurations. After combining the received signals from the hydrophone array, the authors obtained a 0.01% BER at a data rate of 10 kbps for QPSK without coding. Unencoded QAM produced a BER of about 12% at a data rate of 20 kbps. However, when iterative channel estimation was applied, error-free transmission was obtained within two iterations.

In view of the related work presented here, one can conclude that there are many techniques that allow for higher data rates than a channel's MIP will allow with binary signaling.

Also, it is clear that these techniques come at a cost, both in terms of algorithmic complexity and the need for an array of hydrophones at the receiver. After reading these papers, though, one does not walk away with an understanding of which methods can provide reliable communication at the highest data rates in a given channel such as the Hudson. There are two obvious drawbacks to the way research in this field is currently being directed. Every experiment is conducted in a different body of water, and no single experiment compares techniques (such as FH-FSK, QPSK with DFE, and OFDM) that have vast differences. Instead smaller variations in algorithms, such as channel estimation subroutines for OFDM, are investigated. While it seems that OFDM is rapidly becoming the method of choice for underwater acoustic communication, it is clear that the field is not yet ready to see how OFDM (or any other technique for that matter) performs in a network as opposed to just a point-to-point link. For example, with OFDM, can the system be designed so that all the carriers in the network remain orthogonal in environments with Doppler distortion? How will the system perform if some nodes are mobile? The research community would really benefit from a channel characterization study that is immediately followed by a comparison of communication techniques in the same channel so that the relationship between the channel's properties and technique can be established. Such a repository of information will truly assist those designing protocols for software defined radio platforms that allow the system adapt to a given environment. However, for the underwater acoustic communication community, it seems that such an approach is still years away.

2.5.12 Limitations of Experimental Setup

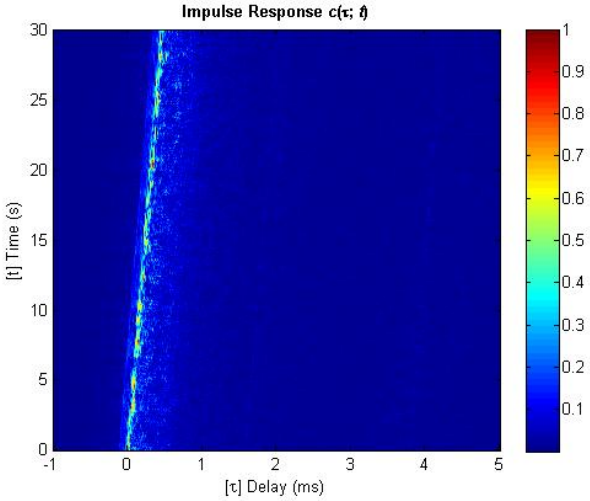

Figure 2-63: Successive impulse response estimates with uncorrected clock skew.

There are several aspects of the experimental setup that contributed to error in the estimates of the channel characterization functions. The first and most significant is clock skew in the DAQ boards. The NI PCI-6221 has a timebase stability of 50.0 parts per million (ppm). The NI PCI-6123 is even worse, with a timebase stability of 100.0 ppm. These numbers indicate that the clocks on the transmitter and receiver can be noticeably out of sync with one another. In fact, with the two devices working in tandem to sound the channel, the combined error was approximately an extra 3 samples every second. With a sampling rate of 200 ksamples/sec, this amount corresponds to a clock skew of 0.0015%, which is well within the devices' specifications but substantial enough to cause problems. Figure 2-63 shows the successive impulse response estimates in the Hudson at 200m without compensating for clock skew. As time goes on, it takes progressively longer for the correlation to occur. Interestingly, Dessalermos observed a similar slope in his impulse response estimates but attributed it to boat drift [Dessalermos 2005]. As for the Hudson experiment, there was no drift, since both boats were anchored. Moreover, the phenomenon was also observed on subsequent indoor experiments in the Davidson Laboratory towing tank,

where the transmitter and receiver were mounted to fixed apparatus in a perfectly still environment.

Many high quality DAQ boards have an input for GPS time synchronization or some other form of external clocking. Unfortunately none of the devices used in this experiment has external clocking capabilities. If that option were available, the clocks of the transmitter and receiver could have been synchronized to the same source, eliminating the clock skew problem altogether.

Another possible solution to the clock skew problem is to resample the recorded data. Resampling involves interpolation and decimation to change the sampling rate by a rational factor (as well as filtering to remove aliasing). Since the recorded data gains approximately an extra 3 samples per second, it needs to be reduced to a sampling rate of 199,997 samples per second. Because there are no common factors in 199,997 and 200,000, resampling is accomplished by interpolating by a factor of 199,997 and then decimating by a factor of 200,000. Not only are the computational requirements burdensome, but virtually every point in the new signal is interpolated, thus introducing error.

Even if the error caused by interpolation is negligible, there is no way to determine precisely what the sampling rate should be. For example, perhaps the clock skew is actually 2.x or 3.x samples per second. Given all these unknowns and potential sources of error, the simplest approach was chosen – to delete 3 samples per second from the recording. The first sample of three chirp signals spanning the 1-second interval was eliminated. By dividing the interval into three equal pieces, the skew within each second is kept to a minimum. Also, deleting the first sample of the chirp and not one in the middle minimizes the error, since the weak intensity of the reference and received signals at the low end of the spectrum has little effect on the overall corre-

lation. In addition, deleting the first sample avoids creating a phase discontinuity in the chirp signal.

The second problem with the channel sounding experiment was with the emitter and the inadequate means of obtaining its frequency response. As seen in Figure 2-42, the frequency response is irregular, varying 35 dB over the 20-100 kHz range spanned by the chirp signal. The worst part is that the 1-meter reference signal is not an accurate indication of the device's true properties. Because the estuary is so shallow at the test site, the reference signal also included reflections from the surface and bottom that combined constructively and/or destructively, contributing to the dips and valleys seen over the 80 kHz range. Instead, one of two options should have been taken. The simplest and best approach would have been to utilize a reputable manufacturer's calibrated emitter with known specifications. Alternatively, the custom emitter could have been taken to a facility with an anechoic underwater chamber for analysis. Either way, there would have been greater confidence in the results.

2.6 Related Work

Loubet and Jourdain estimated the impulse response of a shallow water channel in the Northern Sea. They used a Maximum Length Binary Sequence (MLBS) with a BPSK carrier of 550 Hz to sound the channel which was 500 meters deep and 4 kilometers wide (distance between the transmitter / receiver pair) [Loubet 1993]. The authors discovered three multipath arrivals that fluctuated in delay and amplitude due to surface reflections. They also plotted the time evolution of the phase values corresponding to the three multipath components and found them to be stable, even though the amplitudes fluctuated.

In May 1997 Cook and Zaknich sounded the Fremantle Fishing Boat Harbour in Western Australia [Cook 1998]. The channel was approximately 150 meters long and 4 meters deep, and both the transmitter and receiver were positioned at a depth of 1.5 meters. The chirp signal used

in this experiment had a duration of 10.306 ms and a bandwidth of 6 kHz centered at 17550 Hz and was repeated every 100 ms for 1 minute. The authors illustrated four impulse response estimates, taken at 0, 5, 25, and 50 seconds into the experiment. The plots consistently showed two strong multipath components occurring within a 1-ms window followed by several other weak arrivals that vary in delay and amplitude. One noteworthy aspect of this work is that the authors used inexpensive, non-specialized hardware including a laptop PC, power amplifier operating in switching mode, and piezo-electric transducer, to sound the channel.

Chitre, Potter, and Heng conducted an experiment in February 2004 to measure the time-variability of the impulse response of Singapore waters [Chitre 2004]. The hydrophone was attached to the bottom of a 4-meter pole mounted to an anchored barge. The emitter was also attached to the bottom of a 4-meter pole mounted to the research vessel, which made transmissions at distances of 50, 100, 550, 780, and 1020 meters from the barge. The sounding signal was a 30-ms DSSS/BPSK waveform with a bandwidth of 40 kHz centered around 40 kHz. It was repeated 100 times at a rate of 10 Hz at each of the five locations. The recorded signal was sampled at 250 ksamples/sec. The multipath arrivals were detected using a sign correlator.

At 50 and 100 meters, the authors noted that a ray model fit the observed data well. The surface-reflected arrival, which appeared less than 0.25 ms after the direct arrival, suffered very little attenuation, while the bottom-reflected arrival, which appeared 3.6 ms after the direct arrival at 50 meters and 1.8 ms at 100 meters, was 10 – 15 dB lower than the direct arrival. Other weaker reflections caused by multiple surface-bottom interactions and reflections from nearby objects were occasionally observed. At the remaining distances, the direct and surface-reflected arrivals could no longer be independently resolved. The authors also examined the fading characteristics of the main arrival and discovered fading slightly less severe than Rayleigh when the direct arriv-

al was fully resolvable and fading more severe than Rayleigh when the direct and surface-reflected arrival overlapped, most likely due to destructive interference.

In his master's thesis, Desselermos analyzed data gathered in the New England Shelf during the April 2000 ForeFRONT-2 experiment [Desselermos 2005]. During this trial the receiver was mounted 30 meters above the shelf's floor. The transmitter was deployed over the side of the vessel at a depth of 20 meters. The channel was sounded at increasing distances in the range of 700 – 6550 meters from the fixed receiver. CTD measurements were taken prior to each sounding.

The transmitted signal consisted of many different waveforms including 50-ms LFM chirps in the range of 8 – 16 kHz, a comb signal consisting of 17 tones where each was separated by 500 Hz in the 8 – 16 kHz range, and DSSS/BPSK signals generated with Gold sequences that correspond to bit rates of 10 – 400 bps. At each distance Desselermos estimated the time-variant impulse response using both the LFM chirp and DSSS signal and computed the scattering function and multipath intensity profile from the impulse response estimates obtained with just the DSSS signal. He then used the BELLHOP model [BELLHOP 2010] to plot the eigenrays of the channel and derive the theoretical MIP based on the arrival time of each eigenray. At 700 meters the delay spread exceeded 15 ms, while at greater distances it tended to be shorter due to the attenuation of multiple reflections. There was good correlation between the theoretical MIPs and those calculated from measurements.

Aik, Sen, and Nan presented experimental analysis of medium frequency (9 – 28 kHz) channel measurements in very shallow waters of 15 – 30 meters over the range of 80 – 4000 meters in the coastal seas of Singapore [Aik 2006]. The channel was sounded with broadband BPSK signals modulated with m-sequences. The symbol rate was 4625 bps, and the carrier frequency was 18.5 kHz. Similar to Desselermos's findings, the authors noted that the delay spread general-

ly decreases as the distance increases, with the excessive time delay (maximum excess delay set to 20 dB) ranging from 5.5 ms at 80 meters down to 0.5 ms at 4000 meters, with a slight increase to 7 ms occurring at 130 meters. They also observed that coherence time increased over longer distances. At 80 meters the coherence time was approximately 0.11 s, while it was 0.5 s at 4000 meters, even with the ship drifting during that particular sounding. The authors also determined that the channel exhibited a mix Rayleigh and Ricean fading, with Ricean being the better fit for most of the longer distances, and that the Gaussian distribution was a poor fit for the ambient noise present in Singapore waters.

In August 2008 Kim et al. conducted a channel sounding experiment in Jinhae Bay, near the southern cost of Korea, where the water is about 20 meters deep [Kim 2009]. The receiver was fixed while the transmitter was moved to distances of 105, 193, 304, 425, 600, and 1000 meters away from the receiver to sound the channel. An ITC-1001 omnidirectional projector was utilized for transmission while a vertical array of 8 B&K Type 8103 hydrophones was employed as the receiver. Two signals were used to sound the channel – a pure tone signal consisting of 5 frequencies, each separated by 5 kHz, in the 20 – 40 kHz range and a broadband BPSK signal modulated by an m-sequence of 1023 bits at 5000 sps.

Kim et al. have stated that frequency and coherence time tend to be inversely related, though the plots shown in Figure 5a only partially support that claim. As for the relationship between distance and coherence time, at 105 and 600 meters they calculated a shorter coherence time than at the remaining distances. The fading distribution was Rician at mid-range distances; however, the Rayleigh distribution was a better fit at both extremes.

2.7 Summary and Future Work

This chapter discussed the characterization of underwater acoustic channels for the purpose of estimating the channel's impact on the performance of a digital communication system.

It provided an in-depth comparison of various sounding signals, description of the procedure used to gather data about the channel, and explanation of the signal processing that converts raw data into meaningful characterization functions. The analyses of two WSSUS underwater channels, an artificial office test setup and the Hudson River estuary, were presented. In both cases, the channel's scattering function and all derived functions were computed. Values for Doppler shift and spread, delay spread, coherence bandwidth, and coherence time were provided. In addition, various distributions were fitted to amplitude fluctuations, and the channel was found to degrade from Ricean to Gamma fading (worse than Rayleigh) over increasing distances in the Hudson.

Future work in this area is broken down into the following four steps, which are listed in order of increasing complexity:

1) Determine the true frequency response of the emitter. As mentioned in Section 2.5.12, the response of the emitter obtained at 1 meter is not accurate, since constructive and destructive interference from multiple reflections cannot be eliminated from the test setup. The transducer should be taken to a facility with an anechoic chamber for analysis or, if this is not possible, a device with known specifications should be deployed instead.
2) In addition to sounding the channel with LFM chirps, both a DSSS/BPSK signal based on a PN sequence and white noise should be used. Before attempting to use them in the Hudson, these other signals should first be tested in the time-invariant tub to see how the impulse response estimates derived from them compare to those obtained when using LFM chirps.
3) When going out into the Hudson, characterize the channel at several other distances – some less than 200 meters and some greater than 505 meters. With only two points in the data set, making generalizations is virtually impossible. In addition to sounding signals, waveforms containing modulated data should be transmitted and recorded for offline analysis. It would

be interesting to see how the data rates predicted by the characterization functions differ from those actually obtained in the channel.

4) Deploy buoys complete with computers, transducers, and standard RF-based wireless communication to perform channel characterization remotely at any time. Having the ability to gather data without the hassle of manually deploying two boats and a team of scientists would be a huge asset to the underwater communications research community. Data can be gathered at any time of the day during any day of the week in any season of the year by simply issuing a command to power up the system. Once data is recorded, it can be downloaded via 802.11 to a computer in the lab for offline processing. Such a system truly affords the ability to make statistical observations and generalizations. Of course, many details about the design of the system for long-term deployment, including buoy location, method of battery replenishment, and thermal considerations, must be carefully engineered.

Chapter 3
Simulation of Underwater Channel and Physical Layer

3.1 Purpose

The work described in this chapter takes impulse response measurements such as those from Chapter 2 and applies them to an underwater channel model for use in the OMNeT++ discrete event simulator [OMNeT 2010]. The basic idea is to simulate any underwater channel based on real measurements to recreate the distortion the actual channel would impose on a signal. Through a mathematical process called convolution, the impulse response estimates obtained from the channel sounding experiment serve as the basis for this model. In this simulation the application layer of a node generates real data packets, which get converted into modulated acoustic signals. These modulated waveforms are "mixed" with the channel's properties and sent to a receiver implemented fully in software, where the actual BER is computed. This form of simulation results in more accurate BERs than what is currently being generated by underwater network simulators that derive a BER based solely on SINR. The simulation currently offers PSK (phase-shift keying) and FSK (frequency-shift keying) transmission and means of adjusting the sampling rate, carrier frequency, and symbol rate among other configuration parameters.

3.2 Related Work

BELLHOP is a program, originally written in Fortran but now also available in MATLAB, that performs two-dimensional acoustic ray tracing for a given sound speed profile or sound speed field in ocean waveguides with flat or contoured absorbing boundaries [Porter 1994; Rodríguez 2008]. It can output ray coordinates, travel time, amplitude, eigenrays, acoustic pressure, and transmission loss. Though this program is not specifically oriented toward underwater

communication, it provides useful information about the channel and has been integrated into other simulations described below. Figures 3-1 and 3-2 as well as Figure 1-2 from Chapter 1 depict some of BELLHOP's outputs.

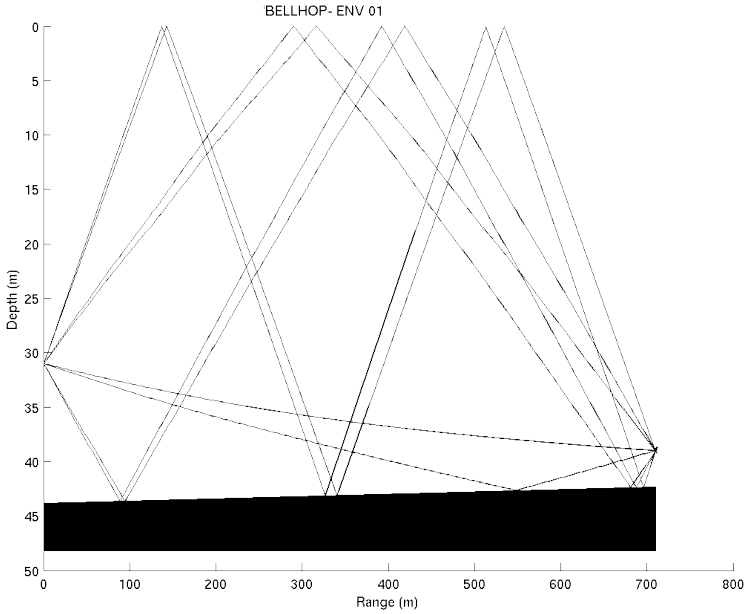

Figure 3-1: Eigenrays characterizing the acoustic propagation over 700 meters in the New England Shelf traced using BELLHOP [Dessalermos 2005].

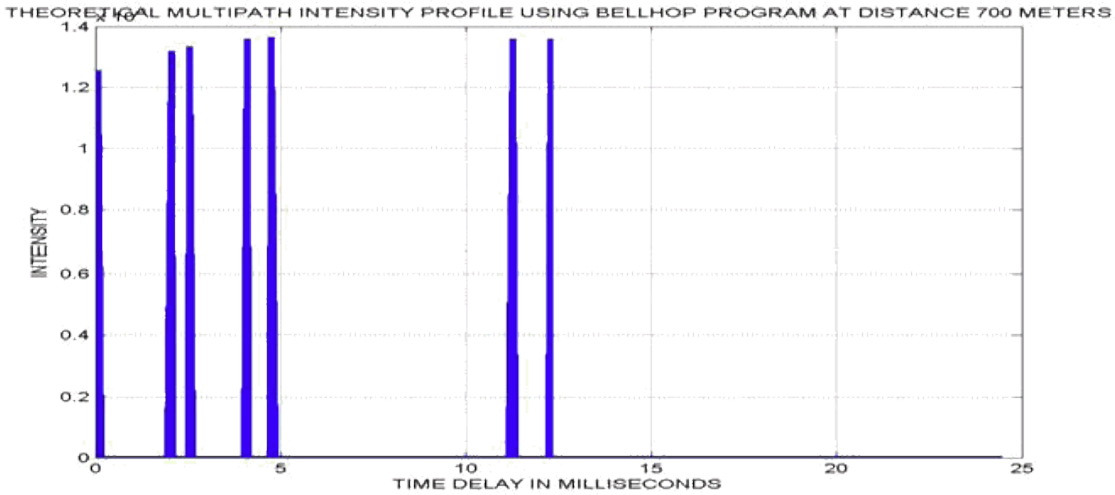

Figure 3-2: BELLHOP theoretical estimate of multipath intensity profile of New England Shelf at 700 meters [Dessalermos 2005].

Diamant and Chorev describe a tool for measuring and emulating the underwater acoustic channel which relies on the impulse response matrix (IRM) evaluation [Diamant 2005]. Using data obtained during a channel sounding experiment, the authors construct a matrix of sequential impulse responses to represent the time-varying impulse response of the channel, with the Mediterranean Sea used as a case study. Two-dimensional convolution is performed with a modulated waveform and the emulated IRM to produce a new signal that exhibits the emulated channel response (ECR). Transmission loss over the channel is evaluated by subtracting the measured the level at the receiver from the measured the source level of the transmitter. Noise, either generated with an empirical formula or recorded in the actual channel, can be added to the signal to emulate various SNRs. The authors found the ECR to a given signal to be highly correlated (typically greater than 80%) with the response of the actual channel.

Harris and Zorzi implemented a model [Harris 2007] for ns-2 [NS2 2010] that mainly relies on empirical formulas to describe the underwater acoustic channel. Their model is divided into the following four categories: propagation, channel, physical layer, and modulation. The propagation model calculates the SNR of a signal at the receiver after attenuation, ambient noise, and possible interference from other nodes are taken into account. Thorp's approximation for the absorption of a wave at a given frequency and formulas for the PSD of various noise-producing agents [Coates 1989] form the basis for calculating the range of transmission and SNR at the receiver. The channel model maintains lists of nodes to calculate neighbor sets and determine if packet collisions occur during transmissions. In addition, the channel model is responsible for calculating propagation delays. The physical layer model calculates the available bandwidth for the channel given the distance between the transmitter and receiver and relies upon the modulation model to calculate the effective bit rate and bit error rate, given the SNR and bandwidth used.

The Underwater Sensor Network Lab (UWSN) at the University of Connecticut has developed a simulator called Aqua-Sim [Aqua-Sim 2010] as an extension to ns-2. While still a work in progress, the software currently supports 3D and mobile networks, simulates underwater acoustic channels, and implements a complete protocol stack from the physical layer up to the application layer. UnderwaterChannel and UnderwaterPhy are the C++ classes most relevant to the other works in this section. In this simulator, these objects focus on power consumption and propagation range and delay for recreating packet collisions. There is no support for estimating bit error rates based on SNR or channel impulse response estimates. Based on the publications listed on the lab's web site as well as a code walk-through, the simulator's strong point is for evaluating protocols, especially those at the MAC and routing layers.

Another work in progress is the NS2 UAN Simulator being developed by the Fundamentals of Networking Laboratory (FuNLab) at the University of Washington's Department of Electrical Engineering [FuNLab 2010]. While not intended for distribution, the ns-2 version of the software is available for downloading. It features a FH-FSK PHY which attempts to closely model FH-FSK as implemented in the WHOI Micro-Modem, an improved propagation layer which uses BELLHOP output to compute signal attenuation and channel delay spread, and several new MAC layers including pure ALOHA, ALOHA with random backoff, and a reservation MAC with a dedicated control channel. The team, led by Leonard Tracy, is currently developing new modules for ns-3, none of which is presently available.

The World Ocean Simulation System (WOSS) also employs the BELLHOP model for propagation modeling [Guerra 2009]. This project interfaces with three separate databases to obtain the information necessary to run the BELLHOP model, which includes the sound velocity profile, the bathymetric profile, and the type of bottom sediments. The World Ocean Database [WOD 2010] which contains the constituent data for generating sound velocity profile for under-

water channels across the world. The bathymetric data have been taken from the General Bathymetric Chart of the Oceans [GEBCO 2010], a public database offering samples of the depth of the sea bottom with an angular spacing of 30 seconds of arc. Finally, the type of bottom sediments is taken from the National Geophysical Data Center's Deck41 database [NGDC 2010]. Noise power is computed through empirical formulas and is modeled as a white process within the frequency band of the modulated carrier. Upon obtaining a specification of the physical layer, the MIRACLE package [MIRACLE 2010] for ns-2 handles the remaining part of the simulation, namely the computation of Signal-to-Interference-plus-Noise Ratio (SINR) for all transmissions and the error rates corresponding to these SINR values.

Shin and Park implemented a simulation for the underwater environment in OMNeT++ [Shin 2008]. The propagation model is simply based on empirical formula, since the authors were more concerned with simulating various ack techniques. Finally, Nasri et al. proposed simulating the underwater channel using the hardware description language VHDL-AMS [Nasri 2009]. The one aspect of this work that differentiates it from the others is the simulation of a multipath Rayleigh fading channel via Jakes's model [Jakes 1975].

The simulation described in this section differs from the previous work in that it is fully based on channel measurements, providing the most accurate representation of the channel possible. The impulse response measurements give the delay spread and frequency response of the channel. The CTD measurements are used to calculate the propagation delay. Ambient noise is added to the signal to realistically estimate the SNR with which the signal is received. Unlike prior efforts, the simulation employs real software defined radio concepts. The transmitter and receiver manipulate sampled signals just as they would in a digital communication system, with hard limiters, filters, correlators, and comparators, making it easy to test how a given PHY layer implementation works in the channel. Furthermore, the simulation is written in a modular fashion

so that it can easily be extended with new processing blocks and any component can be replaced with an alternate implementation.

3.3 Simulation

Measurements form the basis of the simulation. CTD measurements yield the sound velocity profile of the channel, which is then used to calculate the propagation delay. Noise measurements are combined with a formula for the estimated transmission loss to approximate the SNR with which a packet is received. Data packets are converted into modulated waveforms and convolved with the impulse response of the channel, and then noise is added to produce a new signal that has the SNR previously computed by the simulation. At this point the new signal contains a reasonably close approximation to the distortion the channel would have imposed upon a real transmitted waveform. Finally, the distorted signal is demodulated and its BER is computed.

Figure 3-3 depicts the architecture of the OMNeT++ simulation. The application and link layers are implemented as objects within OMNeT++. While these layers are outside the scope of this project, they exist to form a more cohesive network stack whose functionality can be extended in future work. At the present time the application layer simply creates and receives messages. The transport layer, which typically handles reliable end-to-end delivery with sophisticated functions such as retransmission and flow control, has not been implemented. The link layer manages checksums. Before frames are sent to the PHY layer, the link layer computes the checksum and stores it inside the frame. Upon receiving an incoming frame from the PHY later, the link layer recomputes the checksum and compares it to the stored checksum. If the two values match, the message is passed up to the application layer. Otherwise, it assumes that the data is corrupt, and the frame is discarded. The PHY layer and underwater acoustic channel model is actually implemented in MATLAB and exported as a shared library with which the OMNeT++

simulation links. More details about the implementation of the PHY layer and channel appear in the subsequent sections.

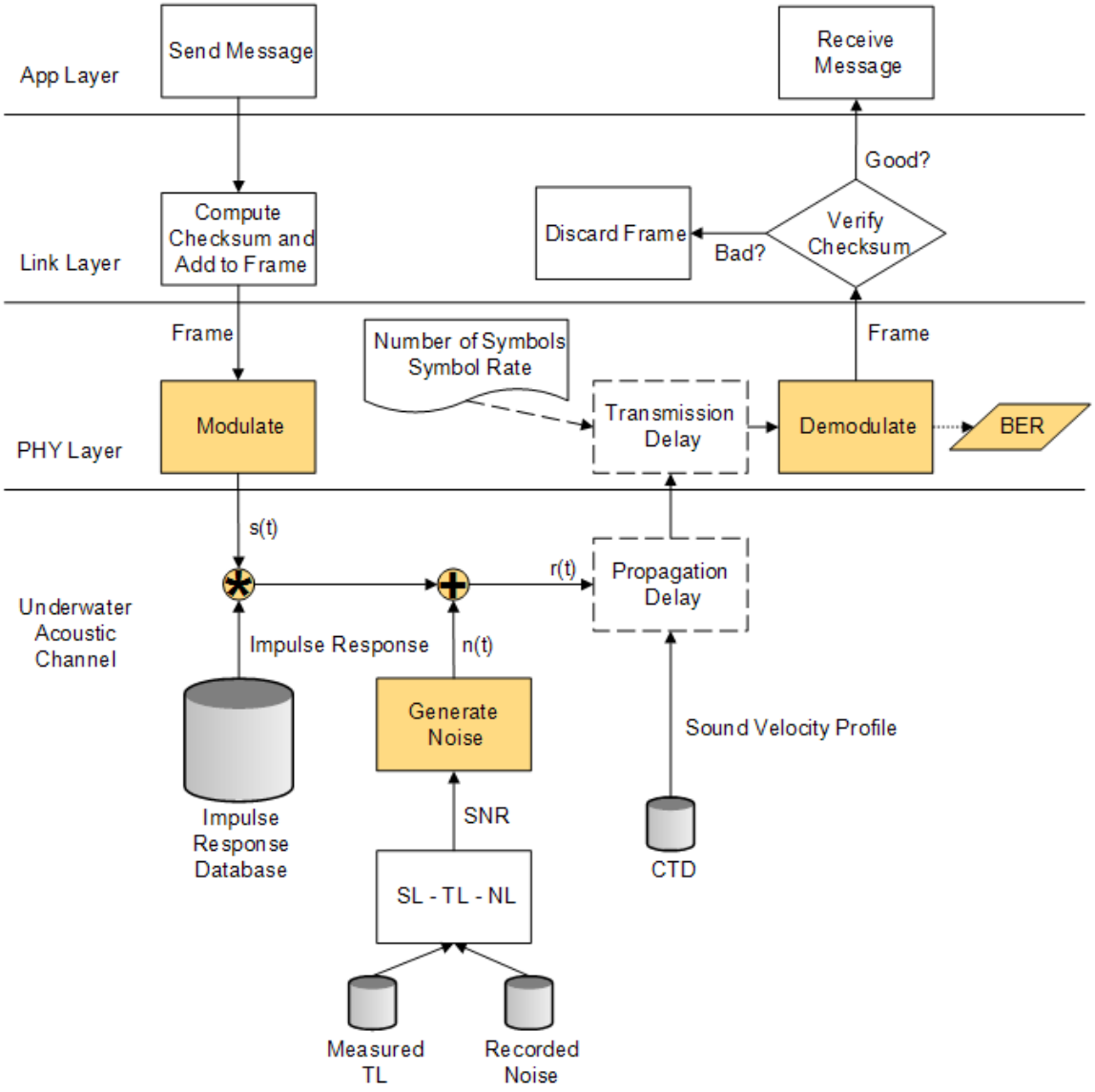

Figure 3-3: Architecture of OMNeT++ simulation for PHY layer and underwater acoustic channel. Areas in gray represent data from measurements. Areas in yellow are implemented in MATLAB. A simple implementation of the application and link layers is included for future extensions but is not the focus of this work.

3.3.1 Propagation and Transmission Delay

Propagation delay is the amount of time it takes for the beginning of a (possibly long) signal to travel from the transmitter to the receiver over a channel. In order to calculate it, the distance of the link and the propagation speed must be known. In this simulation the link distance is an adjustable user parameter with acceptable values of 200 or 505 meters, since those are the only two distances for which measurements exist. Note that simulator itself can model any distance, as long as the corresponding data is present. The speed of sound in water is often estimated to be 1500 m/s; however, this simulation aims to be more accurate. The sound velocity profile is computed using Equation (2.23). The velocity values obtained over the water column are averaged. This average rate is then used to calculate the approximate propagation delay with the standard *time* = *distance* / *rate* formula. Transmission delay is the amount of time it takes to send all of the packet's bits into the channel. It is given by the formula $D_T = N / R$, where D_T is the transmission delay, N is the number of symbols, and R is the rate of transmission in symbols per second.

3.3.2 Transmission Loss

Transmission loss, which describes the weakening intensity of a signal over a distance, was not measured during the August 2008 experiment in the Hudson River estuary. It was, however, measured in a previous experiment conducted in the same region of the Hudson by a group of researchers at Stevens [Roh 2008]. They estimated the attenuation coefficient α at 0.058 dB/m between 10 – 80 kHz, yielding the following formula for transmission loss:

$$TL = 10\log(r) + \alpha r = 10\log(r) + 0.058r, \tag{3.1}$$

where r is the distance over which the loss is being calculated. The simulation uses this equation to estimate transmission loss.

3.3.3 Noise

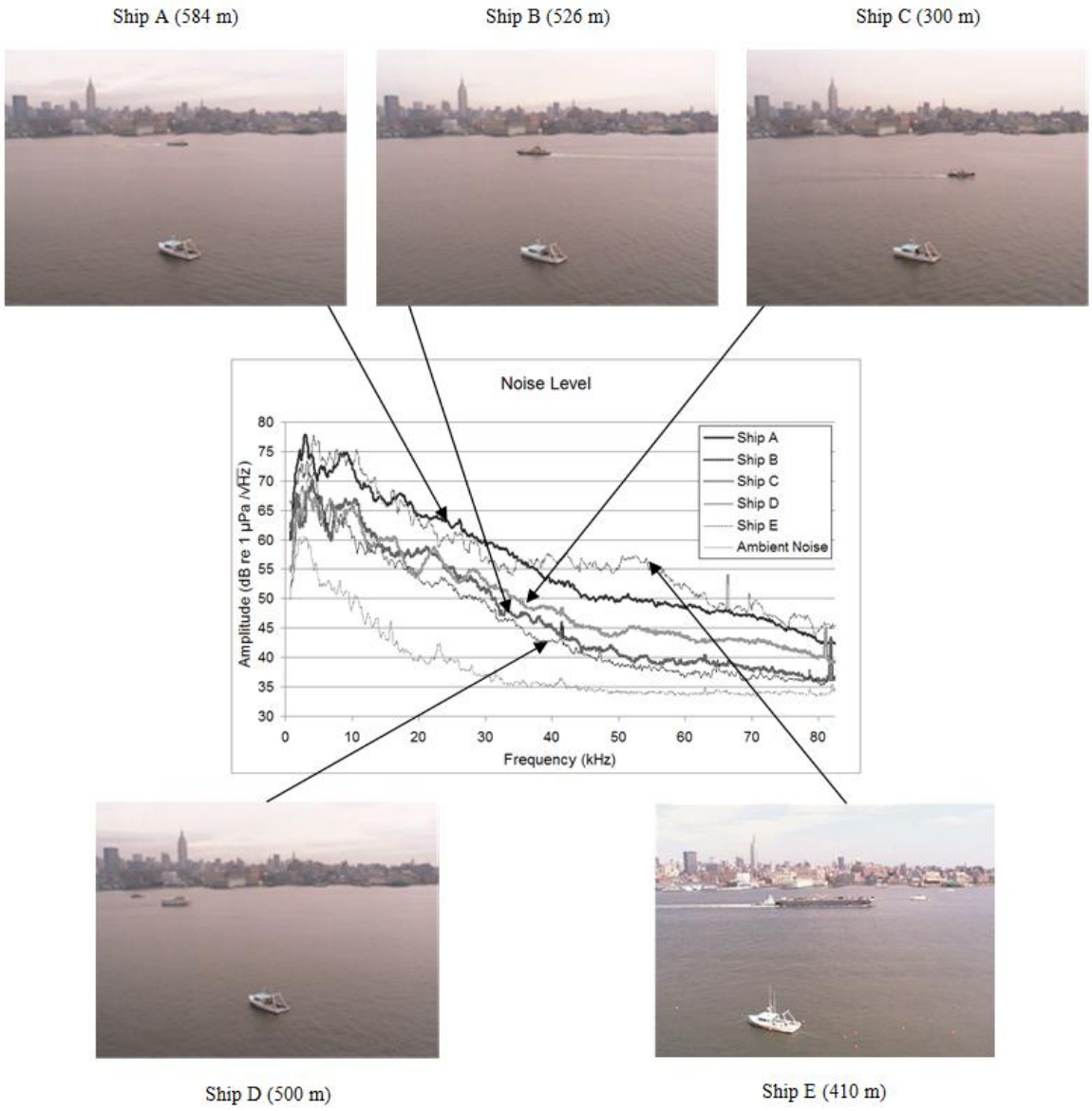

Figure 3-4: Noise levels in the Hudson River estuary produced by different passing ships.

Noise measurements from previous experiments were incorporated into the simulation [Borowski 2008]. Two categories of noise are modeled. The first is just ambient noise in the estuary, while the other is the noise level when various ships are present. Figure 3-4 shows the noise levels recorded in the Hudson River estuary under different conditions ranging from am-

bient noise to a small speed boat to a large tug and barge. The simulation chooses a noise level randomly for each packet that is sent through the channel.

The simulation determines the average noise level in the frequency band of the modulated signal. For instance, binary FSK signaling at 1000 symbols per second typically requires 2 kHz of bandwidth for good performance with a noncoherent receiver. Therefore, if the carrier frequency were 50 kHz, the simulation would find the average noise level in dB from 49 – 51 kHz. Since the values are given in dB, they must converted to linear scale, averaged, and then converted back to dB scale, as in

$$dB_{avg} = 10\log_{10}\left(\left(1/n\right)\sum_{1}^{n} 10^{(L/10)}\right), \tag{3.2}$$

where L is the level of a particular frequency component in dB.

3.3.4 Modulation

The simulation can generate binary FSK or PSK waveforms. With binary FSK, two tones are used. One tone represents the data bit '0' and the other represents '1'. Since the FSK receiver implements noncoherent demodulation, the tones must be separated by the number of Hz equal to the symbol rate. So, at 2000 bits per second, the tones must be separated by 2000 Hz. More formally, modulation index k is set to 1. The modulation index is defined by the formula

$$k = 2f_d/R, \tag{3.3}$$

where f_d is the frequency deviation in Hz (½ the separation between the two tones) and R is the data rate in symbols per second. As with other inexpensive noncoherent FSK systems, a modulation index of 1 is required to obtain reasonable receiver performance [Marrow 2002]. A specific type of FSK, called continuous-phase FSK or CPFSK, has been implemented as it is more bandwidth-efficient than FSK that breaks phase at the start of each symbol. Figure 3-5 shows the MATLAB code to generate a CPFSK signal.

```
%% Performs CPFSK modulation.
%  Pre: data is a vector of length numberOfBits of zeros and ones.
samplingRate = 200000;
symbolsPerSecond = 1000;
samplesPerBit = floor(samplingRate / symbolsPerSecond);
carrierFreq = 80000;
fc = [(carrierFreq - symbolsPerSecond / 2) (carrierFreq + symbolsPerSecond /
2)];
txFSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    for j = 1:samplesPerBit
        t = t + 2 * pi * fc(data(i)+1) / samplingRate;
        if (t > pi)
            t = t - 2 * pi;
        end
        txFSK(j + samplesPerBit*(i-1)) = cos(t);
    end
end
```

Figure 3-5: MATLAB code to generate a CPFSK waveform.

While FSK changes the frequency of the carrier signal, PSK changes the signal's phase. With BPSK, the signal representing a '0' is 180° out of phase with the signal representing a '1'. The MATLAB code to generate a BPSK signal, shown in Figure 3-6, is quite similar in structure to the code for FSK. Note that *t* is reset to 0 before generating the next symbol. This step is necessary so that any carrier frequency can be used and successfully demodulated by a simple correlation receiver. If this step were not present, the symbol rate would need to evenly divide the carrier frequency, which would need to evenly divide the sampling rate so that there would be an even number of periods of the sinusoid in each symbol and that the phase transitions would occur only 180° apart at $y = 0$.

```
%% Performs PSK modulation.
%  Pre: data is a vector of length numberOfBits of zeros and ones.
samplingRate = 200000;
symbolsPerSecond = 1000;
samplesPerBit = floor(samplingRate / symbolsPerSecond);
carrierFreq = 80000;
txPSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    offset = samplesPerBit * (i-1);
    for j = 1:samplesPerBit
        t = t + 2 * pi * carrierFreq / samplingRate;
        if (t > pi)
```

```
            t = t - 2 * pi;
        end
        if (data(i) == 0)
            txPSK(j + offset) = -cos(t);
        else
            txPSK(j + offset) = cos(t);
        end
    end
    t = 0.0;
end
```

Figure 3-6: MATLAB code to generate a PSK waveform.

3.3.5 Channel Emulation

The impulse response estimates of the Hudson River estuary from Section 2.5.4 form the kernel of the simulation. The database seen in Figure 3-3 is implemented as two directories of wav files, one pertaining to the measurements at 200 meters and the other at 505 meters. The code in Figure 3-7 shows how the impulse estimates have been extracted. There are two noteworthy aspects of this code segment. The first is that each impulse response must be "cropped" to contain only significant multipath components. Any value that is within 6 dB (0.25) of the strongest arrival's intensity must be included in the estimate, as it contains relevant information about the delay spread and frequency response of the channel at that moment. In a personal conversation with Jim Preisig at WUWNet'09 [Preisig 2009], he stated that values within 4 dB of the strongest intensity are significant, though he agreed that choosing a -6 dB cutoff would also be reasonable. The second interesting point is that the impulse responses can be normalized only after all of them have been extracted. This way, the amplitudes of the estimates obtained during a deep fade are not artificially increased to equal those of estimates obtained during periods with high SNR. While this feature is not exploited in the current simulation, it is beneficial to create the database with measurements that are as accurate as possible for future extensions.

```
%% Extracts impulse estimates to individual wav files.
seconds = 0.005;
len = seconds * samplingRate;
impulseResponse = zeros(numOfImpulseResponses, len);
```

```
for i = 1:numOfImpulseResponses
    snip = recordedSignal((i-1)*referenceSamples+1:i*referenceSamples);
    temp = fftshift(real(xcorr(snip, conj(referenceSignal))));
    [maxValue maxIndex] = max(temp);
    earlyPeakIndex = ...
        find(temp(max(maxIndex - 0.0005*samplingRate, 1):end) > 0.25*maxValue);
    [mainPeak mainPeakIndex] = ...
         max(temp(max(maxIndex - 0.0005*samplingRate, 1):end));
    offset = mainPeakIndex - earlyPeakIndex;
    temp = temp(maxIndex - offset:maxIndex - offset + len);
    impulseResponse(i,:) = temp(1:len);
end
[maxVal maxIndex] = max(max(abs(impulseResponse)));
impulseResponse = 0.98 * impulseResponse / maxVal;
for i = 1:numOfImpulseResponses
    filename = sprintf('IR_200m/IR_%d', i);
    wavwrite(impulseResponse(i,:), samplingRate, filename);
end
```

Figure 3-7: MATLAB code to extract impulse estimates to individual wav files.

When a frame is passed through the simulated channel, it is convolved with one randomly chosen impulse response estimate. As the lengths of the input vectors grow, the execution time of convolution performed in the time domain grows quadratically. Therefore, to improve the performance of the simulation for long data frames and/or delay spreads, FFT convolution is implemented. Convolution in the time domain corresponds to multiplication in the frequency domain. Thus, in FFT convolution the input signal is transformed into the frequency domain using FFT, multiplied by the frequency response of the filter, and then transformed back into the time domain using the inverse FFT [Smith 2003]. The complexity of FFT convolution is $O(n \log n)$, which is a huge improvement over $O(n^2)$.

After the convolution routine is finished executing, noise is added to the signal so that the resulting SNR = SL – TL – NL, where SL is the user-specified source level, TL is the transmission loss described in Section 3.3.2, and NL is a randomly chosen noise level described in Section 3.3.3. The noise added to the signal is AWGN. This is a reasonable simplification, since the simulation is concerned only with noise in the frequency band of the data transmission, which occupies a small amount of space on the acoustic spectrum.

3.3.6 Demodulation

The simulator can demodulate PSK and FSK signals. A correlation receiver is used to demodulate PSK waveforms. It works by dividing the incoming stream into small sections with the duration of an individual symbol. Each section is multiplied by the reference signals of the same length that represent the symbols '0' and '1'. The results are then summed over the symbol duration, and finally the comparator chooses the symbol whose sum was the greatest. Figure 3-8 shows the block diagram for a correlation receiver with *M* reference signals. Note that in the case of BPSK waveforms, only 2 reference signals are required. Figure 3-9 shows how to translate the block diagram into MATLAB code.

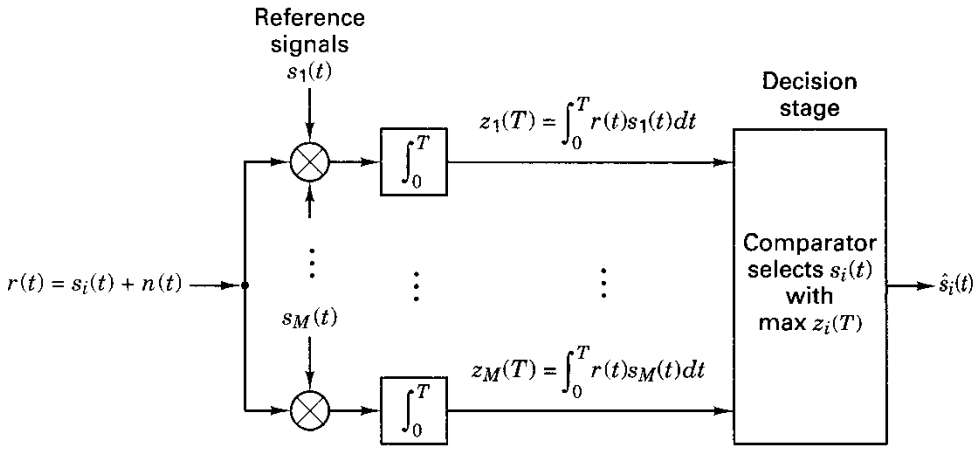

Figure 3-8: Correlation receiver with *M* reference signals {$s_i(t)$} [Sklar 2001].

```
%% Demodulates PSK signal using a correlation receiver.
t = 0:samplesPerBit-1;
psk0 = -cos(2 * pi * carrierFreq/samplingRate * t);
psk1 =  cos(2 * pi * carrierFreq/samplingRate * t);
rxPSK = zeros(1, numberOfBits);
for i = 1:numberOfBits
    rcv = packet((i-1)*samplesPerBit + 1:i*samplesPerBit);
    zero = rcv .* psk0;
    one = rcv .* psk1;
    z0 = sum(zero);
    z1 = sum(one);
    rxPSK(i) = (z1 > z0);
end
```

Figure 3-9: MATLAB code implementing a correlation receiver for PSK signals.

While the PSK waveform that was transmitted originally contained ’0’ and ‘1’ reference signals of $-\cos(2\pi ft)$ and $\cos(2\pi ft)$, where f is the carrier frequency and t is time, the phase of received waveform may be distorted so that transitions no longer occur at 0° and 180°. For instance, perhaps they now occur at 25° and 205°. In order to improve the performance of the correlation receiver, it would be necessary to estimate these offsets. A simple solution, the one implemented in the simulator, is to send training symbols of alternating ones and zeros at the beginning of the signal. The receiver can first try various offsets to find the one that maximizes the correlation of the known symbols. Since the model currently convolves a single impulse response with the data signal, this type of synchronization is adequate. However, when communicating through and modeling real time-variant channels, more sophisticated methods are necessary. A receiver which jointly performs carrier synchronization and fractionally spaced decision feedback equalization of the received signal and whose parameters are adaptively adjusted using a combination of the RLS algorithm and second-order digital PLL was shown to perform well in short-range shallow water of 2 nautical miles at 10 kbps [Stojanovic 1994]. Since the simulator is written in modular fashion, one can write the code for this receiver in MATLAB and incorporate it into model with minimal effort.

Noncoherent detection of FSK waveforms is implemented via both the quadrature receiver and bandpass filters/envelope detectors. Each receiver is also able to employ a hard limiter [Jones 1963] as the first stage of demodulation process. This procedure places a cap on high-amplitude samples and raises low-amplitude samples to the value of the cap. Utilization of a hard limiter helps equalize the signal so that each tone in the FSK signal has the same amplitude before it is passed to subsequent stages of the receiver. This is especially useful when working with transducers that do not have a flat frequency response. Two correlators are used for each tone, one for the in-phase (I) and quadrature (Q) channels. Since measurements of the signal’s phase

cannot be exploited, the receiver is merely an energy detector. By summing the squares of the correlation on each channel, the same values will be fed to the decision stage if either channel had full correlation while the other had none or if the incoming signal partially correlated with both references. Figure 3-10 shows the block diagram for a quadrature receiver; Figure 3-11 shows how to translate the block diagram into MATLAB code. The implementation of *hardlimit* is found in Section B.2 of the appendix.

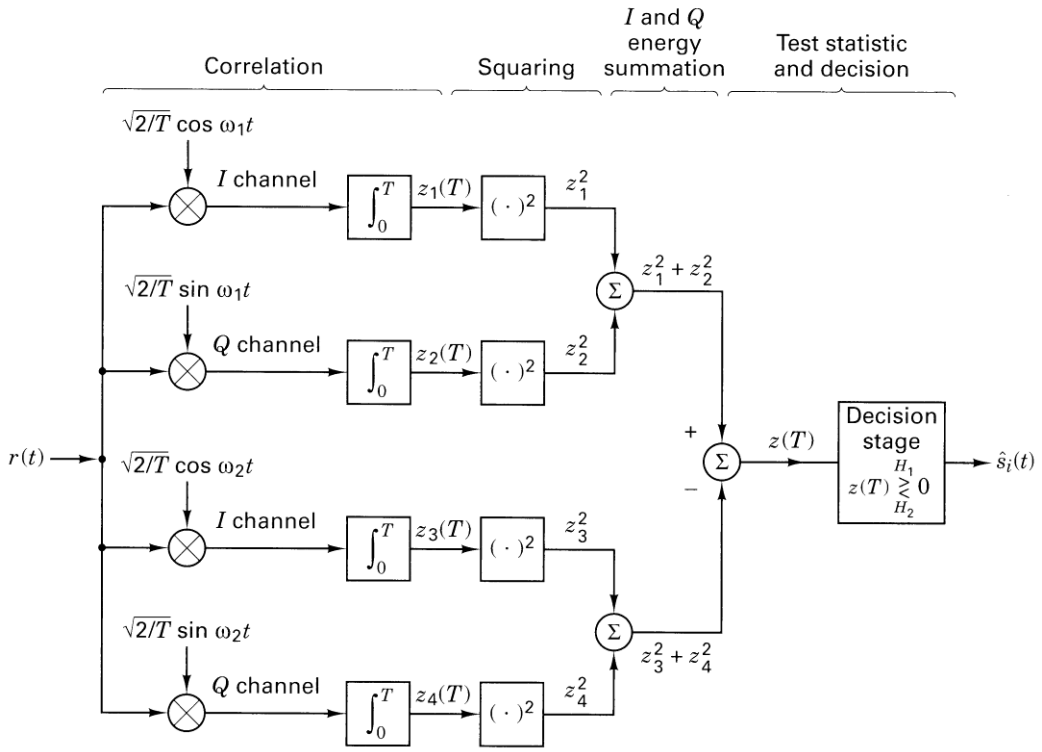

Figure 3-10: Quadrature receiver for noncoherent detection of FSK signals [Sklar 2001].

```
%% Demodulates FSK signal using a quadrature receiver.
fc = [(carrierFreq - symbolsPerSecond/2) (carrierFreq + symbolsPerSecond/2)];
t = 0:samplesPerBit-1;
cos0 = cos(2 * pi * fc(1)/samplingRate * t);
sin0 = sin(2 * pi * fc(1)/samplingRate * t);
cos1 = cos(2 * pi * fc(2)/samplingRate * t);
sin1 = sin(2 * pi * fc(2)/samplingRate * t);
rxFSK_q = zeros(1, numberOfBits);
for i = 1:numberOfBits
    if (useLimiter)
        rcv = hardlimit(packet((i-1)*samplesPerBit + 1:i*samplesPerBit));
```

```
    else
        rcv = packet((i-1)*samplesPerBit + 1:i*samplesPerBit);
    end
    Izero = rcv .* cos0;
    Qzero = rcv .* sin0;
    Ione = rcv .* cos1;
    Qone = rcv .* sin1;
    z1 = (sum(Izero))^2;
    z2 = (sum(Qzero))^2;
    z3 = (sum(Ione))^2;
    z4 = (sum(Qone))^2;
    energy0 = z1 + z2;
    energy1 = z3 + z4;
    rxFSK_q(i) = (energy1 > energy0);
end
```

Figure 3-11: MATLAB code implementing a quadrature receiver for FSK signals.

The simulation also implements noncoherent FSK detection with bandpass filters followed by envelope detectors, as shown in Figure 3-12. When using binary FSK, the incoming signal is passed through two separate filters (M = 2 in Figure 3-12) to eliminate signals outside the band used by each tone. Second-order IIR filters based on the design in [Smith 2003] are used. For a second-order filter, the best performance is obtained when the product BT is close to 1.0, where B is the -3dB bandwidth in Hz and T is the duration of a symbol [WJC 1980]. Therefore, the filters have been designed to operate with bandwidth $1/T$ centered on the tones representing 0s and 1s.

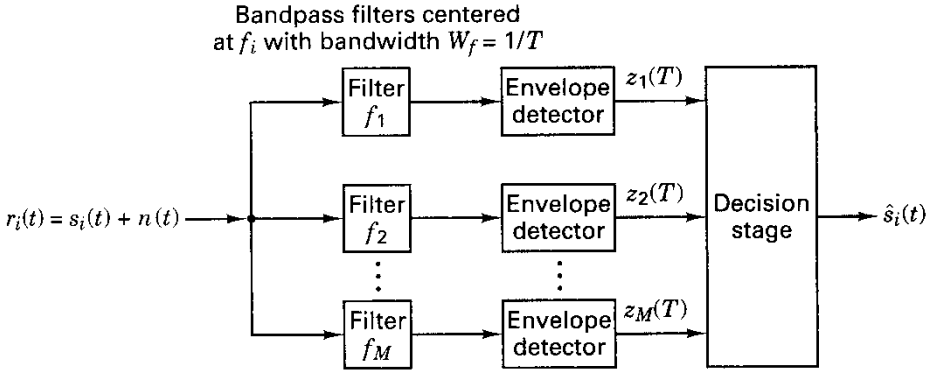

Figure 3-12: Noncoherent detection of FSK signals using bandpass filters and envelope detectors [Sklar 2001].

Upon completion of the filtering stage, the resulting signals are passed to the envelope detectors, each of which first applies the Hilbert transform to obtain the imaginary part x_i of a signal containing only real values x_r. The envelope is then obtained by taking the magnitude of the signal, as shown by Equation (2.22). Finally, for each symbol the envelopes are summed over several samples, and the symbol corresponding to the greatest sum is outputted by the decision stage. Figure 3-13 shows how to translate the block diagram in Figure 3-12 to MATLAB code. The implementation of *filterSignal* is found in Section B.3 of the appendix.

```
%% Demodulates FSK signal using a bandpass filters and envelope detectors.
fc = [(carrierFreq - symbolsPerSecond/2) (carrierFreq + symbolsPerSecond/2)];
if (useLimiter)
    zeroSignal = abs(hilbert(filterSignal(hardlimit(packet), samplingRate, ...
                             fc(1), symbolsPerSecond)));
    oneSignal = abs(hilbert(filterSignal(hardlimit(packet), samplingRate, ...
                             fc(2), symbolsPerSecond)));
else
    zeroSignal = abs(hilbert(filterSignal(packet, samplingRate, fc(1), ...
                             symbolsPerSecond)));
    oneSignal = abs(hilbert(filterSignal(packet, samplingRate, fc(2), ...
                            symbolsPerSecond)));
end
diff = oneSignal - zeroSignal;

rxFSK_e = zeros(1, numberOfBits);
for i = 1:numberOfBits
    % Sample in second half of symbol to avoid ring in IIR filter.
    rcv = sum(diff((i-1)*samplesPerBit + ...
                   floor(samplesPerBit/2):i*samplesPerBit));
    rxFSK_e(i) = (rcv > 0);
end
```

Figure 3-13: MATLAB code implementing a receiver for FSK signals with bandpass filters and envelope detectors.

3.4 Emulator Validation

In order to validate the design of the measurement-based channel emulator, several experiments were conducted in the office test environment. The premise is that a frame that is transmitted through the channel will have the same number of bit errors as a clean frame waveform that is convolved with the impulse response of the channel. The office test environment was used because it is easily accessible and offers a simplistic time-invariant channel. The simulation is,

indeed, expected to be less accurate for the Hudson River estuary because of its complex time-variant nature and severe fading. However, another field experiment in the Hudson was not possible due to lack of funding.

3.4.1 Procedure

Thirty composite test signals were generated and transmitted through the tub. Binary FSK and PSK signals were transmitted at five different bit rates to cover cases with and without channel-induced ISI. Communication was tested in three frequency bands – 7.5 kHz, 12.5 kHz, and 17.5 kHz – that evenly divided and collectively spanned the usable channel bandwidth. The sampling rate for all signals was 48 kHz.

Each composite signal starts with a 5-second LFM chirp over the entire channel bandwidth (0 – 24 kHz) that is used to estimate the system's impulse response at the beginning of that particular test. The time-invariant office test environment provides the opportunity to use a longer chirp signal than was used in the Hudson experiment, resulting in the best autocorrelation, and hence, most accurate impulse response estimate. Modulated waveforms appear after the long chirp signal. Fifty duplicate packets, each beginning with a short chirp signal for synchronization purposes and separated from the next by 1 second of silence, were transmitted in all tests except for those running at only 250 bps, where only twenty duplicate packets were transmitted. Table 3-1 summarizes the bit rates tested at each frequency. Only bit rates that even divided the carrier frequency were used in this experiment. Figures 3-14 and 3-15 show the time and frequency domain views of the recorded chirp signal and FSK-modulated packets during one of the fifteen tests.

Table 3-1: Bit rates tested at each carrier frequency in the office tub.

Frequency (Hz)	Bit Rate (bps)				
7,500	250	500	1250	2500	3750
12,500	250	500	1250	2500	3125
17,500	250	500	1250	2500	3500

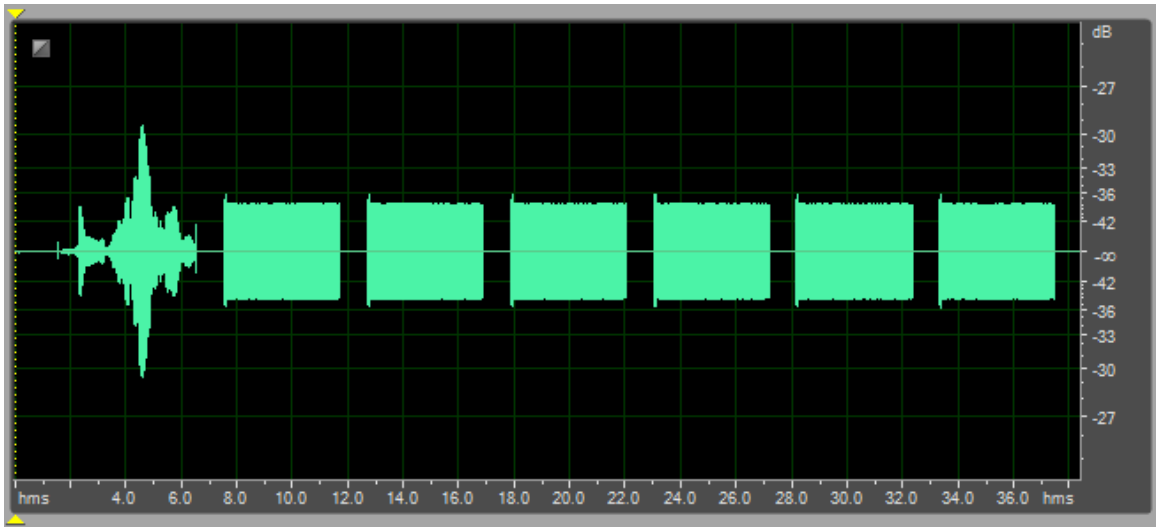

Figure 3-14: Time domain view of recorded 5-second LFM chirp signal followed by FSK-modulated packets at 500 bps with a 12.5 kHz carrier.

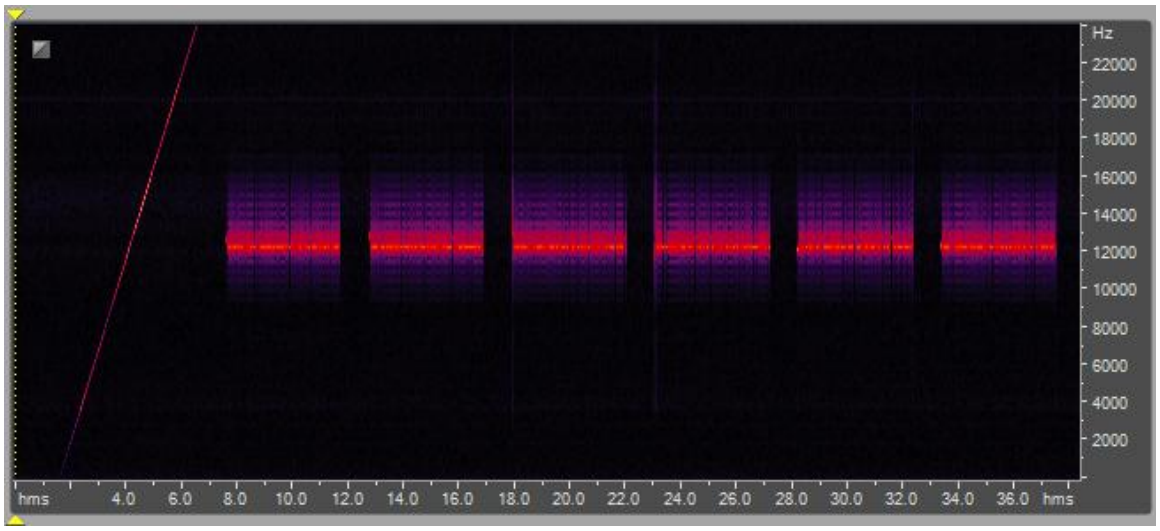

Figure 3-15: Frequency domain view of Figure 3-14 (recorded 5-second LFM chirp signal followed by FSK-modulated packets at 500 bps with a 12.5 kHz carrier).

For each of the fifteen trials in this experiment, the recorded signal of modulated waveforms was processed in MATLAB. The series of FSK-modulated packets was demodulated with the envelope detector and quadrature receiver, both with and without the hard limiter, as described in Section 3.3.6. Detection of PSK packets was accomplished via the correlation receiver, also described in Section 3.3.6. The BER of each packet was computed, and then the average BER for that trial was calculated.

The simulated BER was obtained by first estimating the channel's impulse response. As in Section 2.2.2, the impulse response was obtained by cross-correlating the received signal with the complex conjugate of the reference signal, which in this case is the 5-second LFM chirp signal at start of each trial. The impulse response was then convolved with the signal containing the waveform of a single modulated packet. Note that only one packet is needed here, since only one impulse response estimate was obtained at the start of the test. In theory, because the office tub is a time-invariant environment, the estimated impulse response should remain nearly constant over the duration of a trial.

3.4.2 Analysis

Appendix C lists the results of each of the fifteen trials in this experiment. Tables 3-2 through 3-5 summarize the comparison of BERs obtained with data transmission versus convolution with the channel's impulse response. The percent difference is defined merely as the difference between the two BERs. The standard formula for percent error cannot be applied, since it requires the measured and expected values to be positive (> 0). In general, the average difference is quite good, varying only approximately 3.34%. There are several cases where the difference is 0. These instances appear mostly at the slower bit rates, where the symbol duration easily exceeds the channel's delay spread. There are also a few cases of large discrepancies of greater than 20% appearing in seemingly random positions in the tables of Appendix C.

Table 3-2: Overall comparison of BERs obtained with data transmission versus convolution.

Overall % Difference			
Average	Min	Max	Std. Dev.
3.34	0.00	33.31	8.21

Table 3-3: Comparison of BERs obtained with data transmission versus convolution, per carrier frequency.

	% Difference			
Frequency (Hz)	Average	Min	Max	Std. Dev.
7,500	1.74	0.00	12.47	3.35
12,500	3.66	0.00	33.31	8.16
17,500	4.61	0.00	27.69	8.11

Table 3-4: Comparison of BERs obtained with data transmission versus convolution, grouped by the type of modulation-demodulation.

			% Difference			
Modulation	Demodulation	Implementation	Average	Min	Max	Std. Dev.
FSK	Envelope Detector	Amplitude Comp.	2.60	0.00	20.16	5.22
		Hard Limiter	3.34	0.00	27.69	7.28
	Quadrature Receiver	Default	4.35	0.00	24.41	7.36
		Hard Limiter	2.89	0.00	24.31	6.14
PSK	Correlator	N/A	3.51	0.00	33.31	9.01

Table 3-5: Comparison of BERs obtained with data transmission versus convolution, grouped by bit rate.

	% Difference			
Bit Rate (bps)	Average	Min	Max	Std. Dev.
250	0.00	0.00	0.00	0.00
500	2.03	0.00	24.41	6.29
1250	7.50	0.00	27.69	9.72
2500	3.91	0.00	13.25	4.21
>3000	3.25	0.00	33.31	8.41

The amount of error increases with the carrier frequency. Since there should be no correlation between frequency and error, it becomes necessary to investigate the channel more closely. The frequency response of the impulse response estimate obtained at the start of each test can be computed using an FFT algorithm, which is accomplished easily with MATLAB's *freqz* function. Figure 3-16 shows the impulse response obtained during one of the tests. Figure 3-17 shows the frequency and phase response of the system derived from the impulse response in Figure 3-16. Over the course of the entire experiment, the impulse response and corresponding frequency and phase responses remained nearly constant. Upon viewing Figure 3-17 it makes sense why there are more errors at 12.5 kHz and 17.5 kHz than at 7.5 kHz. There are dips of more than 20 dB in the frequency response centered on 12.7 kHz and 17.5 kHz, which naturally give rise to drastic changes in the phase response at those frequencies. While the impulse response estimates do capture these properties, it is clear that they are not perfectly accurate, as the simulated bit error rates deviate from the measured values.

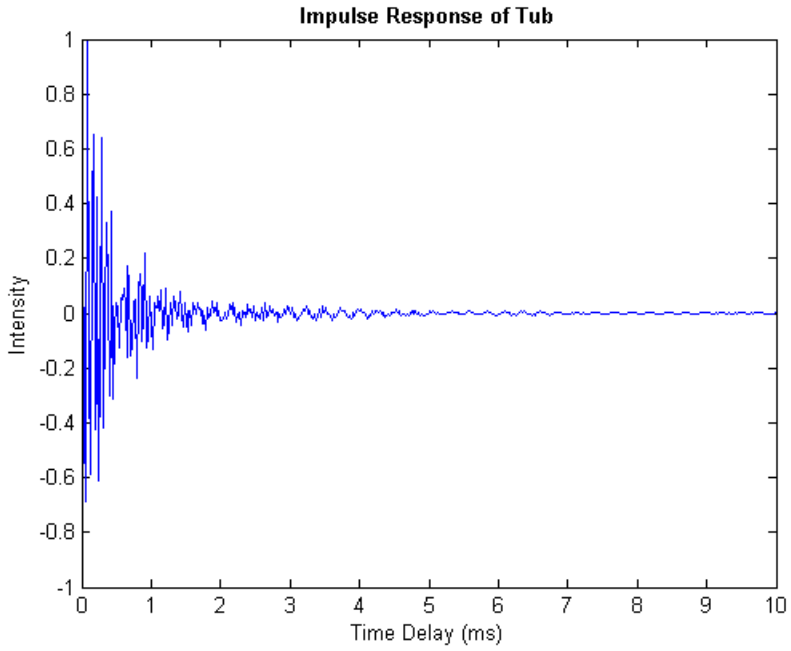

Figure 3-16: Impulse response of office test tub during validation experiment.

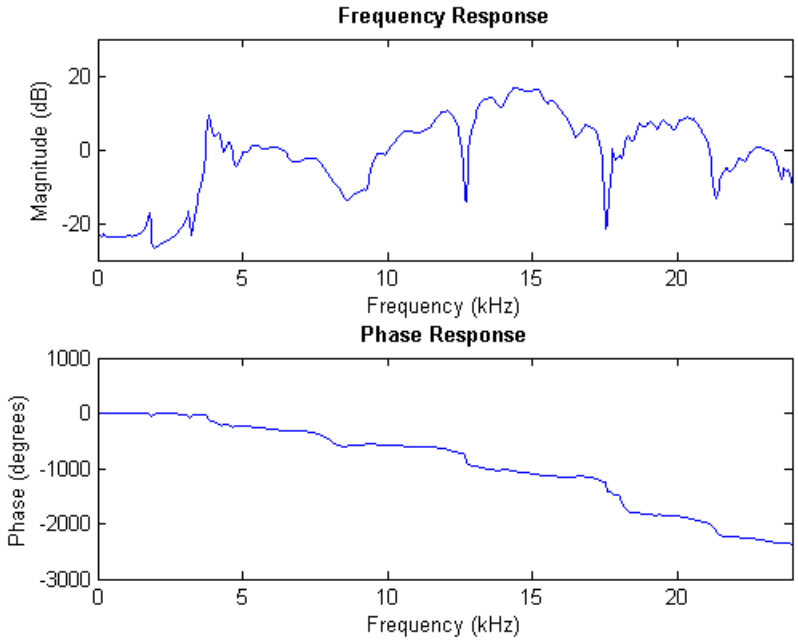

Figure 3-17: Frequency and phase response of office test tub, derived from impulse response in Figure 3-16.

There is no obvious conclusion to draw about the discrepancies between the simulated and measured values grouped by the modulation-demodulation routine. The average difference is within a few percent for all tests, though PSK demodulation does possess the worst-case value of the experiment. There is, however, a trend seen with increasing bit rate. The simulator is quite accurate for slow data rates, where the symbol duration exceeds the delay spread of the channel. At 250 bps and 500 bps, except for the FSK trial at 12.5 kHz bps with the default quadrature receiver, the simulated and measured bit error rates are identical. For this particular case, it seems that the simulator exaggerates the dip in the frequency response at 12.7 kHz, since it yields a BER of about 24% while the BER for real packets was actually 0%. When using the impulse response, the tone representing a '1' is so much lower in amplitude than the tone representing a '0' that incorrect correlation produces a bit error in favor of '0' 50% of the time a '1' should be present, resulting in the simulated BER of 24%. Note that the particular sequence of bits used in this experiment contains a 50.4/49.6 mixture of zeros and ones, respectively. As an aside, this one case

demonstrates the value of the hard limiter for FSK-based receivers. At higher bit rates, the simulator's accuracy decreases. It produces the worst results at 1250 bps, when the symbol duration is roughly equal to the delay spread of the channel.

In conclusion, using impulse response measurements to simulate a time-invariant channel is very accurate when the symbol duration exceeds the multipath spread of the channel and no ISI exists. The accuracy drops slightly when the symbol time is less than the channel's delay spread. This method, however, becomes error-prone when the symbol duration is very close to the length of the delay spread, since a small discrepancy between the estimated and actual channel responses can lead to very different results at the receiver. The simulator seems to perform well for both FSK and PSK signals transmitted at various carrier frequencies, though it works best when the system has linear frequency and phase response. Unfortunately, there is no data describing the simulator's accuracy for time-variant channels such as the Hudson River estuary. The model might be too simplistic to generate accurate BERs without the inclusion of varying impulse response estimates and fading. More details about future plans are found in Section 3.7.

3.5 Implementation

OMNeT++ 4.0 was chosen as the platform for this simulation because of its clean architecture and relative ease of development. Other platforms were considered and eliminated because of significant shortcomings. OPNET is perhaps the largest commercial network simulator [OPNET 2010]. While this software seems to do everything one could imagine, it is not easy to develop, debug, or maintain OPNET applications. In fact, after taking a 3-day course at their headquarters, one leaves with only a superficial understanding of where controls are located in the GUI and how to produce some performance metrics for a simple network topology. Worse, OPNET has numerous parameters, and it is not obvious which ones should be changed to model certain behaviors. Additionally, OPNET results are hard to understand because details about

what is going on under the hood are concealed. ns-2 is popular in the academic world, but has been in a state of transition for several years since the introduction of ns-3. While development of both platforms is ongoing, ns-3 is not backwards compatible with ns-2, nor does it have all the models that ns-2 currently has. In addition, it is not uncommon to see disorganized class files with large blocks of code commented out in ns-2. (For instance, look at mac/channel.cc in ns-allinone-2.34.) Therefore, like OPNET, ns leaves one feeling uneasy about the clarity, accuracy, and interactions among the underlying algorithms that produce simulation results. Finally, the architecture is not conducive to building a channel and PHY layer simulation. Currently, the channel object tied in with MAC layer objects, which violates conventional software engineering paradigms that promote creating independent, reusable blocks of code. Should PHY also be squeezed into the same location, as it was in Aqua-Sim, since it logically fits between the channel and MAC layer? While doing so is easier than restructuring the code, it perpetuates bad practice which will render creating future enhancements even more difficult.

OMNeT++, on the other hand, had just been upgraded to version 4.0 in February 2009 and provides a simple solution to the problem at hand. Unlike the other platforms, OMNeT++ is only a discrete event simulator. It does not include default network algorithms that the user must override, modify, or parameterize in unobvious ways. It offers a clean framework that allows developers to create network models as they see fit. Since the model described in this chapter relies on MATLAB for the computationally heavy signal processing routines, a simple framework for the channel and various network layers is all that is needed. Thus, development moved forward with OMNeT++ and was integrated with MATLAB for the modulation, convolution, and demodulation routines.

While it is beyond the scope of this dissertation to explain every detail about the implementation of the simulator, it is worth mentioning the configuration and the means of integrating

OMNeT++ and MATLAB. Note that the following description applies to the Linux operating system. The simulation can also run on Windows, but the minor variations in the process are not described here. Ubuntu Linux 9.10 x64, MATLAB R2009b, and OMNeT++ 4.0 were used. The simplest way to explain this material is to walk through an example. As previously mentioned, all signal processing routines are implemented in MATLAB, including the simple function for calculating transmission loss across the channel, as described in Section 3.3.2. Figure 3-18 shows the body of the function, which is saved in the M-file *getTransmissionLossDB.m*.

```
function transmissionLossDB = getTransmissionLossDB(linkDistanceInMeters)
    transmissionLossDB = 10 * log10(linkDistanceInMeters) + ...
                         0.058 * linkDistanceInMeters;
```

Figure 3-18: MATLAB code to calculate the transmission loss over the acoustic link.

This function and several others are grouped together into one library *libchannel* for the underwater channel, while a separate group of files comprises *libphy*, the library responsible for tasks associated with the PHY layer of a network stack. Continuing with the transmission loss example, the MATLAB compiler mcc generates the *libchannel.so* library. It is convenient to write a short shell script for MATLAB build process, as in Figure 3-19. For clarity when specifying library paths in OMNeT++, the shared objects are copied into the *lib* directory, which is located within the main directory of the simulation.

```
#!/bin/sh

rm -f *~

mcc -B csharedlib:libchannel -v getAverageSoundVelocity.m getNoiseLevelDB.m
getTransmissionLossDB.m simulateChannel.m fconv.m

mcc -B csharedlib:libphy -v modulateFSK.m demodulateFSK.m modulatePSK.m
demodulatePSK.m chirp.m filterSignal.m hardlimit.m

cp libchannel.so libphy.so ../lib/
```

Figure 3-19: Bash shell script for building MATLAB shared libraries used in the simulator.

After compiling the libraries, the next step is to integrate the MATLAB functions with the OMNeT++ simulation. Since the initialization functions must be called once and only once, the calls cannot be placed inside the constructor for the UWChannel or Phy objects, since every time an object is created the initialization routine will run. The following solution circumvents this problem:

1) Create a main.cc file for the simulation. The contents of the file can be identical to that of *path-to-omnetpp-4.0/src/envir/main.cc*, as long as the include path contains all the directories required for compilation.
2) Put the initialization code before the call to *setupUserInterface(argc, argv, NULL)* and termination code after it.

Appendix B.6 lists the code for the main function of the simulation which includes calls to initialization and termination routines.

After the libraries are initialized, the exported MATLAB functions can be called from OMNeT++. Input and output parameters are of type *mxArray*. Even a simple data type must be converted to an array of one element. Figure 3-20 shows how the function signature for *transmissionLossDB* from Figure 3-18 looks when compiled into a C shared library:

```
extern LIB_libchannel_C_API bool MW_CALL_CONV
mlfGetTransmissionLossDB(int nargout, mxArray** transmissionLossDB,
                         mxArray* linkDistanceInMeters);
```

Figure 3-20: Function signature for mlfGetTransmissionLossDB in C shared library.

When calling *mlfGetTransmissionLossDB*, double *linkDistanceInMeters* is first converted to a scalar, double-precision array *x1_ptr*. The output of the function is stored in the reference *y1_ptr*, and the number of output arguments is set to *1*. The output is then converted back to an array of doubles via *mxGetPr*, of which the first and only element is accessed with the array index [0]. Finally, the memory is deallocated with calls to *mxDestroyArray*, and the pointers are set to

NULL. Figure 3-21 shows the C code to obtain the value of transmission loss from the MATLAB library.

```
mxArray *x1_ptr;
mxArray *y1_ptr = NULL;
double linkDistanceInMeters = 505;
double transmissionLossDB;

x1_ptr = mxCreateDoubleScalar(linkDistanceInMeters);

// Call the implementation function for transmission loss
mlfGetTransmissionLossDB(1, &y1_ptr, x1_ptr);
transmissionLossDB = mxGetPr(y1_ptr)[0];
printf("Transmission loss      : %.2f dB\n", transmissionLossDB);

mxDestroyArray(x1_ptr); x1_ptr = NULL;
mxDestroyArray(y1_ptr); y1_ptr = NULL;
```

Figure 3-21: C code that calls the MATLAB library to obtain the value of transmission loss over a 505-m acoustic link.

3.6 Simulation Output

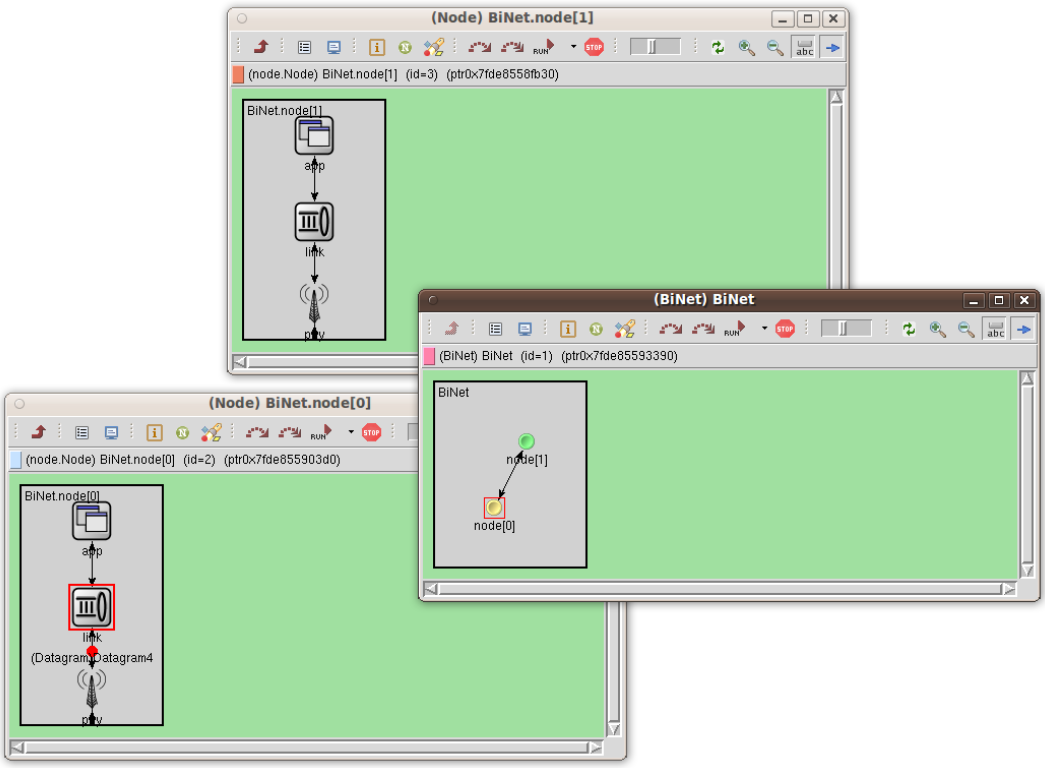

Figure 3-22: Graphical representation of OMNeT++ simulation.

The OMNeT++ simulation has a graphical representation of the network to visualize how packets move among the layers of the network stack, nodes, and channel. When the simulation is active, a red dot moves along the path from source to destination. Figure 3-22 shows the graphical representation of the OMNeT++ simulation. It contains two nodes and tests communication from the designated transmitter node[0] to the receiver node[1]. The red dot in the node[0] window indicates that the packet is passing from the link layer down to the PHY layer of the network stack.

Figure 3-23 depicts the OMNeT++ Tk environment. The majority of the window is occupied by a text area which displays the simulation's log. This log contains information about the initialization process as well as the creation, departure, and arrival times of packets.

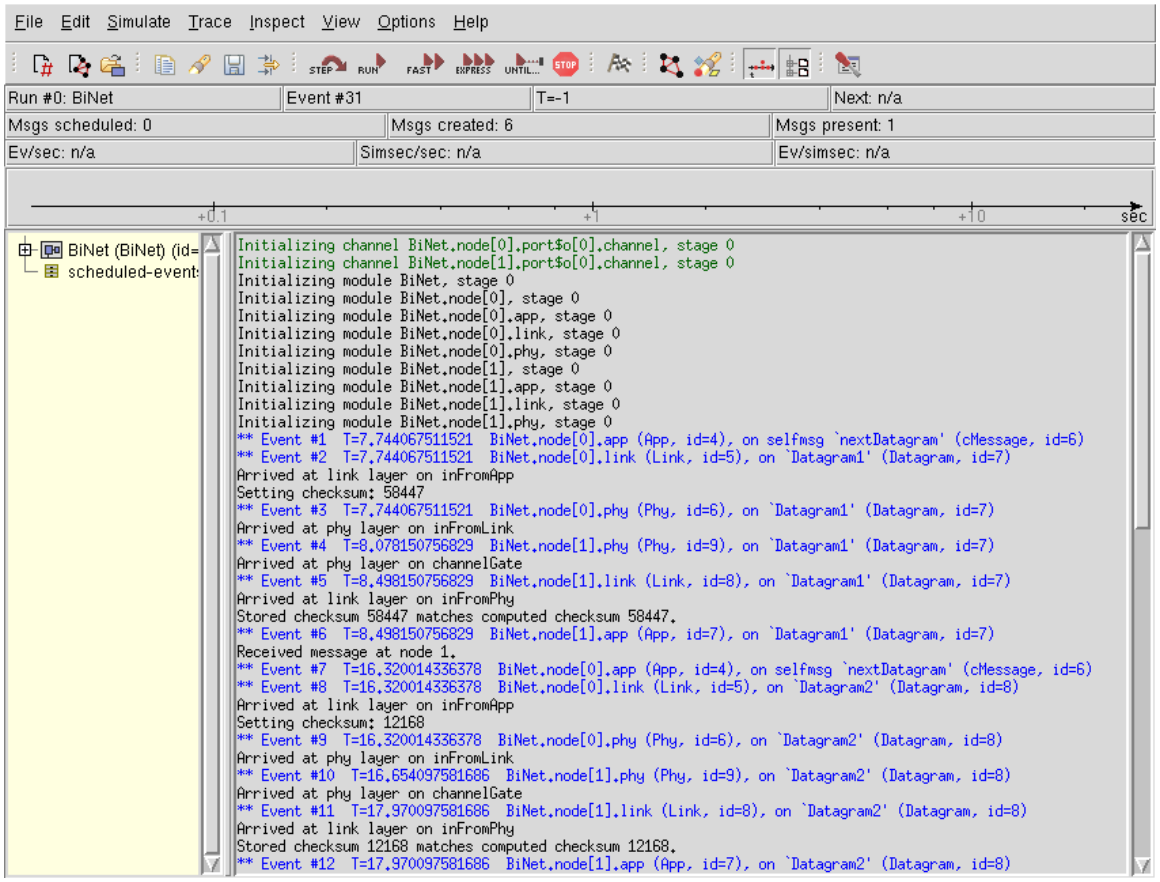

Figure 3-23: OMNeT++ Tk environment.

Finally, and most importantly, text output about the channel and BER of packets is displayed in the terminal from which the executable was run. Figure 3-24 shows the output produced for the second datagram generated by node[0]. The complete set of transmitted bits is displayed, followed by the source level, transmission loss, and noise level of the channel, which are used to compute the SNR of the received waveform. The filename of the randomly chosen impulse response is printed, followed by the BER, received bits, and expected and actual payloads. Additional pieces of relevant information, including the type of modulation and number of samples before and after convolution, are displayed for completeness.

```
Generating datagram 2...
Bits sent:
10001010:01010100:00000000:10111110:00000000:00000000:00000000:00000001
01010100:01101000:01100101:00100000:01010101:00101110:01010011:00101110
00100000:01001000:01101111:01110101:01110011:01100101:00100000:01101111
01100110:00100000:01010010:01100101:01110000:01110010:01100101:01110011
01100101:01101110:01110100:01100001:01110100:01101001:01110110:01100101
01110011:00100000:01110110:01101111:01110100:01100101:01100100:00100000
01111001:01100101:01110011:01110100:01100101:01110010:01100100:01100001
01111001:00100000:01110100:01101111:00100000:01100001:01100100:01101101
01101111:01101110:01101001:01110011:01101000:00100000:01010111:01101001
01101100:01110011:01101111:01101110:00100000:01101111:01110110:01100101
01110010:00100000:01110100:01101000:01100101:00100000:01100011:01101111
01101101:01101101:01100101:01101110:01110100:00101110:00100000:01001100
01101111:01110101:01110000:01100001:01110011:01110011:01101001:00100000
01110011:01100001:01101001:01100100:00100000:01110011:01101001:01101101
01101001:01101100:01100001:01110010:00100000:01100010:01100101:01101000
01100001:01110110:01101001:01101111:01110010:00100000:01110111:01101111
01110101:01101100:01100100:00100000:01100010:01100101:00100000:01110111
01101001:01101100:01100100:01101100:01111001:00100000:01110101:01101110
01100001:01100011:01100011:01100101:01110000:01110100:01100001:01100010
01101100:01100101:00100000:01101001:01101110:00100000:01110100:01101000
01100101:00100000:01010110:01101001:01110010:01100111:01101001:01101110
01101001:01100001:00100000:01001000:01101111:01110101:01110011:01100101
00100000:01101111:01100110:00100000:01000100:01100101:01101100:01100101
01100111:01100001:01110100:01100101:01110011:00101110:
PSK-modulated data in 166800 samples.
Source level           : 120.00 dB
Transmission loss      : 56.32 dB
Noise level            : 33.82 dB
SNR                    : 29.86 dB
Opening IR file data/IR_505m/IR_277.wav.
Number of samples in packet after convolution: 176400
Demodulating 190 bytes.
BER: 4.67%
Bits received:
10001010:01010100:00000000:10111110:00000000:00000000:00000000:00000001
01010110:01101000:01110101:00100000:01010101:00101111:01010011:00101111
00100000:01001000:01101111:01110101:01110011:01100101:00100000:01101111
```

Signal/channel properties

Bit error rate

```
01110111:00110000:01010010:01110101:01110000:01110010:01110101:01110011
01100101:00101111:00110100:00110001:01110100:01101001:01110111:01110101
00110011:00100000:01110111:01101111:01110100:00110101:01100100:00100000
01111001:01100101:01110011:01110100:00100101:01110010:01110100:00100001
01111001:00100000:01110100:01101111:00100000:01100001:01100100:01101101
01101111:01101110:00101001:01110011:01101000:00100000:01010111:01101001
01101100:01110011:01101111:01101110:00100000:00101111:01110110:00110101
01110010:00100000:01110100:01101000:01110101:00100000:01100011:01101111
01101101:01101101:01100101:00101110:00110100:00101111:00100000:01001100
00101111:01110101:01110000:01100001:01110011:01110011:01101001:00100000
01110011:01100001:01101001:01100100:00100000:00110011:01101001:01101101
01101001:01101100:00100001:01110010:00100000:01100011:01110101:00101000
01100001:01110111:00101001:01101111:01110010:00100000:00110011:01101111
01110101:00101100:00100100:00100000:01100011:01110101:10100000:01110111
01101001:01101100:01110100:00101100:00111001:00100000:01110101:01101111
00110001:01100011:01100011:01100101:00110000:01110100:00110001:01100011
01101100:00100101:00100000:01101001:01101111:00100000:01110100:00101000
01100101:00100000:01010110:01101001:01110010:01110111:00101001:01101110
00101001:01100001:00100000:01001000:01101111:01110101:01111001:01100101
00100000:01101111:01110111:00110000:01000100:00100101:01101110:01110101
01100111:01100001:01110100:00100101:01110011:00101110:
Expected payload: The U.S. House of Representatives voted yesterday to admonish
Wilson over the comment. Loupassi said similar behavior would be wildly
unacceptable in the Virginia House of Delegates.
Actual payload  : Vhu U/S/ House ow0Rupruse/41tiwu3 wot5d yest%rt!y to admon)sh
Wilson /v5r thu comme.4/ L/upassi said 3imil!r cu(aw)or 3ou,$ cu.wilt,9
uo1cce0t1cl% io t(e Virw)n)a Houye ow0D%nugat%s.
```

Figure 3-24: Terminal output of OMNeT++ simulation.

3.7 Future Work

There are several obvious opportunities for future work, all of which are somewhat inter-related. The first place to start is in evaluating the current single impulse response model for a time-variant channel such as the Hudson River estuary. It would no longer be sufficient to use a long chirp at the start of each test. The conditions will change much too rapidly for the impulse response estimate to remain valid for several minutes. In reality the conditions change too rapidly to remain constant across the duration of a packet transmission, with a coherence time of only 50 ms (see Section 2.5.9). However, since signals used to accurately estimate the channel cannot be transmitted at the same time as packets, the next best approach is to estimate the channel from the pilot signal used at the start of each packet. The rest of the procedure would remain intact, where the measured BER for each packet is compared to that which is obtained when convolving an impulse response estimate with the original waveform. Again, because packets are so much

longer than the channel's coherence time, it is expected that this method will not provide accurate results.

If the current model is indeed inaccurate, the next step would be to look into two dimensional convolution, as in Diamant and Chorev's IRM evaluation [Diamant 2005]. Several details must be analyzed carefully, including how to accurately construct the matrix of impulse response estimates. The creators of IRM, unfortunately, do not provide such details in their paper. For instance, if impulse response estimates are taken every 50 ms, is the matrix constructed so that the 50-ms blocks of samples all get the same copy of the impulse response estimate? Is there a way to interpolate the values between estimates to provide a more gradual transition between the measurements? If so, can the values be computed in reasonable time? As of now, these questions have not been investigated, at least not from the perspective of underwater acoustic communication.

The next iteration of the simulation must also account for multipath fading. Jakes's model can simulate a Rayleigh fading channel, MATLAB's Communications Blockset can model a Ricean fading channel, and Yip and Ng have developed a simulation model for Nakagami-*m* fading channels, where $m < 1$ [Yip 2000], which also covers the Gamma distribution. It might also be possible to use 2D convolution to partially account for fading, as the correlation coefficient will change in each impulse response estimate. Of course, each impulse response estimate must not be normalized to the same maximum value. However, this approach is not nearly as viable as the aforementioned models, since there is still only one estimate every 50 ms, and fading takes place on a much shorter time scale. Perhaps interpolated impulse responses can help, but it seems that too many interpolated estimates will be required to accurately model fading. If none of the suggested methods work correctly, one can always take the amplitude values from the recorded comb signal and apply them to the waveform in the simulator.

When the answers to these questions are known, or at least understood in greater detail, the simulator can be extended with other types of modulation techniques and methods of equalization. Upon the completion of that phase, the simulator would contain a very thorough model of a specific underwater channel and PHY layer of a network stack. Looking forward, one can envision using the simulator to test various MAC protocols or even investigate the potential benefits of cross-layer protocols. OMNeT++ makes such extensions easy.

Chapter 4
Softwater Modem
A Software Modem for Underwater Acoustic Communication

4.1 Overview

The Softwater Modem is a software modem for underwater acoustic communication that enables users to run applications on the familiar sockets interface without any additional hardware except for transducers and associated amplification. Data transmission and acquisition is performed by any ordinary sound card. A standard TCP or UDP transport protocol runs on top of IP, which runs on top of custom datalink and PHY layers that constitute the modem and are implemented in Java outside the operating system. The modem process is seamlessly made part of the protocol stack via the Linux TUN driver as shown in Figure 4-2. The modem uses FDMA with binary and 4-FSK in any frequency band supported by the computer's sound card and can run at any bit rate supplied by the user. The transmitter sends a per-packet LFM chirp signal that the receiver uses for packet synchronization as well as channel estimation, with the option of applying impulse response estimates to channel equalization. Frames can contain up to 255 bytes and are encoded with Reed-Solomon codes, for which the user can specify the number of parity bytes. The chapter describes in detail the architecture of this system, which currently demonstrates two-way communication as well as real-time channel estimation techniques. Additionally, it provides estimates of the processing time required per frame and the performance as given by decreasing BERs for increasing normalized SNRs in an AWGN channel.

4.2 Motivation

As discussed in Chapter 2, there is no such thing as a typical underwater acoustic communication channel. The large variation in channel conditions among different locations – espe-

cially the difference between deep water and shallow water – suggest that vastly different communication parameters (modulation technique, frequency band, frame length, error correction methods, etc.) would be optimal for different locations. Existing acoustic modems are implemented at least partly with custom hardware and paired with a fixed-point or floating-point DSP [Benthos 2010; Freitag 2005]. Such solutions typically offer a limited choice of operating parameters. Furthermore, in the case of commercial products, modem parameters are often chosen based on worst-case channel assumptions in order to maximize the modem's utility, necessitating a series of products, each tailored to specific environments that vary in depth, link distance, and expected severity of multipath [LinkQuest 2010], and each likely to be ill-suited for channels with properties differing from those for which the modem is customized. While logical, this is an unfortunate development because flexible, optimized communication is especially important in the bandwidth-limited underwater environment.

Recently there has been great interest in "software defined radio," (SDR) wherein software performs packet transmission / receipt functions and adjusts RF communication parameters on the fly in response to changing channel conditions. The success of SDR as applied to RF has been constrained by the fact that RF channels operate at high speed (e.g., multi-megabits per second), placing very tight real-time constraints on the necessary signal processing. The underwater acoustic environment is better suited for an SDR-like approach because acoustic link speeds are so much slower than RF links. There is plenty of time for modern hardware to perform even sophisticated signal processing needed to adapt to the challenging and rapidly changing underwater channel.

Accordingly, an all-software acoustic modem was built for underwater operation. The modem is able to sense and adapt to its environment on a very short time scale. In particular, a "sounding signal" precedes every packet and is used by the receiver to compute and apply the

channel's inverse impulse response to the modulated data signal that follows the sounding signal. In this way the receiver mitigates channel distortion on a packet-by-packet basis. While this approach is not as rapidly adaptive as a hardware DFE that can update filter taps on a symbol-by-symbol basis, data suggest that this approach should be able to cope with channels where the coherence time is at least on the order of the duration of a communication packet, as seen in places like the Massachusetts Bay east of Boston [Yang 2004].

Besides being able to adapt to channel conditions, a software modem offers the advantage of being far less costly than current hardware devices – such as the Benthos 013424 LF (9-14 kHz) omnidirectional modem at $8800/pair as of April 2009 – and of being easily configurable. Modem parameters can be selected to match the environment, thereby avoiding worst case assumptions and making communication more efficient. The carrier frequency, symbol time, and LFM chirp guard time are among the parameters that users can adjust. Moreover, this modem has the added benefit of being written in open source software that requires no license fees.

An additional advantage of this software architecture is that it supports TCP/IP based communication. Applications written to use the popular sockets interface can run unaltered on top of the modem layer, any number simultaneously. The effect is as if an Ethernet had been replaced by a (much slower) acoustic channel.

4.3 Related Work

There are several underwater acoustic modems in existence, either as commercial productions or research efforts, which make use of custom hardware to varying degrees. According to the specifications, the Benthos ATM series operates at baud rates of 140 – 15,360 bps, has a BER of 10^{-7} with high SNR, offers data redundancy, 1/2 rate convolutional coding, multipath guard period selection, and MFSK and PSK modulation schemes, and commonly operates over distances of 2 – 6 km [Benthos 2010]. LinkQuest offers many different UWM models, each of

which is tailored for a specific channel including “near vertical or horizontal” links and “long-range shallow to very shallow environments with very harsh multipath conditions” [LinkQuest 2010]. Some UWM models work in depths of 7000 meters while others are good for only 200 meters. Some have a payload data rate in the range of 80 to 320 bps, while others offer rates as high as 14,000 bps. DSPComm’s AquaComm modem uses DSSS/OFDM to achieve data rates of 100 – 480 bps (depending on the model) with a BER of at most 10^{-6} over links of up to 3 km [DSPComm 2010]. The Tritech AM-300 Acoustic Modem features two types of signaling to cover different types of channels [Tritech 2010]. Its spread spectrum option works between 25 – 100 bps, offering reliable data transfer in channels with SNR as low as -6 dB. The QPSK data link operates between 8 – 16 kbps, allowing for the transfer of high volumes of data. The specifications state that when using an array of hydrophones at the receiver, it is possible to transfer data over a 2-km horizontal link at 16 kbps.

The WHOI Micro-Modem, whose features and performance was discussed in detail in Section 2.5.11, offers FH-FSK signaling at a default data rate of 80 bps and BPSK and QPSK signaling with data rates up to 5 kbps [Freitag 2005]. The Reconfigurable Modem (rModem) [Sozer 2006] was built to simplify experimental studies of algorithms on all layers of the network stack with rapid prototyping via Simulink tools [Simulink 2010]. It has four input and output channels, an anti-aliasing filter, 240 kHz analog-to-digital converters (ADC) and digital-to-analog converters (DAC), automatic gain control (AGC), an FPGA and floating point DSP, and onboard memory. It was tested in Woods Hole, MA, in 2006 in a store-and-forward network of four nodes. Intermediate nodes time-reversed the signal before retransmitting it with a new preamble and training sequence. However, demodulation and detection, the most basic functionality of a modem, was not tested and left as the subject of a future experiment, which does not seem to have been conducted (or may just be undocumented).

Several purely software-defined modem implementations also exist. Soundmodem was one of the first packet radio platforms made to run on a standard PC with a sound card [Sailer 2000]. It offers several modulation techniques, including FSK, PSK, and QPSK, and can be interfaced with the AX.25 stack on Windows, Linux, or UNIX. Soundmodem was tested in the office tub environment as well as with a cable attached directly from the line output of one computer to the mic input of the other computer. In general, the software seems to have synchronization problems, causing very few packets to be received correctly. A thorough walk-through of the code for FSK did not produce a clear picture of what was going wrong with reception but did reveal some programming errors.

With J-QAM [Olds 2008], high data rates (up to 400 kbps) have been achieved by using a PC sound card with RF transmission. Since the phase-coherent detection methods necessary for QAM modulation generally work well only in vertical underwater acoustic channels with little multipath distortion [Pelekanakis 2003] and funding ran out, J-QAM was not investigated further. However, if funding becomes available for more experiments in the Hudson River estuary, J-QAM is among the applications to test because it is well-written and thoroughly documented, and would lay to rest questions about using QAM in extremely shallow, horizontal channels.

GNU Radio [GNU 2010] is probably the largest, most flexible SDR platform to date. It features numerous modulation/demodulation and signal processing techniques for radio communication and integrates easily with the Universal Software Radio Peripheral (USRP), an ADC/DAC motherboard with a USB interface [Ettus 2010]. Signal processing blocks are written in C++, while flow graphs that connect the signal processing blocks are written in Python. Figure 4-1 shows a flow graph constructed in GRC for a GMSK[12] modulator. Unfortunately, GNU Ra-

[12] Minimum Shift Keying (MSK) is a digital modulation scheme where the phase remains continuous while the frequency changes. To reduce side lobes in MSK transmissions, MSK is augmented with a pre-modulation Gaussian-shaped low-pass filter. This enhanced technique is known as Gaussian Minimum Shift Keying (GMSK).

dio has a steep learning curve and is in a constant state of change. When it was first considered for use in the Softwater Modem, noncoherent FSK was the modulation technique of choice, since it is robust to multipath fading channels. However, FSK modulation was removed from GNU Radio's source tree because it didn't work correctly and no one was maintaining it. In addition, GNU Radio is limited in that it operates on streams of data. It is not oriented to processing flows of variable-length packets commonly used in underwater acoustic communication systems.

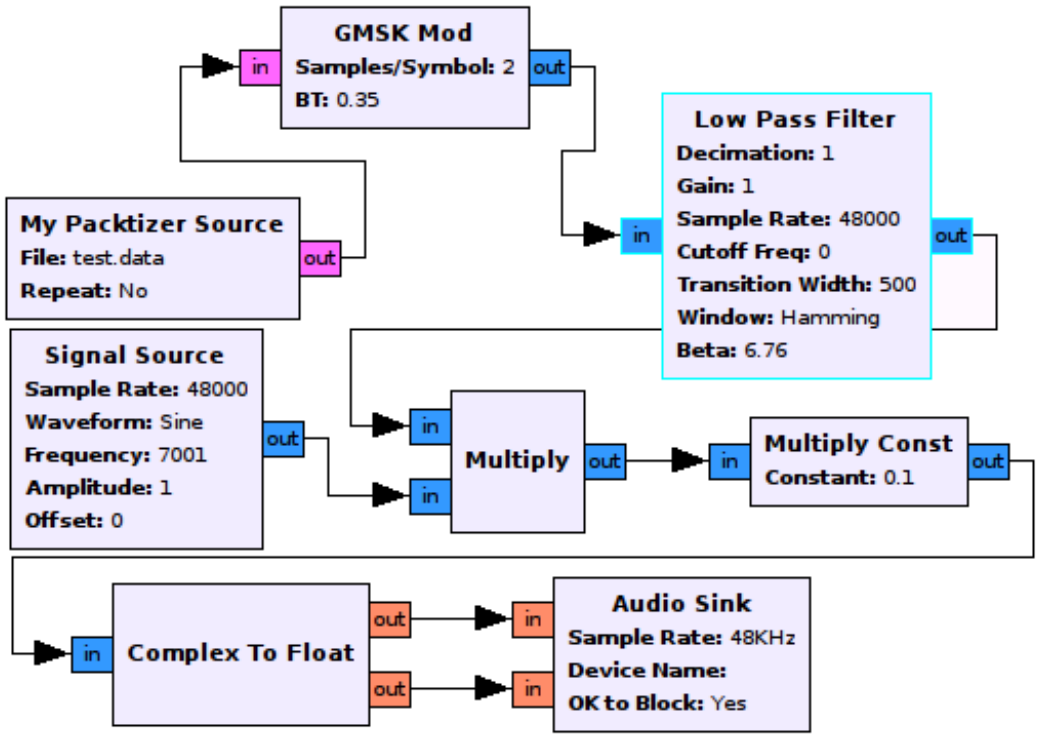

Figure 4-1: GRC (GNU Radio Companion) flow graph [Miller 2009].

As GNU Radio matures, some efforts are being made to use it for underwater acoustic communication, as in the Underwater Acoustic Networking plaTform (UANT) developed by the Networked and Embedded Systems Laboratory at UCLA [Torres 2009]. UANT uses GNU Radio to achieve configurability at the physical layer. TinyOS has been adopted for the use on the network platform, since it provides a full network stack. UANT affords users the flexibility to change the properties of the acoustic modem at run time to adapt to the dynamic characteristics of

the underwater channel, though none of this functionality is automated. In reality, UANT seems to be a nice GUI on top of a network stack built upon GNU Radio. Its limitations are inherently those of GNU Radio.

Several other standalone modem prototypes have been developed over the years. The prototype from UCSB [Fu 2006] combines DSP techniques, hardware-software integration, and network protocols, but operates at a fixed rate of 161 bps. The design from Yuan-Ze University uses a PC sound card with MATLAB as an SDR OFDM communication system [Hwang 2003]; however, the system cannot operate in real time. Furthermore, the initial code optimizations performed at UCONN [Yan 2007] did not result in a real-time DSP-based OFDM receiver. Finally, by 2009 researchers at UCONN produced the Aqua-fModem, a real-time OFDM modem prototype that uses a bandwidth of 5.5 kHz and yields an overall data rate of 3.1 kbps after 1/2 rate nonbinary LDPC (low-density parity-check) coding and QPSK modulation [Zhou 2009]. No other information about this system is available.

Since each of the aforementioned systems has limitations, the focus here is shifted to the implementation of a flexible software modem that performs well in various types of underwater acoustic channels while requiring reasonable amounts of processing power, as those found in an average laptop PC. The original goal was to develop a platform with several modulation/demodulation techniques that would be able to sense and adapt to the channel's changing conditions and propagate that information to neighboring nodes via a handshaking protocol that runs on a control channel. However, since the amount of development time proved to be too much for one person, the project was scaled back. While the modem, presently called the Softwater Modem, currently offers only binary and 4-FSK, it has adjustable parameters that allow it to perform well in many environments, in particular, shallow water channels. In addition to carrier frequency and symbol rate parameters, the Softwater Modem has parameters for setting the

threshold for frame detection and length of the chirp signal and guard time, toggling between half and full duplex, adjusting the number of payload and parity bytes, and properly estimating the channel's impulse response as to apply an inverse filter (zero-forcing equalizer). All of these features can be exploited in future work that proceeds with the original goal for the project. Furthermore, unlike most previous efforts, the Software Modem permits users to run existing TCP/IP applications across nodes in the system.

4.4 System Architecture

4.4.1 Software Architecture

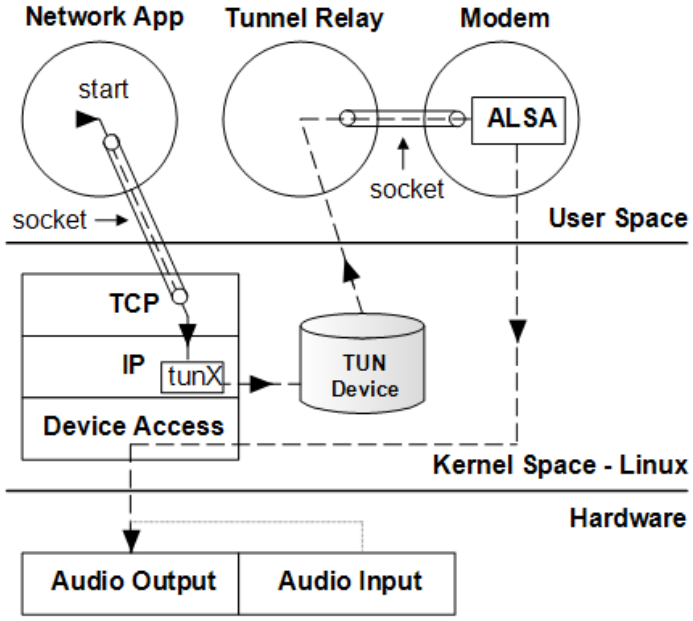

Figure 4-2: Software architecture of acoustic modem, with arrows depicting the flow of data generated by the network application through the system and down to the sound card, where it is emitted as an analog bandpass modulated waveform.

The overall architecture of the system includes three layers of user space applications. The highest layer is the application itself, which can use either TCP or UDP. The lowest layer is the Java application that implements the functionality of an acoustic modem. Between the two is the tunnel relay application, which is responsible for passing IP datagrams between the network

application and Java modem. Figure 4-2 depicts the overall architecture of the system and shows how each of the component applications is linked.

The tunnel relay application is written in C and utilizes the Linux TUN device [Krasnyansky 2010]. If use with Windows is required, the tunnel relay program can be rewritten to exploit the Microsoft TUN Miniport Adapter instead. The TUN device is a virtual point-to-point network device designed to provide low level kernel support for IP tunneling. It interfaces with a user-space application via the /dev/net/tunX character device and the tunX virtual point-to-point interface. A user-space application can write datagrams to /dev/net/tunX, and the kernel will receive them from the tunX interface. Similarly, every datagram that the kernel writes to the interface can be read from the /dev/net/tunX device.

The tunX device is bound to a private IP address in the 10.0.0.0 – 10.255.255.255 range to avoid interference with applications running on the Internet. The *route* command informs the kernel that datagrams destined for a particular host (or network) should be associated with the tunX device. Therefore, it is possible to build a routing table specifically for the independent acoustic network.

The tunnel relay application is also responsible for setting the MTU of the tunX device. This process informs the kernel about how to break large data streams into datagrams of manageable size, most often of which are very small in comparison to Ethernet (<255 bytes vs. 1500 bytes) for transmission through the underwater channel. An application that wishes to have precise control over packet construction can query the OS to determine the MTU of the tunX device to ensure it constructs datagrams that fit into MTU bytes.

The tunnel relay application communicates with the Java modem via UDP, as sockets are the only form of IPC that works with Java. Both the tunnel relay application and Java modem bind to the local loopback IP address, but with different port numbers, to allow full-duplex com-

munication between the two processes. Thus, the Java modem listens on one socket for datagrams from the tunX device that need to be transmitted acoustically, while it sends datagrams that have been received acoustically on the other socket where the tunnel relay program is listening.

To better understand how this system works, it is helpful to trace the steps that a message takes as it passes from the network application to the sound card. When the network application has data to send, it writes the data to a socket. The data passes to the Linux kernel, where transport and IP headers are added. If the destination address is one mapped to the TUN device, the datagram will arrive there, enabling the tunnel relay application to forward it to the UDP port where the Java modem is listening. If the destination address is not mapped to the TUN device, the datagram will go directly to the link layer of the kernel and out through the computer's network interface card, allowing the acoustic network to peacefully coexist with other networks on the system. The Java modem adds its own link layer headers to the datagram, and converts the frame into a modulated acoustic waveform. The modem then writes the acoustic signal to the output device in the Java Sound API, which actually makes use of ALSA (Advanced Linux Sound Architecture) in the JVM. The ALSA library traps to the kernel, which handles device access, and sends the signal to the output of the sound card. The process by which the system receives an acoustic frame is essentially the reverse of the aforementioned description.

4.4.2 Associated Hardware

Two laptops have been used in the development of this system, a Lenovo T60p and a newer T500. The T60p has a T7200 dual-core Intel processor running at 2.0 GHz, 2 GB of memory, and an ADI1981 codec running on top of the integrated Intel high-definition audio (HDA) controller. The sound card supports a maximum sampling rate of 48 kHz. The T500 has a P8400 dual-core Intel processor running at 2.26 GHz, 2 GB of memory, and a Conexant CX20561 codec paired with an Intel HDA controller. This integrated sound card supports a maximum sampling

rate of 192 kHz. For the sake of comparing software performance statistics, a desktop PC with a Q6600 quad-core Intel processor over-clocked to 3.0 GHz, 8 GB of memory, and integrated ADI1988/HDA audio was also brought in. All three systems were running Ubuntu Intrepid 8.10 with gcc 4.3.2, ALSA 1.0.17, and Java 1.6.0 Update 13.

4.5 Modem Architecture

The modem portion of the system is written entirely in Java. While the use of Java does come at a cost to execution speed, the laptops are still able to keep up with continuously demodulating packets. Moreover, the benefits of platform independence; modular, extensible code; and relatively fast development time further support the use of Java for the modem prototype.

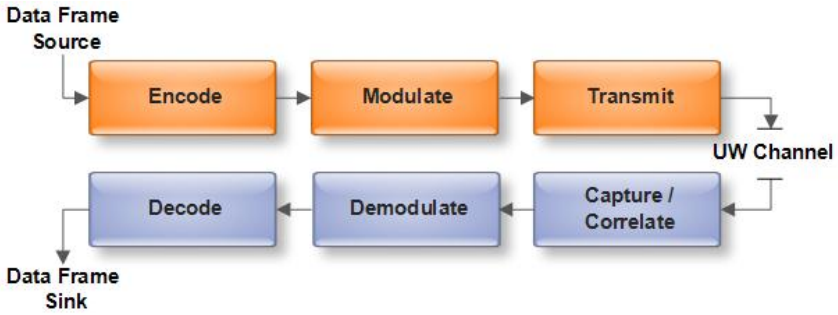

Figure 4-3: Processing blocks within the Java modem.

The modem functionality is divided into two main tasks, transmit and receive, which can be executed in parallel. Both of these functions are implemented as a series of stages which are processed by threaded objects. Pairs of adjacent stages communicate via a thread-safe queue, where one stage places its output on the queue for the next stage to use as its input. Figure 4-3 summarizes the processing blocks that comprise the Java acoustic modem.

4.5.1 Transmitter Design

The transmitter consists of three stages – the source encoder, the modulator, and playback mechanism. The encoder reads incoming datagrams from the UDP socket attached to the tunnel relay application, wraps them in a frame header, and optionally applies Reed-Solomon codes.

The modulator converts the incoming byte-oriented data frame into symbols of 0s and 1s for binary FSK (or the 2-bit symbols 00, 01, 10, and 11 for 4-FSK) and then translates the symbols into the samples of the sine wave that correspond to the frequency representing a given symbol. The modulator also prepends an LFM chirp signal and guard time block to the beginning of a data frame for synchronization and channel estimation purposes at the receiver. Finally, the transmitter takes the buffered modulated signal and sends it to the sound card for playback. The transmitter can also optionally record each outgoing modulated data frame in a wav file for future reference.

4.5.2 Receiver Design

The receiver consists of three stages as well – the correlator, demodulator, and decoder. The correlator continually reads blocks of samples from the sound card. The block itself must be twice as long as the LFM chirp signal that precedes each packet, so that the chirp signal is guaranteed to fit within two consecutive blocks. For every incoming block, the correlator concatenates it with the previous block before using cross-correlation to detect the start of a frame. If a frame is detected, the exact number of samples within a frame is buffered and then placed on the queue for the demodulator to pick up.

The demodulator converts an acoustic signal into a bit stream of 0s and 1s. It optionally takes the impulse response obtained during frame detection, inverts it with the Levinson-Durbin algorithm [Proakis 2007], and convolves it with the signal as a means of performing channel equalization on a packet-by-packet basis. Regardless of whether inverse filtering is applied, the demodulator bandpass filters the signal, holds a tournament to see which of the carriers has the strongest signal over the duration of a symbol, and outputs the corresponding symbol (0 or 1 for binary FSK; 00, 01, 10, 11 for 4-FSK). It also optionally computes the SNR for the frame before placing it on the queue for the decoder to pick up. At the user's request, the demodulator can also

record each incoming unprocessed frame and demodulated frame in a separate wav file and each impulse response and inverse impulse response in a csv file, which can be post-processed with the MATLAB scripts included with the source code.

The decoder applies Reed-Solomon codes to an incoming data frame and allows the user to know if no errors were detected, if errors were found and corrected, or if errors were found but could not be corrected. The decoder also verifies the CRC[13] in the header before extracting the frame payload, or IP datagram, and sending it out to the TUN device via the UDP socket.

4.6 Frame Format

All data frames begin with an LFM chirp signal followed by a block of silence, known as the guard time. The duration of both the chirp signal and guard time can be configured by the user. Modulated data appears after the guard time, beginning with the frame header. The 4-byte frame header format is extremely simple, containing only two fields, a 16-bit CRC and 16-bit length attribute. All other information relevant for communication is contained in the headers for the IP and the transport layer, whether TCP or UDP. The IP header is 20 bytes, while the TCP and UDP headers are 20 and 8 bytes, respectively. The TCP header can be longer if options are enabled within the operating system's network stack. The actual payload appears after all headers and can fill the remaining bytes of the frame up to the 255-byte limit.

Reed-Solomon (R-S) codes can be enabled by the user. The R-S codes can contain an arbitrary number of parity bytes as long as the sum of parity, header, and payload bytes does not exceed the 255-byte limit. Since R-S codes operating on 8-bit symbols can have $n = 2^8-1 = 255$ symbols per block, 255 was chosen as the upper bound on frame size. In comparison with the frame sizes supported by the Micro-Modem [Freitag 2005], 255 is a reasonable limit and should

[13] When there are too many errors in the frame, R-S can pick a valid codeword that is not the codeword that was transmitted. With a valid but incorrect codeword selected, R-S will not indicate any error syndrome, and the only way to know for certain whether the data is correct is with some other detection method, like a CRC [Jacobsen 2008].

be more than adequate for underwater channels requiring use of noncoherent FSK demodulation. If enabled, the R-S parity bytes appear after the frame header and before the IP header. Figure 4-4 depicts the format of a data frame for the Softwater Modem. Note that the sizes of the blocks within the diagram are not drawn to scale.

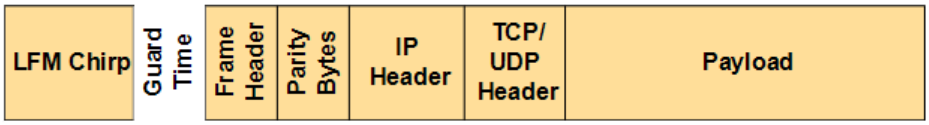

Figure 4-4: Format of a data frame.

4.7 Signal Processing

As described in Section 3.3.4, the modulation index is set to 1 to obtain reasonable performance with noncoherent FSK detection. In fact, this index value corresponds to the minimum tone spacing for noncoherent FSK signaling, which occurs when

$$|f_0 - f_1| = 1/T, \tag{4.1}$$

where f_0 is the frequency of the tone representing a '0', f_1 is the frequency of the tone representing a '1', and T is the symbol duration, equivalent to $1/R$, so that the two tones remain orthogonal [Sklar 2001]. The same modulation index applies when the modem is operating in 4-FSK mode.

The LFM chirp signal that precedes a frame is generated as an array of floats according to the formula

$$x[t] = \cos\left(2\pi\left(f_0 + \frac{k}{2}t\right)t\right), \tag{4.2}$$

where f_0 is the starting frequency at time $t = 0$ and k is the rate of frequency increase. The chirp signal covers only the frequency band required by the data modulation, or $4 \times f_d$, starting f_d Hz (see Section 3.3.4) lower than the lowest tone in the transmission. If a slow data rate results in a chirp less than 1 kHz wide, the chirp is expanded to cover 1 full kHz in order to result in a manageable autocorrelation function for use in frame synchronization. A second buffer containing the

samples of the reference LFM chirp used to mark incoming data frames is precomputed and stored at the receiver.

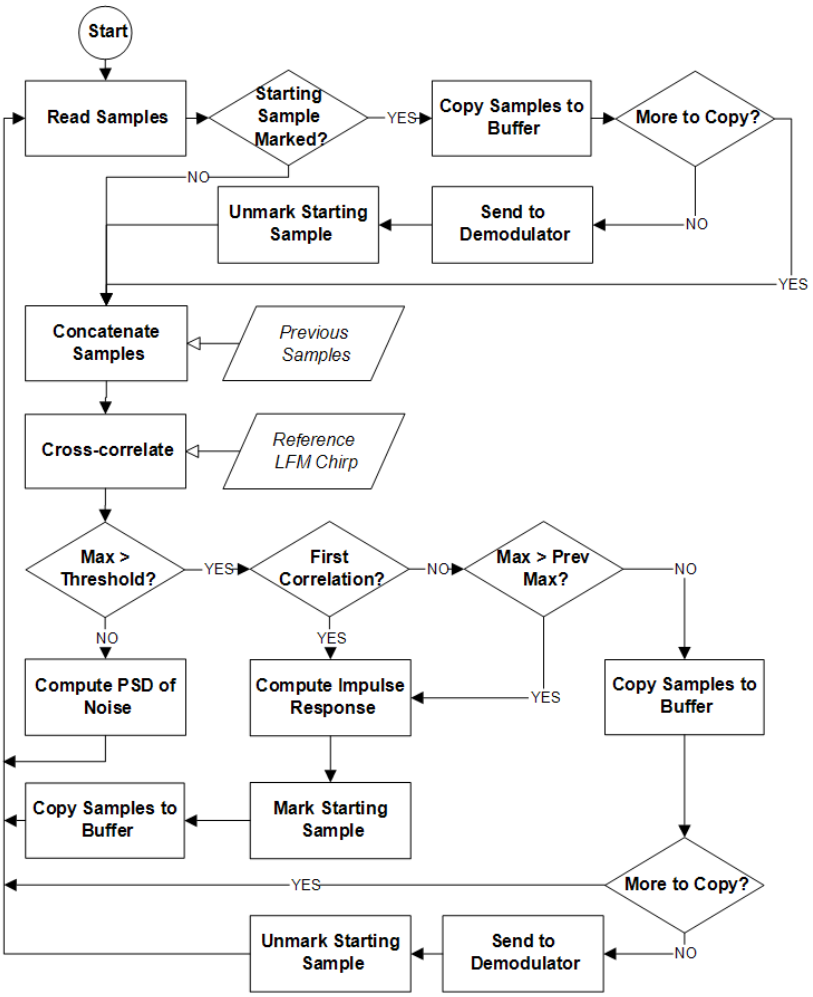

Figure 4-5: Capture/correlate block of receiver.

As seen in Figure 4-5, the receiver continuously reads a block of samples from the audio device and concatenates it with the previously read block before cross-correlating the samples with the reference LFM chirp signal. The implementation is based on that in *Numerical Recipes* [Press 2007], using FFT to first convert the signals into the corresponding frequency-domain representations, multiplying the transforms, and returning the result to a time-domain signal via

IFFT. The FFT-based cross-correlation routine is orders of magnitude faster than the time-domain approach, which cannot be executed in real time.

The maximum, minimum, and average (excluding the absolute value of the maximum and minimum) values of the signal returned by the cross-correlation routine are then computed. If the current iteration is the first time the correlation exceeds the user-defined multiple of the average correlation, or if the maximum value is greater than the maximum found during the previous iteration, the index of the maximum value is marked as the potential starting point of a data frame. If the user specified that the modem should look for significant arrivals occurring before the strongest component, the modem does so and buffers the resulting estimate of the channel's impulse response at the time the data frame is received. A data frame is deemed present when the starting sample is marked and the current maximum does not exceed the threshold or the previous maximum. At this point, the modem begins copying incoming samples to a buffer that is the size of is MTU + 4 bytes. Once all the samples have arrived, the receiver passes the data buffer, impulse response estimate, and last noise amplitude level to the demodulator block for processing. If no data frame is present during a given iteration of the loop, the samples are assumed to be noise, and the noise amplitude is computed and stored, overwriting the previous value.

The demodulator block is comprised of a series of stages that implement noncoherent FSK detection via the envelope detector. Figure 4-6 illustrates the demodulation process for binary FSK demodulation, which has the same basic structure as 4-FSK demodulation.

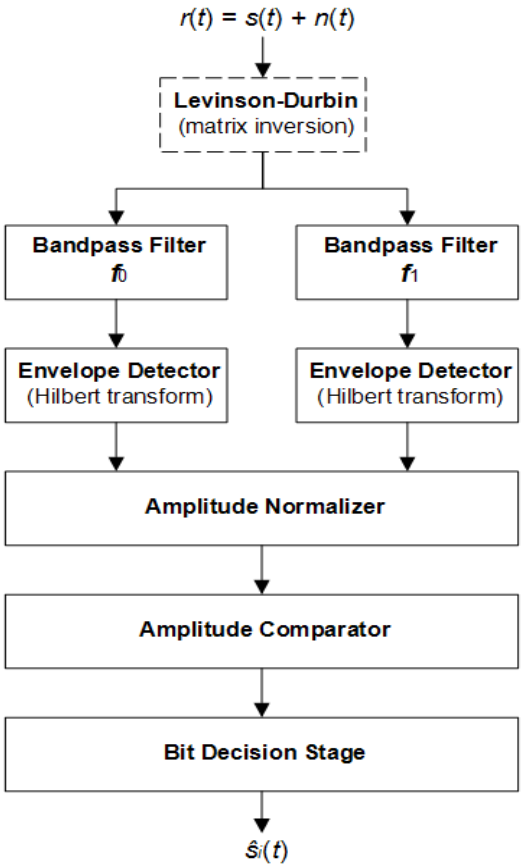

Figure 4-6: Stages of noncoherent FSK detection.

The first stage, which serves as a means of channel equalization on a frame-by-frame basis, is optional. If enabled by the user, this stage takes the impulse response estimate obtained via the cross-correlation of the received LFM chirp and reference LFM chirp and inverts it by means of the Levinson-Durbin algorithm. As long as the channel remains fairly constant over the duration of a data frame and the SNR > 12 dB to prevent noisy impulse response estimates, this method works to mitigate ISI induced by multipath arrivals and problems with the frequency response of the transducers.

Figure 4-7 shows the time-domain view of a portion of a data frame that passed through an air-based acoustic channel between a pair of desktop Accent Acoustics PC speakers and an Altec Lansing microphone. The data rate was 2 kbps (symbol time of 0.5 ms), the carrier tones

were 4 and 6 kHz, and the 50-ms LFM chirp preceding the data transmission swept from 3 to 7 kHz. The chirp signal exhibits a dip around 4 kHz, the same frequency used to represent '0' data bits. Figure 4-8 shows the delay spread of the channel. Because of this unequal representation of tones, this data frame could not be successfully demodulated. There were 64 bit errors out of 1184 bits, producing a BER of 5.41%. The default 16 parity bytes were transmitted as well, but even the R-S codes could not save this frame.

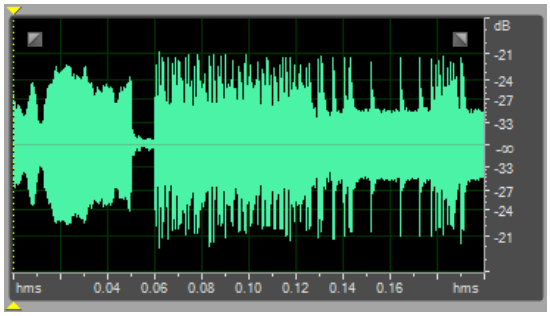

Figure 4-7: Unequalized reception of data frame.

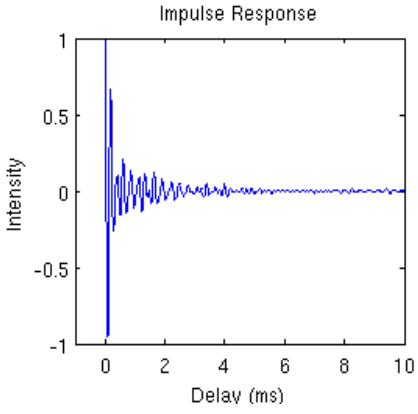

Figure 4-8: Delay spread of channel in 3-7 kHz band.

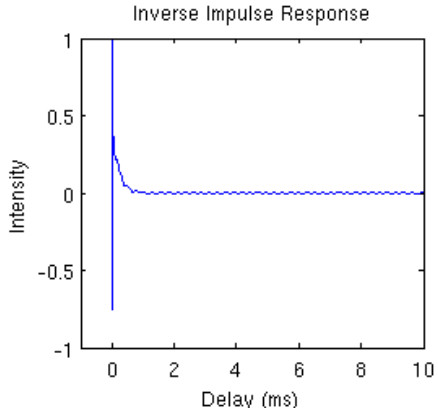

Figure 4-9: Inverse impulse response of channel in 3-7 kHz band.

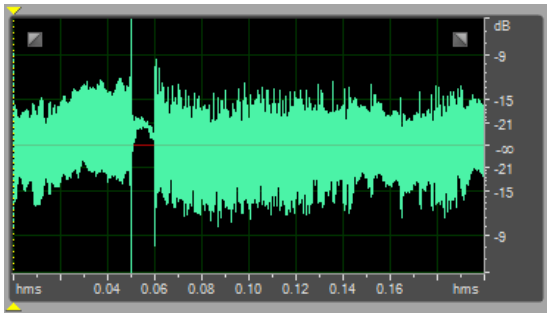

Figure 4-10: Reception of equalized data frame.

The Levinson-Durbin algorithm inverts the channel's impulse response h by solving for h_{inv} in the Toeplitz system $R \times h_{inv} = q$, where R is the symmetric Toeplitz matrix containing the positive lags of h's autocorrelation function and q is the Dirac delta function. The inverse impulse response is then convolved with the samples containing the LFM chirp and modulated data via FFT convolution. Figure 4-9 depicts the inverted form of the impulse response. Figure 4-10 displays the equalized time-domain view of the same data frame shown in Figure 4-7. It is apparent that the inverse filter does not exhibit linear phase, since the guard time between the chirp and the modulated data does not have 0 mean. However, since the modem has been designed to work with noncoherent detection, this filter attribute does not pose a problem. Also, the spectral splatter associated with rapidly turning the transmitter on and off becomes more evident after applying the inverse filter, with large spikes appearing at the beginning and end of the acoustic signals. While spectral splatter does not degrade performance in a point-to-point system, it is something to consider when other devices begin sharing the channel, especially one that is bandwidth-limited. Since Reed-Solomon codes perform well when errors occur in bursts, they have been built into the system as a means of combating spectral splatter from neighboring devices.

Bandpass filtering is the first mandatory step in the modem's implementation of noncoherent detection. When using binary FSK, the incoming signal is passed through two separate filters (four in 4-FSK) to eliminate signals outside the band corresponding to the tones representing

bits. As described in Section 3.3.6, second-order IIR filters based on the design in [Smith 2003] are used. The filters have been designed to operate with bandwidth $1/T$ centered on the tones representing 0s and 1s.

Upon completion of the filtering stage, the resulting signals are passed to the envelope detection stage, which first applies the Hilbert transform to obtain the imaginary part of a signal containing only real values. The envelope is then obtained by applying the formula in Equation (2.22).

Regardless of whether the inverse filter is enabled, the resulting envelopes undergo normalization. This processing block first determines the maximum value in each envelope and then scales the other envelope(s) so that the maximums are all equal. In practice, this over-simplified "equalizer" helps to reduce ISI caused by transducers with a non-linear frequency response.

The amplitude comparator produces another intermediate signal that shows which envelope contains greater amplitude at every sample in the signal. For binary FSK, the comparator subtracts one envelope from the other. With 4-FSK, a simple tournament is held for each sample to see which of the signals possesses the greatest amplitude, producing a new signal with only 4 possible values.

Finally, the demodulator makes a decision as to which symbol was present during a given time period. The modem samples many times per symbol period in an attempt to make the best decision, at virtually no cost in execution time. Starting at samples corresponding to 60% of the duration of the symbol and running up to the last sample of the symbol, the modem counts how many samples correspond to each symbol type. The one with the highest count is selected. In practice, sampling in the second half of the symbol produces better results, since the recursion of the IIR filter leads to a "ramp-up" period at the beginning of the pulse and because the filter often

exhibits “ringing” shortly after the ramp-up time. Thus, one can effectively work around the negative effects of IIR filters while benefitting from their computational efficiency.

4.8 Control Interface

The modem allows the user to alter its functionality by editing a text file containing name/value pairs. Upon startup, the modem parses the properties file and uses the values of the parameters for its operation. The following options are supported:

CHIRP_MS = <integer>. CHIRP_MS indicates the length in milliseconds of the LFM chirp signal that precedes frame transmission. If the INVERSE_FILTER option is enabled, the value should not exceed the coherence time of the channel in order to ensure the channel’s characteristics have not changed during the transmission of the sounding signal. Ideally, the signal should be as short as possible, with enough samples to provide a high value of correlation in the presence of a data frame. 50 ms works well in most applications.

BASE_FREQUENCY_RX / TX = <integer>. This number is the frequency in Hz of the lowest carrier in the received/transmitted signal.

FULL_DUPLEX = <TRUE/FALSE>. FULL_DUPLEX indicates whether or not the modem can receive and transmit simultaneously. For testing purposes, if full-duplex operation is enabled, setting BASE_FREQUENCY_TX to the same frequency as BASE_FREQUENCY_RX will allow the modem to demodulate its own transmissions.

GUARD_MS = <integer>. GUARD_MS is the amount of silence that appears after the chirp signal and before the modulated data. This silence is necessary to obtain a clean impulse response estimate. GUARD_MS also dictates the maximum length for which the impulse response (and possibly its inverse) will be computed. Therefore, it should be set be set to a value that exceeds the multipath spread of the channel.

IMPULSE_RISE_MS = <decimal>. In case the strongest arrival is not the first component of the impulse response, setting this option allows the modem to look for components before the main arrival that exceed the threshold of ¼ the intensity (-6 dB) of the main arrival. This option is useful when computing the inverse impulse response of the channel. The value is usually small, not exceeding 1 ms.

INVERSE_FILTER = <TRUE/FALSE>. INVERSE_FILTER indicates whether or not to apply Levinson-Durbin matrix inversion for channel equalization on a frame-by-frame basis.

NUMBER_OF_CARRIERS = <2/4>. This option indicates if binary or 4-FSK is to be used.

PARITY_BYTES = <integer>. PARITY_BYTES indicates the number of bytes used as parity within R-S codes. The maximum frame size is 255 bytes, of which 4 bytes are used for the frame header, 20 for the IP header, and 20 for TCP header (and more if options are enabled). Therefore, the value must be set accordingly to maintain data flow. 16 bytes is the default value.

PAYLOAD_SIZE_IN_BYTES = <integer>. This number indicates the payload size, the size of the frame excluding the 4 byte header. The maximum frame size is 255 bytes, for which the sum of PARITY_BYTES, PAYLOAD_SIZE_IN_BYTES, and the 4 byte frame header cannot exceed.

SYMBOLS_PER_SECOND = <integer>. This option represents the symbol rate of the modem. For binary FSK, the symbol rate is equivalent to the bit rate, and for 4-FSK, it is equal to ½ the bit rate.

THRESHOLD = <integer>. This parameter adjusts the mechanism for frame detection. Frames are detected in a block of samples if correlation = THRESHOLD × (average correlation minus the absolute value of the maximum and absolute value of the minimum within the block).

Other parameters supported by the modem change the sampling rate, output verbosity, number of channels, and endianness of the samples being read from the sound card and determine whether SNR should be computed for every data frame received.

4.9 Performance

4.9.1 Computational Performance

Table 4-1: Processing time of subroutines.

	Desktop	Laptop T60p	Laptop T500
Transmit			
a. Modulate	8.00	12.00	13.40
b. Encode Reed-Solomon	9.33	74.40	82.60
Sum (a:b)	17.33	86.40	96.00
Frame duration	1244.00	1244.00	1244.00
Comp Time/Signal Length	1.39 %	6.95 %	7.72 %
Receive			
c. Cross-correlation	2.36	5.03	4.46
Block length	85.33	85.33	85.33
Comp Time/Signal Length	2.77%	5.89%	5.23%
Demodulate			
d. Levinson-Durbin	3.40	3.80	5.33
e. FFT convolution	29.80	65.00	43.83
f. Bandpass filtering	2.60	3.60	4.16
g. Envelope detection	61.60	117.60	84.50

h. Normalizer	1.60	3.70	1.50
i. Comparator	0.40	2.00	0.33
j. Bit Decision	0.40	1.60	0.50
k. Decode Reed-Solomon	1.33	21.40	17.40
l. Write 2 wav files	2.00	3.40	2.60
m. Write IR data to csv file	16.33	55.40	36.00
Sum (d:m)	119.46	277.50	196.15
Frame duration	1244.00	1244.00	1244.00
Comp Time/Signal Length	9.60 %	22.31%	15.77%

The performance of the Softwater Modem has been characterized in terms of its CPU execution time and bit error rate. Table 4-1 lists the running times of various methods within the Java code in milliseconds on three different platforms. JRat [JRat 2010] was used to profile the application. After compilation the class files were injected with instructions by JRat to enable the collection of performance statistics. The modem was then run through the JRat agent with the key options set as follows:

CHIRP_MS = 50
FULL_DUPLEX = TRUE
GUARD_MS = 10
INVERSE_FILTER = TRUE
NUMBER_OF_CARRIERS = 2
PARITY_BYTES = 16
PAYLOAD_SIZE_IN_BYTES = 128
SAMPLING_RATE= 48000
SYMBOLS_PER_SECOND = 1000

For each platform, the values shown in Table 4-1 are the averages computed over 5 frame transmissions. Each frame consisted of a 4-byte frame header, 16 parity bytes, and 128 bytes of payload, for a total of 1184 bits. At 1000 bits/second, the modulated block of data takes 1184 ms to transmit. The total frame transmission time is equal to 1184 ms plus 50 ms for the LFM chirp signal plus another 10 ms for the guard time, for a total of 1244 ms. As can be seen in the table, the time to encode the data with R-S codes and modulate the entire block is longest on the T500, taking 96 ms, or 7.72% of the total frame transmission time.

Data from the sound card was read in blocks of 4096 samples, which correspond to ap-

proximately 85.33 ms. As stated earlier, two blocks are concatenated before performing the cross-correlation operation. Thus, performing cross-correlation on 8192 samples appears to execute slowest on the T60p, consuming 5.03 ms on average. This time represents 5.89% of the duration of an incoming block of audio samples.

The demodulation procedure operates on the modulated data portion of the frame, excluding the chirp signal and guard time. The Levinson-Durbin algorithm, which is $O(n^2)$, operates on GUARD_MS ms of samples. In this performance evaluation, 10 ms of guard time sampled at 48 kHz equates to 480 samples. The T500 was the slowest platform for this operation, taking 5.33 ms. The profiling was repeated for a 20 ms guard time, or 960 samples, where matrix inversion took 13.50 ms on average. For small numbers of samples, the computation time for Levinson-Durbin matrix inversion seems quite practical, especially given that it is running though a Java VM.

The two subroutines that consistently consume the most time on each system are convolution and envelope detection. This is not surprising, as each requires computing the forward and inverse FFT of 56,832 samples. Faster FFT implementations do exist. Since the worst-case demodulation time is already about 22% of the duration of the data frame with binary FSK (and more with 4-FSK), it might be beneficial to investigate using other FFT implementations, such as FFTW [FFTW 2010] which now includes Java wrappers.

The performance numbers here indicate that, even with Java, the modem is able to operate in real time. The demodulation time is clearly less than the transmission delay. Despite the claims for real-time operation, the modem exhibits significant latency, given by the row *Sum(d:m)* in Table 4-1. Since the entire frame is buffered before any processing occurs, there is a feeling of sluggishness when operating the system in an environment where the processing time masks propagation delay. This condition is less noticeable when operating at high data rates,

where the processing time is a small fraction of a second, and exacerbated at low data rates, where there are many more samples per symbol to process. The demodulated frame can appear at the application more than a full second after the last sample of the frame is received. Unfortunately, there is no easy way to change this behavior; the receiver would need to be redesigned from the bottom up.

4.9.2 Performance in AWGN Channel

The modem's implementation of noncoherent FSK detection is basically the same as the envelope detector described in Section 3.3.6; therefore, its performance in the office test environment is similar to that observed during the validation procedure for the simulator. (See Appendix C for the BERs obtained at various carrier frequencies and symbol rates.)

The best-case scenario would have been to test the Softwater Modem in the Hudson River estuary to see how well it works in a time-variant channel. The Hudson experiment would have tested the modem's implementation of frame synchronization, FSK detection, and channel equalization. However, since funding dissipated, an alternate test had to be devised.

The alternate test was to determine how the modem performed in an AWGN channel. In order to compare the actual performance with the theoretical limit, the BER of modem was plotted for various bit-normalized SNR, or E_b/N_0, values. E_b/N_0 is defined as energy per bit to noise power spectral density ratio. As E_b/N_0 increases, the probability of a bit error decreases at a rate that produces a curve having a waterfall-like shape (when the x-axis is in dB scale). The solid lines in Figure 4-11 show the theoretical relationship between BER and E_b/N_0 for noncoherent FSK detection. Note that in this test, other factors such as fading, ISI from multipath propagation, and impulsive, possibly colored noise are not taken into consideration. Thus, computing the actual BER versus E_b/N_0 serves as a simple, controlled means of validating the receiver's implementation.

The procedure for estimating the BER versus E_b/N_0 was to attach a 12" 3.5 mm cable from the headphone output of the T60p laptop to the microphone input of the T500 and repeatedly transmit a series of packets at increasing output levels. Anytime the T500 was used as the transmitter, the noise floor rose by approximately 10 dB. When using the T60p as the transmitter, the noise floor remained relatively constant (within 2 dB). The computer's noise followed a normal distribution and had no peaks in the transmission band, providing an approximation of how the modem would perform in an AWGN channel.

To estimate SNR, the PSD of both noise and the modulated signal was computed. Upon figuring out which array elements correspond to frequencies in the modulated signal's bandwidth, the dB level for both noise and signal in only the relevant frequency band was averaged, as in Equation (3.2), and the difference of the averages provided the SNR value. The relationship between SNR and E_b/N_0 is defined as

$$E_b/N_0 = SNR \times (W/R), \tag{4.3}$$

where W is the bandwidth of the modulated signal and R is the bit rate. Since the ratio W/R is always 2 for orthogonal noncoherent FSK signaling with minimum tone spacing, E_b/N_0 turns out to be 3 dB greater than the computed SNR.

Table 4-2: Performance test results for binary FSK.

E_b/N_0	BER	Bits Sent	Error Bits
4.13	0.337	103,600	34,926
4.66	0.296	103,600	30,677
5.32	0.162	103,600	16,742
6.76	0.0777	103,600	8,045
8.52	0.0174	103,600	1,807
10.40	0.00717	518,000	3,715
11.51	0.00289	518,000	1,498
12.74	0.000251	518,000	130
14.27	2.220E-5	1,036,000	23
15.47	1.931E-6	1,036,000	2

Table 4-3: Performance test results for 4-FSK.

E_b/N_0	BER	Bits Sent	Error Bits
4.95	0.286	103,600	29,652
6.16	0.169	103,600	17,492
7.38	0.0533	103,600	5,521
9.25	0.00829	207,200	1,718
11.44	5.598E-4	414,400	232
12.67	1.081E-4	1,036,000	112
14.01	2.684E-5	1,554,000	41
15.43	1.689E-5	2,072,000	35
16.76	4.826E-6	2,072,000	10

Binary FSK was used at 2 kbps, with tones at 10 and 12 kHz. The relevant band covered 9-13 kHz. Inverse filtering was disabled. Data frames contained 2072 bits. The threshold for detection was increased as SNR increased in order to prevent false frame detections. The test was repeated with 4-FSK, again at 2 kbps and in the same frequency band, with tones at 9.5, 10.5, 11.5 and 12.5 kHz. Tables 4-2 and 4-3 list the observed BER vs. E_b/N_0 for binary and 4-FSK, respectively, while showing exactly how many bits were transmitted and how many were received erroneously. The performance test results for both modulation techniques are similar, though up to 2,072,000 bits were transmitted at higher signal strengths when testing 4-FSK. It is acknowledged that the confidence intervals for the lower BERs are very wide, since obtaining tight intervals would take on the order of days at the modem's slow bit rate. These estimates hopefully still offer some insight into the expected performance of the Softwater Modem's implementation of binary and 4-FSK compared to the theoretical limit. As seen in Figure 4-11, binary FSK closely approximates the theoretical curve while the performance of 4-FSK is a bit lacking, especially with higher signal strengths. With 4-FSK, it is believed that the system is approaching the distortion limit of the channel, which makes it harder to distinguish closer spaced tones than further spaced tones.

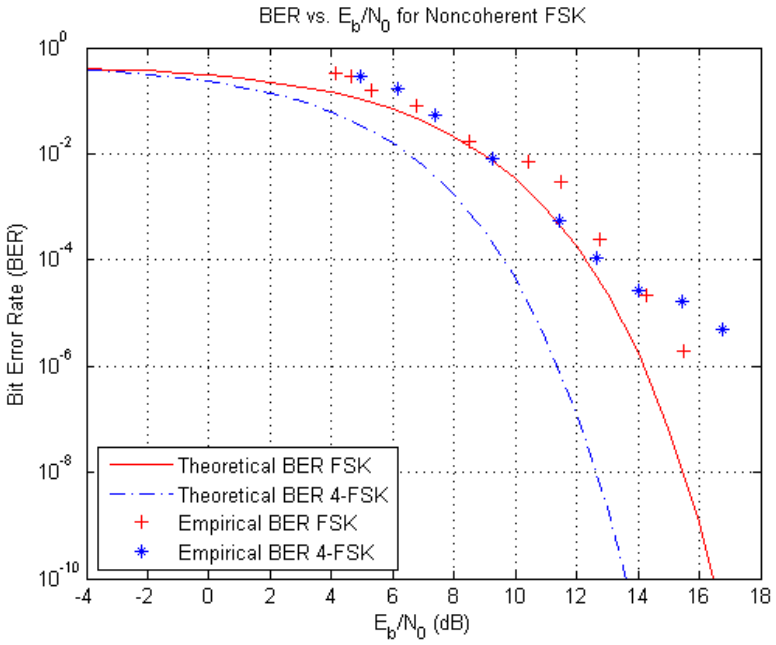

Figure 4-11: Empirical and theoretical BER vs. E_b/N_0.

4.10 Limitations

It is well known the TCP/IP is far from ideal for underwater acoustic communication [Akyildiz 2006]. The additional overhead of relatively large headers increases the transmission time of a packet. More importantly, packet retransmissions and automatic acknowledgment are problematic, even in the simplest case of point-to-point communication between two nodes. For example, the TCP standard sets the initial round-trip time to 3 seconds. Given the slow data rates and long propagation delays of underwater acoustic systems, 3 seconds is not enough time to establish a connection. Therefore, the client begins retransmitting SYN packets before the server's SYN+ACK even reaches the client. Since the mechanism for changing the initial round trip time (irtt) is broken in the Linux kernel, the Softwater Modem has trouble working with TCP/IP; it almost appears as though a SYN flood attack is taking place. Thus, UDP, a simpler transport protocol that uses a smaller header and does not bother with packet retransmissions, is preferred for demonstrating the Softwater Modem's capabilities.

4.11 Future Work

One direction for future work is to augment the repertoire of modulation and signal processing techniques, currently limited to noncoherent binary and 4-FSK based on envelope detection. While the simulator was described before the Softwater Modem in this dissertation for greater readability, its development actually succeeded that of the modem. In fact, the modem was really the author's first step into building a working PHY layer implementation. So, there are many places for improvement, starting with the addition of the methods currently implemented in the simulator, such as the more computationally efficient quadrature receiver for FSK detection, hard limiter for correcting disparities between amplitudes of different tones, and correlation receiver for PSK. It would also be beneficial to develop a LMS-based adaptive DFE for PSK reception.

Another logical extension is to build a feedback mechanism that can update the modem's parameters on the fly as a function of observations of changing channel conditions. As stated earlier, the original goal was to develop a handshaking protocol on a control channel that propagates information about the modem's optimal configuration to other nodes in the network. After adding more modulation and signal processing techniques, development of these adaptive, "cognitive" features can commence.

Because of its LFM chirp mechanism and ability to record frames and save impulse response estimates to file, the modem can serve to some extent as a scientific instrument. Accordingly, it can be used to perform long-term SNR and impulse response capture experiments in interesting bodies of water in order to learn more about the effects of inconstant phenomena such as tides, weather, noise, and wakes on acoustic communication performance. Furthermore, these impulse response estimates can be used as input for the simulator.

Chapter 5
Summary

5.1 Evaluation of Thesis

The work in this dissertation is based on channel estimation. The thesis states that channel estimation techniques can be employed in both a network simulator and software modem to quantify the channel-induced distortion of acoustic signals and thereby to improve the quality of simulation and adaptability of modulation and demodulation, respectively. This thesis has been shown to be true.

The impulse response measurements taken in the Hudson River estuary serve as the input to the network simulation so that BERs are no longer based on simple assumptions about the channel. For time-invariant channels, the BER of a packet computed by the simulator matches within a few percent of the BER of a packet that is actually transmitted through a test environment physical channel.

The software modem computes the channel's impulse response at the beginning of a packet, inverts it, and convolves it with the rest of the waveform to mitigate ISI, irregularities in the frequency response, and other signal distortions imposed by the channel. As a result, in shallow-water channels with properties that change relatively slowly, the modem is able to sense and adapt to the environment and obtain data rates that would otherwise have resulted in numerous bit errors because of multipath propagation.

5.2 Contributions

The work presented in this dissertation covers three separate but related areas of underwater acoustic communication, namely channel characterization, network simulation, and software defined radio. The following contributions have been made to the research community:

1) Characterization of the Hudson River estuary at link distances of 200 and 505 meters. Analysis is highly detailed and includes the scattering function, multipath intensity profile, spaced-frequency correlation function, Doppler power spectrum, spaced-time correlation function, and fading distribution. It is clear that the Hudson is a multipath fading channel with a short coherence time.
2) The procedure for carrying out channel characterization has been described in great detail, possibly more thoroughly than in any other work. The importance of pre-experiment analysis of sounding signal properties and the bounds of a sounding signal's length have been made clear. Moreover, for each of the characterization functions, the formula has been translated into code, making the connection between textbook descriptions and practical implementation easy to follow.
3) A network simulation written in OMNeT++ and MATLAB that simulates the underwater acoustic channel as well as the PHY layer of a network stack. This simulation provides more accurate BERs than previous attempts that use SNR as the only input parameter. The software's modular design makes it easy to add different types of receivers for comparison purposes. Furthermore, the simulation technique affords the opportunity to simulate any body of water if the measurements are available for its input. This capability will make it easy to compare the effects of different channels on a communication system. Similarly, the simulation facilitates the study of choosing the best communication technique for a particular environment.
4) A software-driven binary and 4-FSK modem with a zero-forcing equalizer and Reed-Solomon codes. This modem offers many configurable parameters, including carrier frequency, symbol rate, and guard time, that enable it to work well in a variety of environ-

ments. It interfaces with the Linux TUN driver to enable unmodified network applications to run over an acoustic link.

5) Software for the channel characterization, network simulation, and software modem, key parts of which are printed in the appendix. The code will help others understand how to translate abstract formulas and vague block diagrams found in many textbooks into working systems.

Appendix A
Source Code for Chapter 2

A.1 Chirp Signal Generation

```
function [y, time] = chirp(startFreq, endFreq, samplingRate, numSeconds, me-
thod)
%CHIRP generates an ascending chirp signal.
%   Y = CHIRP(startFreq, endFreq, samplingRate, numSeconds)
%   where startFreq and endFreq (endFreq > startFreq) are the starting and
%   ending frequencies specified in Hz, samplingRate is the sampling rate
%   in Hz of the signal to be generated, and numSeconds is the duration of
%   the signal in seconds, returns a linear frequency modulated complex-
%   valued chirp signal.
%
%   Y = CHIRP(startFreq, endFreq, samplingRate, numSeconds, method)
%   specifies alternate chirp methods.
%   Available methods are 'linear' and 'hyperbolic'. The default is
%   'linear'.
%
%   [Y, TIME] = CHIRP(...) returns a vector of time indices in seconds
%   (TIME).

    if (nargin < 4)
        error('Too few arguments specified in function.');
    end

    if (nargin > 5)
        error('Too many arguments specified in function.');
    end

    if (nargin == 4)
        method = 'linear';
    end

    time = 1/samplingRate : 1/samplingRate : numSeconds;
    if (strcmp(method, 'linear') == 1)
        beta = (endFreq - startFreq) / numSeconds;
        y = exp(1j * (2 * pi * (0.5 * beta * time.^2 + startFreq * time)));
    elseif (strcmp(method, 'hyperbolic') == 1)
        k = (endFreq - startFreq) / (endFreq * numSeconds);
        a = -2 * pi * startFreq / k;
        y = exp(1j * (a * log(1 - k * time)));
    else
        error('Unknown chirp type %s.', method);
        return;
    end
```

A.2 Comparison of Autocorrelation Function of Various Sounding Signals

```
% Author: Brian Borowski
% Created: 01/18/2008
% Last modified: 01/27/2008
% Compares autocorrelation function of several sounding signals.
```

```
%% Start with a clean slate.
clc;
clear all;
close all;

%% Initialization.
carrierFreq = 12000;

% For BPSK/DSSS:
% - symbolsPerSecond must evenly divide samplingRate and carrierFreq.
% - The bandwidth of the main lobe will be (2 x symbolsPerSecond) Hz,
%   centered on carrierFreq.
symbolsPerSecond = 12000;
samplingRate = 48000;
samplesPerSymbol = samplingRate/symbolsPerSecond;

load mse_ao_m511;
sequence = mse_ao_m511(1,:);

%% Generate signals.
hfmChirp = real(chirp(5000, 20000, samplingRate, 0.05, 'hyperbolic'));
lfmChirp = real(chirp(5000, 20000, samplingRate, 0.05, 'linear'));

lengthOfSequence = length(sequence);
totalSamples = samplesPerSymbol * lengthOfSequence;
t = 1:totalSamples;
carrier = cos(2.0 * pi * carrierFreq * t / samplingRate);
bpskSignal = zeros(1, totalSamples);
pos = 0;
index = 1;
for i = 1:totalSamples
    bpskSignal(i) = carrier(i) * sequence(index);
    pos = pos + 1;
    if (pos == samplesPerSymbol)
        pos = 0;
        index = index + 1;
    end
end

whiteNoise = wgn(1, 0.05 * samplingRate, 1);

% Use Zero-Pole-Gain design to filter white noise.
Wn = [5000/(samplingRate/2) 20000/(samplingRate/2)];
[z, p, k] = butter(10, Wn, 'bandpass');
[sos, g] = zp2sos(z, p, k);
Hd = dfilt.df2sos(sos, g);
filteredWhiteNoise = filter(Hd, whiteNoise);

%% Plot autocorrelation functions.
figure;
[auto, lags] = xcorr(hfmChirp, 'coeff');
time = lags / samplingRate * 1000;
plot(time, auto);
xlim([-2 2]);
ylim([-1.2 1.2]);
seconds = length(hfmChirp) / samplingRate;
if (seconds < 1)
    signalLengthStr = sprintf('%.1f ms', seconds * 1000);
else
```

```
    signalLengthStr = sprintf('%.1f s', seconds);
end
graphTitle = sprintf('Autocorrelation of HFM Chirp 5-20 kHz, %s', ...
                     signalLengthStr);
title(graphTitle, 'FontWeight', 'bold');
xlabel('Delay (ms)');
ylabel('Correlation Coefficient');

figure;
[auto, lags] = xcorr(lfmChirp, 'coeff');
time = lags / samplingRate * 1000;
plot(time, auto);
xlim([-2 2]);
ylim([-1.2 1.2]);
seconds = length(lfmChirp) / samplingRate;
if (seconds < 1)
    signalLengthStr = sprintf('%.1f ms', seconds * 1000);
else
    signalLengthStr = sprintf('%.1f s', seconds);
end
graphTitle = sprintf('Autocorrelation of LFM Chirp 5-20 kHz, %s', ...
                     signalLengthStr);
title(graphTitle, 'FontWeight', 'bold');
xlabel('Delay (ms)');
ylabel('Correlation Coefficient');

figure;
[auto, lags] = xcorr(bpskSignal, 'coeff');
time = lags / samplingRate * 1000;
plot(time, auto);
xlim([-2 2]);
ylim([-1.2 1.2]);
seconds = length(bpskSignal) / samplingRate;
if (seconds < 1)
    signalLengthStr = sprintf('%.1f ms', seconds * 1000);
else
    signalLengthStr = sprintf('%.1f s', seconds);
end
graphTitle = sprintf('Autocorrelation of DSSS/BPSK Signal, %s', ...
                     signalLengthStr);
title(graphTitle, 'FontWeight', 'bold');
xlabel('Delay (ms)');
ylabel('Correlation Coefficient');

figure;
[auto, lags] = xcorr(filteredWhiteNoise, 'coeff');
time = lags / samplingRate * 1000;
plot(time, auto);
xlim([-2 2]);
ylim([-1.2 1.2]);
seconds = length(filteredWhiteNoise) / samplingRate;
if (seconds < 1)
    signalLengthStr = sprintf('%.1f ms', seconds * 1000);
else
    signalLengthStr = sprintf('%.1f s', seconds);
end
graphTitle = ...
   sprintf('Autocorrelation of Bandpass Filtered White Noise 5-20 kHz, %s', ...
           signalLengthStr);
title(graphTitle, 'FontWeight', 'bold');
```

```
xlabel('Delay (ms)');
ylabel('Correlation Coefficient');

figure;
[auto, lags] = xcorr(whiteNoise, 'coeff');
time = lags / samplingRate * 1000;
plot(time, auto);
xlim([-2 2]);
ylim([-1.2 1.2]);
seconds = length(whiteNoise) / samplingRate;
if (seconds < 1)
    signalLengthStr = sprintf('%.1f ms', seconds * 1000);
else
    signalLengthStr = sprintf('%.1f s', seconds);
end
graphTitle = sprintf('Autocorrelation of White Noise, %s', signalLengthStr);
title(graphTitle, 'FontWeight', 'bold');
xlabel('Delay (ms)');
ylabel('Correlation Coefficient');

%% Compute and plot PSD of each sounding signal.
h = spectrum.welch;
hfmHpsd = psd(h, hfmChirp, 'Fs', samplingRate);
lfmHpsd = psd(h, lfmChirp, 'Fs', samplingRate);
bpskHpsd = psd(h, bpskSignal, 'Fs', samplingRate);
filteredWhiteNoiseHpsd = psd(h, filteredWhiteNoise, 'Fs', samplingRate);
whiteNoiseHpsd = psd(h, whiteNoise, 'Fs', samplingRate);

maxVal1 = max(max(hfmHpsd.Data), max(lfmHpsd.Data));
maxVal2 = max(max(bpskHpsd.Data), max(whiteNoiseHpsd.Data));
maxVal = max(maxVal1, maxVal2);

figure;
hold on;
plot(hfmHpsd.Frequencies/1000, pow2db(hfmHpsd.Data / maxVal), ...
     'Color', [1 0 0]);
plot(lfmHpsd.Frequencies/1000, pow2db(lfmHpsd.Data / maxVal), '--', ...
     'Color', [0 0.5 0]);
plot(bpskHpsd.Frequencies/1000, pow2db(bpskHpsd.Data / maxVal), ':', ...
     'Color', [0 0 1], 'LineWidth', 2);
plot(filteredWhiteNoiseHpsd.Frequencies/1000, ...
     pow2db(filteredWhiteNoiseHpsd.Data / maxVal), '-.', 'Color', [1 0.5 1]);
plot(whiteNoiseHpsd.Frequencies/1000, ...
     pow2db(whiteNoiseHpsd.Data / maxVal), 'Color', [0 0 0]);
hold off;

legend('HFM Chirp, 5-20 kHz', ...
       'LFM Chirp, 5-20 kHz', ...
       'DSSS/BPSK, 12 kHz Carrier', ...
       'Bandpass Filtered White Noise, 5-20 kHz', ...
       'White Noise');
grid on;
title('Welch Power Spectral Density Estimate', 'FontWeight', 'bold');
xlim([0 samplingRate/1000/2]);
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
```

A.3 Comparison of Autocorrelation Function of White Noise Signals of Various Lengths

```
% Author: Brian Borowski
% Created: 02/28/2010
% Last modified: 02/28/2010
% Compares the autocorrelation function of two white noise signals, with
% one being significantly longer than the other.

%% Start with a clean slate.
clc;
clear all;
close all;

%% Generate the white noise signals.
samplingRate = 48000;
lengthMsShort = 10;
shortSignal = rand(1, lengthMsShort * samplingRate / 1000) - 0.5;
lengthMsLong = 1000;
longSignal = rand(1, lengthMsLong * samplingRate / 1000) - 0.5;

%% Plot the autocorrelation functions.
[auto, lags] = xcorr(shortSignal, 'coeff');
time = lags / samplingRate * 1000;
figure;
plot(time, auto, 'r');
[auto, lags] = xcorr(longSignal, 'coeff');
time = lags / samplingRate * 1000;
hold on;
plot(time, auto, 'b');
xlim([-5 5]);
ylim([-1.2 1.2]);
title('Autocorrelation of White Noise', 'FontWeight', 'bold');
xlabel('Delay (ms)');
ylabel('Correlation Coefficient');
s1 = sprintf('%d ms', lengthMsShort);
s2 = sprintf('%d ms', lengthMsLong);
legend(s1, s2);
```

A.4 Channel Characterization

```
% Author: Brian Borowski
% Created: 07/23/2008
% Last modified: 02/02/2010
% Performs channel characterization.

%% Start with a clean slate.
clc;
clear all;
close all;

%% Process the recorded signal.
recorded_signal_file = 'Recordings/SoundingSignal.wav';
[recordedSignal, samplingRate] = wavread(recorded_signal_file);
totalSamples = length(recordedSignal);
recordedSeconds = totalSamples / samplingRate;
```

```
nyquistFreq = samplingRate / 2;

%% Process the reference signal.
referenceSeconds = 0.05;
referenceSamples = referenceSeconds * samplingRate;
chirpStartFreq = 0;
chirpEndFreq = 24000;
referenceSignal = chirp(chirpStartFreq, chirpEndFreq, ...
                        samplingRate, referenceSeconds);

%% Plot the autocorrelation of the reference signal.
[auto, lags] = xcorr(referenceSignal, 'coeff');
time = lags / samplingRate * 1000;
figure;
plot(time, real(auto));
axis([-2 2 -1.2 1.2]);
s = sprintf('Autocorrelation of LFM Chirp %d-%d kHz, %.1f ms', ...
            chirpStartFreq / 1000, chirpEndFreq / 1000, ...
            referenceSeconds * 1000);
title(s, 'FontWeight', 'bold');
xlabel('Delay (ms)');
ylabel('Correlation Coefficient');

%% Plot the envelope of autocorrelation of the reference signal in dB.
figure;
plot(time, mag2db(abs(auto)));
axis([-5 5 -60 0]);
s = sprintf('Envelope of Autocorrelation of LFM Chirp %d-%d kHz, %.1f ms', ...
            chirpStartFreq / 1000, chirpEndFreq / 1000, ...
            referenceSeconds * 1000);
title(s, 'FontWeight', 'bold');
xlabel('Delay (ms)');
ylabel('Magnitude (dB)');

%% Calculate impulse response over time.
numOfImpulseResponses = min(recordedSeconds / referenceSeconds, 101);
if (numOfImpulseResponses == 101)
    recordedSeconds = 5;
end

seconds = 0.011;
len = seconds * samplingRate;
if (mod(len, 2) == 0)
    len = len + 1;
end
impulseResponse = zeros(numOfImpulseResponses, len);
for i = 1:numOfImpulseResponses
    snip = recordedSignal((i-1)*referenceSamples+1:i*referenceSamples);
    temp = fftshift(xcorr(snip, conj(referenceSignal)));
    impulseResponse(i,:) = temp(1:len);
end

[maxVal maxIndex] = max(max(abs(impulseResponse)));
impulseResponse = impulseResponse / maxVal;

%% Plot magnitude levels of main component of impulse response.
magnitude = abs(impulseResponse(:, maxIndex));
time = (0:length(magnitude) - 1) / (1 / referenceSeconds);

figure;
```

```
plot(time, magnitude);
title('Magnitude of Strongest Impulse Response Tap', ...
      'FontWeight', 'bold');
xlabel('Time (s)');
ylabel('Magnitude');

disp('Magnitude of Strongest Impulse Response Tap');
s = sprintf('       Max: %.4f', max(magnitude));
disp(s);
s = sprintf('       Min: %.4f', min(magnitude));
disp(s);
s = sprintf('      Mean: %.4f', mean(magnitude));
disp(s);
s = sprintf('   Std Dev: %.4f', std(magnitude));
disp(s);

%% Plot impulse response over time.
len = length(impulseResponse(1,:));
oneMs = 0.001 * samplingRate;

figure;
upperBound =
int32(numOfImpulseResponses*referenceSeconds*(1.0/referenceSeconds));
imagesc([-1 (length(impulseResponse(1,:))-oneMs)/(samplingRate/1000)], ...
        [0 length(impulseResponse(:,1))/(1.0/referenceSeconds)], ...
        abs(impulseResponse(1:upperBound, 1:len)));
set(gca, 'ydir', 'normal');
title('Impulse Response {\itc}(\tau; {\itt})', 'FontWeight', 'bold');
xlabel('[\tau] Delay (ms)');
ylabel('[t] Time (s)');
zlabel('Normalized Intensity');
colorbar;

%% Plot single impulse response.
ir = real(impulseResponse(1,:));
[maxValue mainPeakIndex] = max(ir);
earlyPeakIndex = find(ir >= 0.25 * maxValue);
offset = mainPeakIndex - earlyPeakIndex;
ir = ir(earlyPeakIndex:end);
ir = ir / max(abs(ir));
figure;
n = (1:length(ir)) * 1000 / samplingRate;
plot(n, ir);
axis([0 10 -1 1]);
title('Impulse Response of Tub', 'FontWeight', 'bold');
xlabel('[\tau] Delay (ms)');
ylabel('Normalized Amplitude');

%% Plot frequency and phase response.
[h w] = freqz(ir, 1);
f = w / (2 * pi) * samplingRate / 1000;
magnitude = abs(h);
stdDev = mag2db(std(magnitude));

decibel = mag2db(magnitude);
len = length(magnitude);
avgDecibel = 10 * log10(sum(10.^(decibel/10))/len);
decibel = decibel - avgDecibel;

disp('Frequency Response');
```

```
s = sprintf('       Max: %.4f dB', max(decibel));
disp(s);
s = sprintf('       Min: %.4f dB', min(decibel));
disp(s);
s = sprintf('      Mean: %.4f dB', 10 * log10(sum(10.^(decibel/10))/len));
disp(s);
s = sprintf('   Std Dev: %.4f dB', stdDev);
disp(s);

figure;
subplot(2, 1, 1),
plot(f, decibel),
axis([0 f(end) -40 20]),
title('Frequency Response', 'FontWeight', 'bold'),
xlabel('Frequency (kHz)'),
ylabel('Magnitude (dB)');

phase = angle(h) * 180 / pi;
subplot(2, 1, 2),
plot(f, phase),
xlim([0 f(end)]),
title('Phase Response', 'FontWeight', 'bold'),
xlabel('Frequency (kHz)'),
ylabel('Phase (degrees)');

%% Plot a single cross-section of the scattering function, at the
 % time delay with the greatest magnitude.
 % Do not attempt to reduce side lobes with windowing or by zeroing out
 % values.
crossSection = fftshift(fft(xcorr(impulseResponse(:,maxIndex))));
maxVal = max(crossSection);
crossSection = crossSection / maxVal;

lambdaSamples = length(crossSection);
lowerBound = floor(lambdaSamples / 2);
upperBound = floor(lambdaSamples / 2);
if (mod(lambdaSamples, 2) == 0)
    upperBound = upperBound - 1;
end
f = -lowerBound:upperBound;
lambda = (1 / referenceSeconds / 2) * f / lowerBound;

figure;
plot(lambda, abs(crossSection));
title('Doppler Power Spectrum', 'FontWeight', 'bold');
xlabel('[\lambda] Frequency (Hz)');
ylabel('Normalized Intensity');

figure;
plot(lambda, pow2db(abs(crossSection)));
title('Doppler Power Spectrum', 'FontWeight', 'bold');
xlabel('[\lambda] Frequency (Hz)');
ylabel('Normalized Intensity (dB)');

%% Compute scattering function.
tauSamples = length(impulseResponse);
lambdaSamples = numOfImpulseResponses;
clear temp;
scatteringFunction = zeros(tauSamples, 2 * lambdaSamples - 1);
for i = 1:tauSamples
```

```
    temp = fftshift(fft(xcorr(impulseResponse(:,i))));
    scatteringFunction(i,:) = temp(end:-1:1);
end

maxVal = max(max(abs(scatteringFunction)));
scatteringFunction = scatteringFunction / maxVal;
scatteringFunction(scatteringFunction < 0.01) = 0;

%% Plot scattering function.
[tauSamples lambdaSamples] = size(scatteringFunction(1:end,:));
tau = (0:tauSamples-1) / samplingRate * 1000 - 1;
lowerBound = floor(lambdaSamples / 2);
upperBound = floor(lambdaSamples / 2);
if (mod(lambdaSamples, 2) == 0)
    upperBound = upperBound - 1;
end
f = -lowerBound:upperBound;
lambda = (1 / referenceSeconds / 2) * f / lowerBound;

figure;
[X, Y] = meshgrid(tau, lambda);
contour(X, Y, (abs(scatteringFunction(1:end,:)))');
title('Scattering Function', 'FontWeight', 'bold'),
xlabel('[\tau] Delay (ms)'),
ylabel('[\lambda] Frequency (Hz)'),
zlabel('Normalized Intensity'),
axis([-1 5 -2 2]);
colorbar;

%% Plot multipath intensity profile and compute delay spreads.
mip = sum(abs(scatteringFunction'));
mip = mip / max(mip);

len = length(mip);
tau = (0:len-1) * 1000/samplingRate - 1;
figure;
plot(tau, mip);
xlim([-1 10]);
title('Multipath Intensity Profile', 'FontWeight', 'bold');
xlabel('[\tau] Delay (ms)');
ylabel('Normalized Intensity');

maximumExcessDelay20 = find(mip >= 0.01); % 20 dB
positiveMIP = mip(maximumExcessDelay20(1):maximumExcessDelay20(end));
positiveTau = tau(maximumExcessDelay20(1):maximumExcessDelay20(end));
sumValue = sum(positiveMIP);
meanDelay = sum(positiveTau .* positiveMIP) / sumValue;
rmsDelaySpread = sqrt( ...
    sum((positiveTau - meanDelay).^2 .* positiveMIP) / sumValue);
disp('Delay Spread');
s = sprintf('      Mean excess delay: %.4f ms', meanDelay);
disp(s);
s = sprintf('       RMS delay spread: %.4f ms', rmsDelaySpread);
disp(s);

maximumExcessDelay10 = find(mip >= 0.1); % 10 dB
maximumExcessDelayMs = (maximumExcessDelay10(end) - ...
                        maximumExcessDelay10(1)) / samplingRate * 1000;
s = sprintf('   Maximum excess delay: %.4f ms', maximumExcessDelayMs);
disp(s);
```

```
%% Plot spaced-frequency correlation function.
sfcf = abs(fftshift(fft(mip)));
sfcf = sfcf / max(sfcf);

lenlambda = length(sfcf);
lowerBound = floor(lenlambda / 2);
upperBound = floor(lenlambda / 2);
if (mod(lenlambda, 2) == 0)
    upperBound = upperBound - 1;
end
f = -lowerBound:upperBound;
freq = nyquistFreq / 2 * f / lowerBound / 1000;
figure;
plot(freq, sfcf);
title('Spaced-Frequency Correlation Function', 'FontWeight', 'bold');
xlabel('\Delta f (kHz)');
ylabel('|{\itR_C}(\Delta f)|');
xlim([freq(1) freq(end)]);

%% Convert spaced-frequency correlation function to dB and calculate the
%  -3, -6 and -10 dB coherence bandwidths.
sfcf = pow2db(sfcf);
sfcf = sfcf - max(sfcf);

figure;
plot(freq, sfcf);
title('Spaced-Frequency Correlation Function', 'FontWeight', 'bold');
xlabel('\Delta f (kHz)');
ylabel('|{\itR_C}(\Delta f)| (dB)');
xlim([freq(1) freq(end)]);

midpoint = floor(length(sfcf)/2);
resultLeft = find(sfcf(1:midpoint) < -3);
resultRight = find(sfcf(midpoint+1:end) < -3);
if (not(isempty(resultLeft)) && not(isempty(resultRight)))
    minusLeft = (resultLeft(end)+1)/lenlambda * nyquistFreq;
    minusRight = (resultRight(1)+midpoint-1)/lenlambda * nyquistFreq;
    coherenceBandwidth3 = round(minusRight - minusLeft);
else
    coherenceBandwidth3 = nyquistFreq;
end

resultLeft = find(sfcf(1:midpoint) < -6);
resultRight = find(sfcf(midpoint+1:end) < -6);
if (not(isempty(resultLeft)) && not(isempty(resultRight)))
    minusLeft = (resultLeft(end)+1)/lenlambda * nyquistFreq;
    minusRight = (resultRight(1)+midpoint-1)/lenlambda * nyquistFreq;
    coherenceBandwidth6 = round(minusRight - minusLeft);
else
    coherenceBandwidth6 = nyquistFreq;
end

resultLeft = find(sfcf(1:midpoint) < -10);
resultRight = find(sfcf(midpoint+1:end) < -10);
if (not(isempty(resultLeft)) && not(isempty(resultRight)))
    minusLeft = (resultLeft(end)+1)/lenlambda * nyquistFreq;
    minusRight = (resultRight(1)+midpoint-1)/lenlambda * nyquistFreq;
    coherenceBandwidth10 = round(minusRight - minusLeft);
else
```

```
    coherenceBandwidth10 = nyquistFreq;
end

disp('Coherence Bandwidth');
s = sprintf('   -3 dB: %d Hz', coherenceBandwidth3);
disp(s);
s = sprintf('   -6 dB: %d Hz', coherenceBandwidth6);
disp(s);
s = sprintf('  -10 dB: %d Hz', coherenceBandwidth10);
disp(s);

%% Compute Doppler shift and spread.
dps = sum(abs(scatteringFunction));

numPoints = length(dps);
lowerBound = floor(numPoints / 2);
upperBound = lowerBound;
if (mod(lambdaSamples, 2) == 0)
    upperBound = upperBound - 1;
end
f = -lowerBound:upperBound;
lambda = (1 / referenceSeconds / 2) * f / lowerBound;

sumValue = sum(dps);
overallShift = sum(lambda .* dps) / sumValue;
overallSpread = sqrt(sum((lambda - overallShift).^2 .* dps) / sumValue);

disp('Doppler Power Spectrum');
s = sprintf('    Doppler Shift: %.4f Hz', overallShift);
disp(s);
s = sprintf('   Doppler Spread: %.4f Hz', overallSpread);
disp(s);

%% Plot Doppler power spectrum.
dps = dps / max(dps);

figure;
plot(lambda, dps);
title('Doppler Power Spectrum', 'FontWeight', 'bold');
xlabel('[\lambda] Frequency (Hz)');
ylabel('Normalized Intensity');

%% Plot spaced-time correlation function and calculate coherence times for
 % correlations of 0.5, 0.25, and 0.1.
stcf = abs(fftshift(fft(dps)));
stcf = stcf / max(stcf);

lentime = length(stcf);
lowerBound = floor(lentime / 2);
upperBound = floor(lentime / 2);
if (mod(lentime, 2) == 0)
    upperBound = upperBound - 1;
end
t = -lowerBound:upperBound;
time = recordedSeconds / 2 * t / lowerBound;

midpoint = floor(length(stcf)/2);
resultLeft = find(stcf(1:midpoint) < 0.5);
resultRight = find(stcf(midpoint+1:end) < 0.5);
if (not(isempty(resultLeft)) && not(isempty(resultRight)))
```

```
    minusLeft = (resultLeft(end)+1)/lentime * recordedSeconds;
    minusRight = (resultRight(1)+midpoint-1)/lentime * recordedSeconds;
    coherenceTime3 = minusRight - minusLeft;
else
    coherenceTime3 = recordedSeconds;
end

resultLeft = find(stcf(1:midpoint) < 0.25);
resultRight = find(stcf(midpoint+1:end) < 0.25);
if (not(isempty(resultLeft)) && not(isempty(resultRight)))
    minusLeft = (resultLeft(end)+1)/lentime * recordedSeconds;
    minusRight = (resultRight(1)+midpoint-1)/lentime * recordedSeconds;
    coherenceTime6 = minusRight - minusLeft;
else
    coherenceTime6 = recordedSeconds;
end

resultLeft = find(stcf(1:midpoint) < 0.1);
resultRight = find(stcf(midpoint+1:end) < 0.1);
if (not(isempty(resultLeft)) && not(isempty(resultRight)))
    minusLeft = (resultLeft(end)+1)/lentime * recordedSeconds;
    minusRight = (resultRight(1)+midpoint-1)/lentime * recordedSeconds;
    coherenceTime10 = minusRight - minusLeft;
else
    coherenceTime10 = recordedSeconds;
end

disp('Coherence Time');
s = sprintf('   -3 dB: %.3f sec', coherenceTime3);
disp(s);
s = sprintf('   -6 dB: %.3f sec', coherenceTime6);
disp(s);
s = sprintf('  -10 dB: %.3f sec', coherenceTime10);
disp(s);

figure;
plot(time, stcf);
title('Spaced-Time Correlation Function', 'FontWeight', 'bold');
xlabel('\Delta t (Seconds)');
ylabel('|{\itR_C}(\Delta t)|');
if (coherenceTime3 == recordedSeconds)
    ylim([0 1.1]);
else
    ylim([0 1]);
end

%% Estimate Doppler shift and spread for strongest components.
startIndex = maximumExcessDelay10(1);
totalPaths = length(maximumExcessDelay10);
scatteringFunction = abs(scatteringFunction(1:end,:));
index = zeros(1, totalPaths);
delayMs = zeros(1, totalPaths);
intensity = zeros(1, totalPaths);
shift = zeros(1, totalPaths);
spread = zeros(1, totalPaths);
avgShift = 0;
avgSpread = 0;

% Compute statistics on the components falling within the maximum excess
% delay (-10 dB of the strongest arrival).
```

```
for i = 1:totalPaths
    rowIndex = maximumExcessDelay10(i);
    intensity(i) = mip(rowIndex);
    index(i) = rowIndex;
    delayMs(i) = (rowIndex - oneMs - 1) / samplingRate * 1000;
    Sband = scatteringFunction(rowIndex,:);
    sumValue = sum(Sband);
    if (sumValue == 0)
        shift(i) = 0;
        spread(i) = 0;
    else
        shift(i) = sum(lambda .* Sband) / sumValue;
        spread(i) = sqrt(sum((lambda - shift(i)).^2 .* Sband) / sumValue);
        avgShift = avgShift + shift(i);
        avgSpread = avgSpread + spread(i);
    end
    scatteringFunction(rowIndex,:) = 0;
end
[s,c] = sort(index, 2, 'ascend');
set = [s', delayMs(c)' intensity(c)' shift(c)' spread(c)'];

s = sprintf('\nStatistics for %d Strongest Components:', totalPaths);
disp(s);
s = sprintf('Index\tDelay(ms)\tIntensity\tShift\t\tSpread');
disp(s);
for i = 1:totalPaths
    s = sprintf('%d\t\t%.3f\t\t%.4f\t\t%.4f\t\t%.4f', ...
                set(i, 1), set(i, 2), set(i, 3), set(i, 4), set(i, 5));
    disp(s);
end
s = sprintf('Average Doppler shift:  %.4f Hz', avgShift/totalPaths);
disp(s);
s = sprintf('Average Doppler spread: %.4f Hz', avgSpread/totalPaths);
disp(s);
```

A.5 Noise Power Spectral Density

```
% Author: Brian Borowski
% Created: 08/03/2009
% Last modified: 02/02/2010
% Computes and plots the PSD of noise in the Hudson River estuary.

%% Start with a clean slate.
clear all;
close all;
clc;

%% Read in the data.
[signal505, samplingRate505] = wavread('Noise505m.wav');
[signal200, samplingRate200] = wavread('Noise200m.wav');

%% Compute PSDs.
h = spectrum.welch('Hann', 256);
Hpsd505 = psd(h, signal505, 'NFFT', 256, 'Fs', samplingRate505);
Hpsd200 = psd(h, signal200, 'NFFT', 256, 'Fs', samplingRate200);

pow505 = pow2db(Hpsd505.Data);
pow200 = pow2db(Hpsd200.Data);
```

```
%% Align to 0 dB level.
if (mean(pow505) > mean(pow200))
    maxVal = max(pow505);
else
    maxVal = max(pow200);
end

pow505 = pow505 - maxVal;
pow200 = pow200 - maxVal;

%% Plot PSDs.
plot(Hpsd505.Frequencies/1000, pow505, 'b');
hold on;
plot(Hpsd200.Frequencies/1000, pow200, 'r--');
hold off;

grid on;
title('Welch Power Spectral Density Estimate', 'FontWeight', 'bold');
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
legend('5:24 P.M.', '6:42 P.M.');
```

A.6 Distribution Fitting of Magnitude Levels in Multipath Arrival

```
% Author: Brian Borowski
% Created: 02/07/2010
% Last modified: 02/18/2010
% Fits magnitude levels to various distributions used to model fading
% channels.

%% Start with a clean slate.
clc;
clear all;
close all;

%% Read in and prepare the data.
magnitudes = csvread('StrongestComponent1.csv');
sortedMagnitudes = sort(magnitudes);

% Beta distribution cannot accept values of 1. Normalize max to 0.99 first.
sortedMagnitudes = sortedMagnitudes / sortedMagnitudes(end) * 0.99;

% Plot the fading envelope.
figure;
seconds = 1:length(magnitudes);
seconds = seconds / 20;
env = mag2db(magnitudes);
env = env - mean(env);
plot(seconds, env);
xlabel('Time (s)');
ylabel('Amplitude (dB)');
title('Fading Envelope of Strongest Impulse Response Tap', ...
      'FontWeight', 'bold');
axis([0 30 -80 20]);

%% Plot the histogram of the measurements.
figure;
```

```
start = (sortedMagnitudes(1));
finish = (sortedMagnitudes(end));
x = linspace(start, finish, 100);
hist(sortedMagnitudes, x);
xlabel('Signal Level');
ylabel('Frequency');
h = get(gca, 'child');
set(h, 'FaceColor', [.98 .98 .98], 'EdgeColor', [.94 .94 .94]);
counts = hist(sortedMagnitudes, x);
hold on;
plot(x, counts, 'o');
hold off;

n = length(sortedMagnitudes);
binWidth = x(2) - x(1);
prob = counts / (n * binWidth);
prob = prob / sum(prob);
legh_ = zeros(1, 7); legt_ = cell(1, 7);   % Handles and text for legend
ax_ = newplot;
set(ax_, 'Box', 'on');
bar(x, prob, 'hist');
h_ = get(gca, 'child');
set(h_, 'FaceColor', [.4 .4 .4], 'EdgeColor', [.4 .4 .4]);
xlabel('Signal Level');
ylabel('Distribution');
title('Probability Distribution Function, Strongest IR Tap', ...
      'FontWeight', 'bold');
legh_(1) = h_;
legt_{1} = 'Measurements';
leginfo_ = {'Orientation', 'vertical', 'Location', 'NorthEast'};

% Use maximum likelihood estimation to fit the data.
alpha = 0.05; % alpha = 0.05 for 95% confidence.
paramEstsRayleigh = raylfit(sortedMagnitudes, alpha);
rayleighEst = raylpdf(x, paramEstsRayleigh(1));

paramEstsRician = mle(sortedMagnitudes, 'dist', 'rician', 'alpha', alpha);
ricianEst = pdf('rician', x, paramEstsRician(1), paramEstsRician(2));
K = paramEstsRician(1)^2 / (2 * paramEstsRician(2)^2);

paramEstsNakagami = mle(sortedMagnitudes, 'dist', 'nakagami', ...
    'alpha', alpha);
nakagamiEst = ...
    pdf('nakagami', x, paramEstsNakagami(1), paramEstsNakagami(2));

paramEstsBeta = betafit(sortedMagnitudes, alpha);
betaEst = betapdf(x, paramEstsBeta(1), paramEstsBeta(2));

paramEstsGamma = gamfit(sortedMagnitudes, alpha);
gammaEst = gampdf(x, paramEstsGamma(1), paramEstsGamma(2));

paramEstsLog = lognfit(sortedMagnitudes, alpha);
logEst = lognpdf(x, paramEstsLog(1), paramEstsLog(2));

rayleighEst = rayleighEst / sum(rayleighEst);
ricianEst = ricianEst / sum(ricianEst);
nakagamiEst = nakagamiEst / sum(nakagamiEst);
betaEst = betaEst / sum(betaEst);
gammaEst = gammaEst / sum(gammaEst);
logEst = logEst / sum(logEst);
```

```
rayleighEst = rayleighEst / sum(rayleighEst);
ricianEst = ricianEst / sum(ricianEst);
nakagamiEst = nakagamiEst / sum(nakagamiEst);
betaEst = betaEst / sum(betaEst);
gammaEst = gammaEst / sum(gammaEst);
logEst = logEst / sum(logEst);

%% Plot the fits.
hold on;
h_ = plot(x, rayleighEst, 'Color', 'red', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(2) = h_;
legt_{2} = 'Rayleigh fit';

h_ = plot(x, ricianEst, 'Color', 'blue', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(3) = h_;
legt_{3} = 'Rician fit';

h_ = plot(x, nakagamiEst, 'Color', 'green', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(4) = h_;
legt_{4} = 'Nakagami fit';

h_ = plot(x, betaEst, 'Color', [1 0.64 0], ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(5) = h_;
legt_{5} = 'Beta fit';

h_ = plot(x, gammaEst, 'Color', 'magenta', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(6) = h_;
legt_{6} = 'Gamma fit';

h_ = plot(x, logEst, 'Color', 'cyan', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(7) = h_;
legt_{7} = 'Lognormal fit';

h_ = legend(ax_, legh_, legt_, leginfo_{:});
set(h_, 'Interpreter', 'none');
hold off;

logTwoData = log2(prob);
for i = 1:length(logTwoData)
    if (isinf(log2(logTwoData(i))))
        logTwoData(i) = 0;
    end
end

%% Test the goodness of the fits with:
%  - Kullback-Leibler divergence
%  - Bhattacharyya distance
```

```
%  - Metric based on the Bhattacharyya coefficient, proposed by Comaniciu,
%    Ramesh, and Meer
KLray = sum(prob .* (logTwoData - log2(rayleighEst)));
KLric = sum(prob .* (logTwoData - log2(ricianEst)));
KLnak = sum(prob .* (logTwoData - log2(nakagamiEst)));
KLbet = sum(prob .* (logTwoData - log2(betaEst)));
KLgam = sum(prob .* (logTwoData - log2(gammaEst)));
KLlog = sum(prob .* (logTwoData - log2(logEst)));

Bray = -log2(sum(sqrt(prob .* rayleighEst)));
Bric = -log2(sum(sqrt(prob .* ricianEst)));
Bnak = -log2(sum(sqrt(prob .* nakagamiEst)));
Bbet = -log2(sum(sqrt(prob .* betaEst)));
Bgam = -log2(sum(sqrt(prob .* gammaEst)));
Blog = -log2(sum(sqrt(prob .* logEst)));

BModray = sqrt(1 - (sum(sqrt(prob .* rayleighEst))));
BModric = sqrt(1 - (sum(sqrt(prob .* ricianEst))));
BModnak = sqrt(1 - (sum(sqrt(prob .* nakagamiEst))));
BModbet = sqrt(1 - (sum(sqrt(prob .* betaEst))));
BModgam = sqrt(1 - (sum(sqrt(prob .* gammaEst))));
BModlog = sqrt(1 - (sum(sqrt(prob .* logEst))));

%% Display the results in a table.
disp('----- Strongest Impulse Response Tap -----');
disp('            K-L     Bhat.   CRM');
s = sprintf('Beta      : %.4f  %.4f  %.4f [alpha = %.4f, beta = %.4f]', ...
    KLbet, Bbet, BModbet, paramEstsBeta(1), paramEstsBeta(2));
disp(s);
s = sprintf('Gamma     : %.4f  %.4f  %.4f [alpha = %.4f, beta = %.4f]', ...
    KLgam, Bgam, BModgam, paramEstsGamma(1), paramEstsGamma(2));
disp(s);
s = sprintf('Lognormal : %.4f  %.4f  %.4f [mu = %.4f, sigma = %.4f]', ...
    KLlog, Blog, BModlog, paramEstsLog(1), paramEstsLog(2));
disp(s);
s = sprintf('Nakagami-m: %.4f  %.4f  %.4f [m = %.4f, omega = %.4f]', ...
    KLnak, Bnak, BModnak, paramEstsNakagami(1), paramEstsNakagami(2));
disp(s);
s = sprintf('Rayleigh  : %.4f  %.4f  %.4f [sigma = %.4f]', KLray, Bray, ...
    BModray, paramEstsRayleigh(1));
disp(s);
s = sprintf(...
    'Rician    : %.4f  %.4f  %.4f [s = %.4f, sigma = %.4f, K = %.4f]', ...
    KLric, Bric, BModric, paramEstsRician(1), paramEstsRician(2), K);
disp(s);
```

A.7 Distribution Fitting of Magnitude Levels in Comb Signal

```
% Author: Brian Borowski
% Created: 07/12/2009
% Last modified: 02/18/2010
% Fits magnitude levels of comb signal to various distributions used to
% model fading channels.

%% Start with a clean slate.
clc;
clear all;
close all;
```

```
%% Initialization.
delta = 1.0;              % Specified in kHz
tones = [35 45 60 75 85]; % Specified in kHz
secondsToProcess = 10;

[data, samplingRate, numBits] = wavread('MultipleTones.wav');
data = data(1:secondsToProcess*samplingRate);
dbEnvelope = zeros(length(tones), length(data));

%% Process each tone separately.
for i = 1:length(tones)
    tone = tones(i);

    n = 10;
    Wn = [(tone-delta)*1000 (tone+delta)*1000] / (samplingRate / 2);
    ftype = 'bandpass';

    % To avoid round-off errors, do not use the transfer function. Instead
    % get the zpk representation and convert it to second-order sections.

    % Zero-Pole-Gain design
    [z, p, k] = butter(n, Wn, ftype);
    [sos, g] = zp2sos(z, p, k);
    Hd = dfilt.df2sos(sos, g);
    filteredData = filter(Hd, data);

    filename = sprintf('%dkHz.wav', tone);
    filteredData = filteredData / max(filteredData) * 0.99;
    wavwrite(filteredData, samplingRate, filename);
    X = hilbert(filteredData);
    Xr = real(X);
    Xi = imag(X);
    magnitudes = sqrt(Xr.^2 + Xi.^2);
    sortedMagnitudes = sort(magnitudes);

    % Beta distribution cannot accept values >= 1. Normalize max to
    % 0.99 first.
    maxVal = sortedMagnitudes(end);
    sortedMagnitudes = sortedMagnitudes / maxVal * 0.99;

    minVal = sortedMagnitudes(1);
    % Beta distribution cannot accept values <= 0. All values are already
    % non-negative. Duplicate first non-negative value in place of 0.
    if (minVal == 0)
        sortedMagnitudes(1) = sortedMagnitudes(2);
    end

    dbEnvelope(i,:) = mag2db(sortedMagnitudes);

    figure;
    seconds = 1:length(magnitudes);
    seconds = seconds / samplingRate;
    env = mag2db(magnitudes);
    env = env - mean(env);
    plot(seconds, env);
    xlabel('Time (s)');
    ylabel('Amplitude (dB)');
    graphTitle = sprintf('Fading Envelope of %d kHz Sinusoid', tone);
    title(graphTitle, 'FontWeight', 'bold');
```

```
ylim([-80 20]);

meanVal = mean(dbEnvelope(i,:));
if (meanVal < 0)
    meanVal = meanVal * -1;
end
dbEnvelope(i,:) = dbEnvelope(i,:) + meanVal;

% Plot the histogram of the measurements.
figure;
start = (sortedMagnitudes(1));
finish = (sortedMagnitudes(end));
x = linspace(start, finish, 100);
hist(sortedMagnitudes, x);
xlabel('Signal Level');
ylabel('Frequency');
h = get(gca,'child');
set(h, 'FaceColor', [.98 .98 .98], 'EdgeColor', [.94 .94 .94]);
counts = hist(sortedMagnitudes, x);
hold on;
plot(x, counts, 'o');
hold off;

n = length(sortedMagnitudes);
binWidth = x(2) - x(1);
prob = counts / (n * binWidth);
prob = prob / sum(prob);
legh_ = zeros(1, 7); legt_ = cell(1, 7);   % Handles and text for legend
ax_ = newplot;
set(ax_, 'Box', 'on');
bar(x, prob, 'hist');
h_ = get(gca, 'child');
set(h_, 'FaceColor', [.4 .4 .4], 'EdgeColor',[.4 .4 .4]);
xlabel('Signal Level');
ylabel('Distribution');
s = sprintf('Probability Distribution Function, %d kHz', tone);
title(s, 'FontWeight', 'bold');
legh_(1) = h_;
legt_{1} = 'Measurements';
leginfo_ = {'Orientation', 'vertical', 'Location', 'NorthEast'};

% Use maximum likelihood estimation to fit the data.
alpha = 0.05; % alpha = 0.05 for 95% confidence.
paramEstsRayleigh = raylfit(sortedMagnitudes, alpha);
rayleighEst = raylpdf(x, paramEstsRayleigh(1));

paramEstsRician = mle(sortedMagnitudes, 'dist', 'rician', 'alpha', alpha);
ricianEst = pdf('rician', x, paramEstsRician(1), paramEstsRician(2));
K = paramEstsRician(1)^2 / (2 * paramEstsRician(2)^2);

paramEstsNakagami = mle(sortedMagnitudes, 'dist', 'nakagami', ...
    'alpha', alpha);
nakagamiEst = ...
    pdf('nakagami', x, paramEstsNakagami(1), paramEstsNakagami(2));

paramEstsBeta = betafit(sortedMagnitudes, alpha);
betaEst = betapdf(x, paramEstsBeta(1), paramEstsBeta(2));

paramEstsGamma = gamfit(sortedMagnitudes, alpha);
gammaEst = gampdf(x, paramEstsGamma(1), paramEstsGamma(2));
```

```
paramEstsLog = lognfit(sortedMagnitudes, alpha);
logEst = lognpdf(x, paramEstsLog(1), paramEstsLog(2));

rayleighEst = rayleighEst / sum(rayleighEst);
ricianEst = ricianEst / sum(ricianEst);
nakagamiEst = nakagamiEst / sum(nakagamiEst);
betaEst = betaEst / sum(betaEst);
gammaEst = gammaEst / sum(gammaEst);
logEst = logEst / sum(logEst);

% Plot the fits.
hold on;
h_ = plot(x, rayleighEst, 'Color', 'red', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(2) = h_;
legt_{2} = 'Rayleigh fit';

h_ = plot(x, ricianEst, 'Color', 'blue', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(3) = h_;
legt_{3} = 'Rician fit';

h_ = plot(x, nakagamiEst, 'Color', 'green', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(4) = h_;
legt_{4} = 'Nakagami fit';

h_ = plot(x, betaEst, 'Color', [1 0.64 0], ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(5) = h_;
legt_{5} = 'Beta fit';

h_ = plot(x, gammaEst, 'Color', 'magenta', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(6) = h_;
legt_{6} = 'Gamma fit';

h_ = plot(x, logEst, 'Color', 'cyan', ...
      'LineStyle', '-', 'LineWidth', 2, ...
      'Marker', 'none', 'MarkerSize', 6);
legh_(7) = h_;
legt_{7} = 'Lognormal fit';

h_ = legend(ax_, legh_, legt_, leginfo_{:});
set(h_, 'Interpreter', 'none');
hold off;
xlim([-0.005 1]);

% Test the goodness of the fits with:
%  - Kullback-Leibler divergence
%  - Bhattacharyya distance
%  - Metric based on the Bhattacharyya coefficient, proposed by Comaniciu,
%    Ramesh, and Meer
logTwoData = log2(prob);
```

```
    for j = 1:length(logTwoData)
        if (isinf(log2(logTwoData(j))))
            logTwoData(j) = 0;
        end
    end

    KLray = sum(prob .* (logTwoData - log2(rayleighEst)));
    KLric = sum(prob .* (logTwoData - log2(ricianEst)));
    KLnak = sum(prob .* (logTwoData - log2(nakagamiEst)));
    KLbet = sum(prob .* (logTwoData - log2(betaEst)));
    KLgam = sum(prob .* (logTwoData - log2(gammaEst)));
    KLlog = sum(prob .* (logTwoData - log2(logEst)));

    Bray = -log2(sum(sqrt(prob .* rayleighEst)));
    Bric = -log2(sum(sqrt(prob .* ricianEst)));
    Bnak = -log2(sum(sqrt(prob .* nakagamiEst)));
    Bbet = -log2(sum(sqrt(prob .* betaEst)));
    Bgam = -log2(sum(sqrt(prob .* gammaEst)));
    Blog = -log2(sum(sqrt(prob .* logEst)));

    BModray = sqrt(1 - (sum(sqrt(prob .* rayleighEst))));
    BModric = sqrt(1 - (sum(sqrt(prob .* ricianEst))));
    BModnak = sqrt(1 - (sum(sqrt(prob .* nakagamiEst))));
    BModbet = sqrt(1 - (sum(sqrt(prob .* betaEst))));
    BModgam = sqrt(1 - (sum(sqrt(prob .* gammaEst))));
    BModlog = sqrt(1 - (sum(sqrt(prob .* logEst))));

    % Display the results in a table.
    s = sprintf('----- %d kHz Sinusoid -----', tone);
    disp(s);
    disp('            K-L     Bhat.   CRM');
    s = sprintf('Beta      : %.4f  %.4f  %.4f [alpha = %.4f, beta = %.4f]', ...
        KLbet, Bbet, BModbet, paramEstsBeta(1), paramEstsBeta(2));
    disp(s);
    s = sprintf('Gamma     : %.4f  %.4f  %.4f [alpha = %.4f, beta = %.4f]', ...
        KLgam, Bgam, BModgam, paramEstsGamma(1), paramEstsGamma(2));
    disp(s);
    s = sprintf('Lognormal : %.4f  %.4f  %.4f [mu = %.4f, sigma = %.4f]', ...
        KLlog, Blog, BModlog, paramEstsLog(1), paramEstsLog(2));
    disp(s);
    s = sprintf('Nakagami-m: %.4f  %.4f  %.4f [m = %.4f, omega = %.4f]', ...
        KLnak, Bnak, BModnak, paramEstsNakagami(1), paramEstsNakagami(2));
    disp(s);
    s = sprintf('Rayleigh  : %.4f  %.4f  %.4f [sigma = %.4f]', KLray, Bray, ...
        BModray, paramEstsRayleigh(1));
    disp(s);
    s = sprintf(...
        'Rician    : %.4f  %.4f  %.4f [s = %.4f, sigma = %.4f, K = %.4f]', ...
        KLric, Bric, BModric, paramEstsRician(1), paramEstsRician(2), K);
    disp(s);
end

u = unique(dbEnvelope(1,:));
assert(length(u) == length(dbEnvelope(1,:)), ...
       'Duplicates found in envelope 1.');

u = unique(dbEnvelope(2,:));
assert(length(u) == length(dbEnvelope(2,:)), ...
       'Duplicates found in envelope 2.');
```

```
u = unique(dbEnvelope(3,:));
assert(length(u) == length(dbEnvelope(3,:)), ...
       'Duplicates found in envelope 3.');

u = unique(dbEnvelope(4,:));
assert(length(u) == length(dbEnvelope(4,:)), ...
       'Duplicates found in envelope 4.');

u = unique(dbEnvelope(5,:));
assert(length(u) == length(dbEnvelope(5,:)), ...
       'Duplicates found in envelope 5.');

zeroIndex1 = find(dbEnvelope(1,:) >= 0);
prob1 = 1:zeroIndex1 - 1;
prob1 = prob1 / length(magnitudes);
zeroIndex2 = find(dbEnvelope(2,:) >= 0);
prob2 = 1:zeroIndex2 - 1;
prob2 = prob2 / length(magnitudes);
zeroIndex3 = find(dbEnvelope(3,:) >= 0);
prob3 = 1:zeroIndex3 - 1;
prob3 = prob3 / length(magnitudes);
zeroIndex4 = find(dbEnvelope(4,:) >= 0);
prob4 = 1:zeroIndex4 - 1;
prob4 = prob4 / length(magnitudes);
zeroIndex5 = find(dbEnvelope(5,:) >= 0);
prob5 = 1:zeroIndex5 - 1;
prob5 = prob5 / length(magnitudes);

figure;
semilogy(dbEnvelope(1, 1:zeroIndex1 - 1), prob1, ...
         dbEnvelope(2, 1:zeroIndex2 - 1), prob2, ...
         dbEnvelope(3, 1:zeroIndex3 - 1), prob3, ...
         dbEnvelope(4, 1:zeroIndex4 - 1), prob4, ...
         dbEnvelope(5, 1:zeroIndex5 - 1), prob5);
grid minor;
grid;
grid minor;
title('Cumulative Distribution of Sinusoids', 'FontWeight', 'bold');
ylabel('Probability Signal Level < Abscissa');
xlabel('Signal Level Relative to Average (dB)');
legend('35 kHz', '45 kHz', '60 kHz', '75 kHz', '85 kHz', ...
       'Location', 'NorthWest');
minValues = [dbEnvelope(1, 1) dbEnvelope(2, 1) dbEnvelope(3, 1) ...
             dbEnvelope(4, 1) dbEnvelope(5, 1)];
minX = min(minValues);
minX = int32(ceil(-minX / 10)) * -10;
xlim([minX 0]);
```

Appendix B
Source Code for Chapter 3

B.1 FFT Convolution

```
function [y] = fconv(x, h, normalize)
%FCONV Fast Convolution
%   y = FCONV(x, h) convolves x and h, where
%       x is the time domain signal and
%       h is the filter kernel.
%
%   y = FCONV(x, h, normalize) convolves x and h and normalizes the
%       result to -1..1 if normalize is 'true'.

    if (nargin < 2)
        error('Too few arguments specified in function.');
    end

    if (nargin > 3)
        error('Too many arguments specified in function.');
    end

    if (nargin == 2)
        normalize = 'false';
    end

    lenY = length(x) + length(h) - 1;
    lenYPow2 = pow2(nextpow2(lenY)); % Find smallest power of 2 > lenY
    X = fft(x, lenYPow2);            % FFT
    H = fft(h, lenYPow2);            % FFT
    Y = X .* H;                      % Multiplication in frequency domain
    y = real(ifft(Y, lenYPow2));     % Inverse FFT
    y = y(1:lenY);                   % Use first lenY elements
    if (strcmp(normalize, 'true') == 1)
        y = y / max(abs(y));         % Normalize output
    end
```

B.2 Hard Limiter

```
function [y] = hardlimit(x)
%HARDLIMIT limits the amplitude of a signal.
%   As indicated in,
%   J. Jones, "Hard-Limiting of Two Signals in Random Noise,"
%   IEEE Trans. on Information Theory, Vol. 9, Iss. 1, pp. 34-42, Jan. 1963,
%   the ideal limiter is described by its amplitude characteristic g(x),
%   with which the limiter output y(t) can be expressed uniquely in terms
%   of the input as
%   y(t) = g[x(t)] = {+1, 0, -1} if x(t) {> 0, = 0, < 0}.

    y = 2 * ((x > 0) - 0.5);
```

B.3 Second-Order IIR Bandpass Filter

```
function [y] = filterSignal(x, samplingRate, freq, symbolsPerSecond)
%FILTERSIGNAL implements a second-order IIR bandpass filter.
%   y = FILTERSIGNAL(x, samplingRate, freq, symbolsPerSecond)
%       produces a bandpass-filtered signal of the input signal x with a
%       -3 dB bandwidth of symbolsPerSecond Hz centered on freq Hz.
%       The samplingRate of x must be provided as an input argument.

    f = freq / samplingRate;
    bw = symbolsPerSecond / samplingRate;
    R = 1 - 3 * bw;
    K = (1 - 2 * R * cos(2 * pi * f) + R * R) / (2 - 2 * cos(2 * pi * f));
    a0 = 1 - K;
    a1 = 2 * (K - R) * cos(2 * pi * f);
    a2 = R * R - K;
    b1 = 2 * R * cos(2 * pi * f);
    b2 = -R * R;

    numSamples = length(x);
    y = zeros(1, numSamples);
    y(1) = 0;
    y(2) = 0;
    for i = 3:numSamples
        y(i) = a0 * x(i) + a1 * x(i - 1) + a2 * x(i - 2) + ...
               b1 * y(i - 1) + b2 * y(i - 2);
    end
```

B.4 Generation of Chirp Signals and FSK and PSK Waveforms

```
% Author: Brian Borowski
% Date created: 09/15/2009
% Date last modified: 03/27/2010
% Generates the signals used to test a channel. The signals include
% sequential 50ms LFM chirps and FSK and PSK waveforms.

%% Start with a clean slate.
clear all;
close all;
clc;

%% Key parameters
generateData = 0;
chirpSeconds = 5;    % 1 long chirp to precede packet stream
chirpMs = 50;        % Used in channel characterization
samplingRate = 48000;
symbolsPerSecond = 3500;
carrierFreq = 17500;
numberOfPackets = 50;

%% Generate sounding signal.
chirpSignal = chirp(0, samplingRate/2, samplingRate, chirpMs/1000);
Xr = real(chirpSignal);
Xr = Xr * 0.98;
soundingSignal = zeros(1, samplingRate * 60);

lenChirp = length(chirpSignal);
```

```
lenSounding = length(soundingSignal);
numChirps = lenSounding / lenChirp;
offset = 0;
for i = 1:numChirps
    soundingSignal(1+offset:lenChirp+offset) = Xr;
    offset = offset + lenChirp;
end
wavwrite(soundingSignal, samplingRate, 'Reference/SoundingSignal.wav');

%% Only use this section once to generate the packet data.
if (generateData)
    packetSizeBytes = 256;
    packetSizeBits = packetSizeBytes * 8;
    data = randi([0 1], packetSizeBits, 1);
    dlmwrite('PacketBits.txt', data, 'delimiter', ' ', 'newline','pc');
end

%% Parameters common to both FSK and PSK modulation
data = dlmread('Reference/PacketBits.txt', ' ');
samplesPerBit = floor(samplingRate / symbolsPerSecond);

if (mod(carrierFreq, symbolsPerSecond))
    error('Carrier frequency must be a multiple of symbol rate.');
end
fc = [(carrierFreq - symbolsPerSecond / 2) ...
      (carrierFreq + symbolsPerSecond / 2)];

if (symbolsPerSecond < 500)
    chirpRange = [(carrierFreq - 500) (carrierFreq + 500)];
else
    chirpRange = [(carrierFreq - symbolsPerSecond) ...
                  (carrierFreq + symbolsPerSecond)];
end
packetChirp = chirp(chirpRange(1), chirpRange(2), samplingRate, 0.05);
packetChirpReal = real(packetChirp);

guardTime = zeros(1, 0.01*samplingRate);
numberOfBits = length(data);
interPacketSilence = zeros(1, samplingRate);

streamChirp = chirp(0, 24000, samplingRate, chirpSeconds);
streamReal = real(streamChirp);
streamReal = streamReal * 0.98;
filename = sprintf('Reference/ChirpRef_%ds.wav', chirpSeconds);
wavwrite(streamReal, samplingRate, filename);

%% Generate FSK waveform.
txFSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    for j = 1:samplesPerBit
        t = t + 2 * pi * fc(data(i)+1) / samplingRate;
        if (t > pi)
            t = t - 2 * pi;
        end
        txFSK(j + samplesPerBit*(i-1)) = cos(t);
    end
end
packetFSK = [packetChirpReal guardTime txFSK interPacketSilence];
packetFSK = packetFSK * 0.98;
```

```
%% Generate stream of FSK packets.
%  Start with one long LFM chirp to accurately estimate the LTI channel.
lenPacketFSK = length(packetFSK);
lenStreamChirp = length(streamChirp);
offset = lenStreamChirp + length(interPacketSilence);
packetFSKStream = zeros(1, numberOfPackets*lenPacketFSK + offset);
packetFSKStream(1:lenStreamChirp) = streamReal;
for i = 1:numberOfPackets
    packetFSKStream(1+offset:lenPacketFSK+offset) = packetFSK;
    offset = offset + lenPacketFSK;
end
filename = sprintf('Reference/FSKStream_%dHz_%dbps.wav', ...
                   carrierFreq, symbolsPerSecond);
wavwrite(packetFSKStream, samplingRate, filename);

%% Generate PSK waveform.
txPSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    offset = samplesPerBit * (i-1);
    for j = 1:samplesPerBit
        t = t + 2 * pi * carrierFreq / samplingRate;
        if (t > pi)
            t = t - 2 * pi;
        end
        if (data(i) == 0)
            txPSK(j + offset) = -cos(t);
        else
            txPSK(j + offset) = cos(t);
        end
    end
    t = 0.0;
end
packetPSK = [packetChirpReal guardTime txPSK interPacketSilence];
packetPSK = packetPSK * 0.98;

%% Generate stream of PSK packets.
%  Start with one long LFM chirp to accurately estimate the LTI channel.
lenPacketPSK = length(packetPSK);
lenStreamChirp = length(streamChirp);
offset = lenStreamChirp + length(interPacketSilence);
packetPSKStream = zeros(1, numberOfPackets*lenPacketPSK + offset);
packetPSKStream(1:lenStreamChirp) = streamReal;
for i = 1:numberOfPackets
    packetPSKStream(1+offset:lenPacketPSK+offset) = packetPSK;
    offset = offset + lenPacketPSK;
end
filename = sprintf('Reference/PSKStream_%dHz_%dbps.wav', ...
                   carrierFreq, symbolsPerSecond);
wavwrite(packetPSKStream, samplingRate, filename);
```

B.5 Verification of Channel Simulation

```
% Author: Brian Borowski
% Date created: 09/15/2009
% Date last modified: 03/27/2010
% Computes the BERs of real packets transmitted through the Rubbermaid tub
```

```
% and compares the values to that obtained when convolving the packet
% waveform with the tub's estimated impulse response.

%% Start with a clean slate.
clear all;
close all;
clc;

%% Initialize basic parameters.
samplingRate = 48000;
symbolsPerSecond = 2500;
carrierFreq = 7500;
numberOfPackets = 50;
useLimiter = 0;

%% Generate reference chirp signal.
chirpSeconds = 5;
streamChirp = chirp(0, samplingRate/2, samplingRate, chirpSeconds);

%% Initialize variables used in generating packets.
data = dlmread('Reference/PacketBits.txt', ' ');
ir_FSK_file = 'Tub/IR_FSK.wav';
ir_PSK_file = 'Tub/IR_PSK.wav';
numberOfBits = length(data);
samplesPerBit = floor(samplingRate / symbolsPerSecond);
if (mod(carrierFreq, symbolsPerSecond))
    error('Carrier frequency must be a multiple of symbol rate.');
end
fc = [(carrierFreq - symbolsPerSecond / 2) ...
      (carrierFreq + symbolsPerSecond / 2)];

if (symbolsPerSecond < 500)
    chirpRange = [(carrierFreq - 500) (carrierFreq + 500)];
else
    chirpRange = [(carrierFreq - symbolsPerSecond) ...
                  (carrierFreq + symbolsPerSecond)];
end
packetChirp = chirp(chirpRange(1), chirpRange(2), samplingRate, 0.05);
guardTime = zeros(1, 0.01 * samplingRate);
samplesPerPacket = length(packetChirp) + length(guardTime) + ...
                   samplesPerBit * numberOfBits;

%% CPFSK modulation
txFSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    for j = 1:samplesPerBit
        t = t + 2 * pi * fc(data(i)+1) / samplingRate;
        if (t > pi)
            t = t - 2 * pi;
        end
        txFSK(j + samplesPerBit*(i-1)) = cos(t);
    end
end

%% PSK modulation
txPSK = zeros(1, samplesPerBit * numberOfBits);
t = 0.0;
for i = 1:numberOfBits
    offset = samplesPerBit * (i-1);
```

```
    for j = 1:samplesPerBit
        t = t + 2 * pi * carrierFreq / samplingRate;
        if (t > pi)
            t = t - 2 * pi;
        end
        if (data(i) == 0)
            txPSK(j + offset) = -cos(t);
        else
            txPSK(j + offset) = cos(t);
        end
    end
    t = 0.0;
end

%% Assemble packets.
txFSK = [guardTime packetChirp guardTime txFSK];
txPSK = [guardTime packetChirp guardTime txPSK];

%% Read raw data samples.
filename = sprintf('Recordings5/FSKStream_%dHz_%dbps.wav', ...
                   carrierFreq, symbolsPerSecond);
rawSamplesFSK = wavread(filename);
filename = sprintf('Recordings5/PSKStream_%dHz_%dbps.wav', ...
                   carrierFreq, symbolsPerSecond);
rawSamplesPSK = wavread(filename);

%% Compute IR at time of FSK test.
irFSK = fftshift( ...
    real(xcorr(rawSamplesFSK(1:10*samplingRate), conj(streamChirp))));
[maxValue maxIndexFSK] = max(irFSK);
earlyPeakIndex = ...
    find(irFSK(maxIndexFSK - 0.0005*samplingRate:end) > 0.25*maxValue);
[~, mainPeakIndex] = max(irFSK(maxIndexFSK - 0.0005*samplingRate:end));
offset = mainPeakIndex - earlyPeakIndex;
irFSK = irFSK(maxIndexFSK - offset:maxIndexFSK + 0.050*samplingRate - offset);
irFSK = irFSK / max(abs(irFSK));
figure;
n = (1:length(irFSK))*1000/samplingRate;
plot(n, irFSK);
xlim([0 10]);
ylim([-1 1]);
title('Impulse Response of Tub during FSK Test', 'FontWeight', 'bold');
xlabel('Time Delay (ms)');
ylabel('Intensity');
figure;
[h w] = freqz(irFSK, 1);
f = w/(2*pi) * samplingRate/1000;
subplot(2, 1, 1),
plot(f, mag2db(abs(h))),
xlim([0 f(end)]),
ylim([-30 30]),
title('Frequency Response', 'FontWeight', 'bold'),
xlabel('Frequency (kHz)'),
ylabel('Magnitude (dB)');

phase = unwrap(angle(h)) * 180/pi;
subplot(2, 1, 2),
plot(f, phase),
xlim([0 f(end)]),
title('Phase Response', 'FontWeight', 'bold'),
```

```
xlabel('Frequency (kHz)'),
ylabel('Phase (degrees)');

irFSK = irFSK * 0.98;
wavwrite(irFSK, samplingRate, 'Tub/IR_FSK.wav');

%% Compute IR at time of PSK test.
irPSK = fftshift( ...
    real(xcorr(rawSamplesPSK(1:10*samplingRate), conj(streamChirp))));
[maxValue maxIndexPSK] = max(irPSK);
earlyPeakIndex = ...
    find(irPSK(maxIndexPSK - 0.0005 * samplingRate:end) > 0.25*maxValue);
[mainPeak mainPeakIndex] = max(irPSK(maxIndexPSK - 0.0005*samplingRate:end));
offset = mainPeakIndex - earlyPeakIndex;
irPSK = irPSK(maxIndexPSK - offset:maxIndexPSK + 0.050*samplingRate - offset);
irPSK = irPSK / max(abs(irPSK));
figure;
n = (1:length(irPSK))*1000/samplingRate;
plot(n, irPSK);
xlim([0 10]);
ylim([-1 1]);
title('Impulse Response of Tub during PSK Test', 'FontWeight', 'bold');
xlabel('Time Delay (ms)');
ylabel('Intensity');
figure;
[h w] = freqz(irPSK, 1);
f = w/(2*pi) * samplingRate/1000;
subplot(2, 1, 1),
plot(f, mag2db(abs(h))),
xlim([0 f(end)]),
ylim([-30 30]),
title('Frequency Response', 'FontWeight', 'bold'),
xlabel('Frequency (kHz)'),
ylabel('Magnitude (dB)');

phase = unwrap(angle(h)) * 180/pi;
phase = phase * 180 / pi;
subplot(2, 1, 2),
plot(f, phase),
xlim([0 f(end)]),
title('Phase Response', 'FontWeight', 'bold'),
xlabel('Frequency (kHz)'),
ylabel('Phase (degrees)');

irPSK = irPSK * 0.98;
wavwrite(irPSK, samplingRate, 'Tub/IR_PSK.wav');

%% Convolve with impulse response.
txFSK = fconv(txFSK, irFSK');
txPSK = fconv(txPSK, irPSK');

%% Generate reference signals for symbols.
t = 0:samplesPerBit-1;
cos0 = cos(2 * pi * fc(1)/samplingRate * t);
sin0 = sin(2 * pi * fc(1)/samplingRate * t);
cos1 = cos(2 * pi * fc(2)/samplingRate * t);
sin1 = sin(2 * pi * fc(2)/samplingRate * t);

psk0 = -cos(2 * pi * carrierFreq/samplingRate * t);
psk1 =  cos(2 * pi * carrierFreq/samplingRate * t);
```

```
if (symbolsPerSecond < 500)
    chirpRange = [(carrierFreq - 500) (carrierFreq + 500)];
else
    chirpRange = [(carrierFreq - symbolsPerSecond) ...
                  (carrierFreq + symbolsPerSecond)];
end
packetChirp = chirp(chirpRange(1), chirpRange(2), samplingRate, 0.05);

%% Synchronize on packet chirp signals.
correlatedSamples = real(fftshift(xcorr(txFSK, conj(packetChirp))));
[~, maxIndex] = max(correlatedSamples);
len = length(correlatedSamples);
if (maxIndex > len/2)
    maxIndex = maxIndex - len;
end
start = maxIndex + length(packetChirp)+length(guardTime);
txFSK = txFSK(start:start+samplesPerBit * numberOfBits);

correlatedSamples = real(fftshift(xcorr(txPSK, conj(packetChirp))));
[~, maxIndex] = max(correlatedSamples);
len = length(correlatedSamples);
if (maxIndex > len/2)
    maxIndex = maxIndex - len;
end
start = maxIndex + length(packetChirp)+length(guardTime);
txPSK = txPSK(start:start+samplesPerBit * numberOfBits);

%% Demodulation
expectedData = dlmread('Reference/PacketBits.txt', ' ');
expectedData = expectedData';
numberOfBits = length(expectedData);
guardTime = zeros(1, 0.01*samplingRate);
interPacketSilence = zeros(1, samplingRate);
packet = zeros(1, samplesPerBit * numberOfBits);

offsetFSK = maxIndexFSK + length(streamChirp) + length(interPacketSilence)/2;
offsetPSK = maxIndexPSK + length(streamChirp) + length(interPacketSilence)/2;
numberOfRawSamplesFSK = length(rawSamplesFSK);
numberOfRawSamplesPSK = length(rawSamplesPSK);
berFSK_e = zeros(1, numberOfPackets + 1);
berFSK_q = zeros(1, numberOfPackets + 1);
berPSK = zeros(1, numberOfPackets + 1);

disp('Bit Error Rates (BERs)');
disp('FSK-E   FSK-Q   PSK');
disp('--------------------');
for iter = 1:numberOfPackets + 1
    if (iter ~= numberOfPackets + 1)
        endSample = min( ...
            length(interPacketSilence) + samplesPerPacket + offsetFSK, ...
            numberOfRawSamplesFSK);
        snip = rawSamplesFSK(1 + offsetFSK:endSample);
        correlatedSamples = real(fftshift(xcorr(snip, conj(packetChirp))));
        [~, maxIndex] = max(correlatedSamples);
        start = maxIndex + length(packetChirp) + length(guardTime);
        packet = snip(start:start + samplesPerBit*numberOfBits);
        packet = packet';
    else
        packet = txFSK;
```

```
end

% Prepare FSK signal for envelope detector.
if (useLimiter)
    zeroSignal = abs(hilbert(filterSignal( ...
        hardlimit(packet), samplingRate, fc(1), symbolsPerSecond)));
    oneSignal = abs(hilbert(filterSignal( ...
        hardlimit(packet), samplingRate, fc(2), symbolsPerSecond)));
else
    zeroSignal = abs(hilbert(filterSignal( ...
        packet, samplingRate, fc(1), symbolsPerSecond)));
    oneSignal = abs(hilbert(filterSignal( ...
        packet, samplingRate, fc(2), symbolsPerSecond)));
    maxZero = max(zeroSignal);
    maxOne = max(oneSignal);
    if (maxZero > maxOne)
        oneSignal = oneSignal * (maxZero/maxOne);
    else
        zeroSignal = zeroSignal * (maxOne/maxZero);
    end
end
diff = oneSignal - zeroSignal;

rxFSK_e = zeros(1, numberOfBits);
rxFSK_q = zeros(1, numberOfBits);
temp = floor(samplesPerBit / 2);
for i = 1:numberOfBits
    % Envelope detector for FSK demodulation
    % Sample in second half of symbol.
    rcv = sum(diff((i-1)*samplesPerBit + temp:i*samplesPerBit));
    rxFSK_e(i) = (rcv > 0);

    % Quadrature (energy) detector
    if (useLimiter)
        rcv = hardlimit(packet((i-1)*samplesPerBit + 1:i*samplesPerBit));
    else
        rcv = packet((i-1)*samplesPerBit + 1:i*samplesPerBit);
    end
    Izero = rcv .* cos0;
    Qzero = rcv .* sin0;
    Ione = rcv .* cos1;
    Qone = rcv .* sin1;
    z1 = (sum(Izero))^2;
    z2 = (sum(Qzero))^2;
    z3 = (sum(Ione))^2;
    z4 = (sum(Qone))^2;
    energy0 = z1 + z2;
    energy1 = z3 + z4;
    rxFSK_q(i) = (energy1 > energy0);
end

berFSK_e(iter) = 100 * sum(abs(rxFSK_e - expectedData)) / numberOfBits;
berFSK_q(iter) = 100 * sum(abs(rxFSK_q - expectedData)) / numberOfBits;

if (iter ~= numberOfPackets + 1)
    endSample = min( ...
        length(interPacketSilence) + samplesPerPacket+offsetPSK, ...
        numberOfRawSamplesPSK);
    snip = rawSamplesPSK(1 + offsetPSK:endSample);
    correlatedSamples = real(fftshift(xcorr(snip, conj(packetChirp))));
```

```
    [~, maxIndex] = max(correlatedSamples);
    start = maxIndex + length(packetChirp) + length(guardTime);
    packet = snip(start:start + samplesPerBit*numberOfBits);
    packet = packet';
else
    packet = txPSK;
end

if (iter == 1 || iter == numberOfPackets + 1)
    firstSymbol = packet(1:samplesPerBit);
    offset = 0;
    oldZ0 = -inf;
    oldZ1 = -inf;
    phi = 0;
    for i = 1:24
        if (expectedData(1) == 1)
            one = firstSymbol .* psk1;
            z1 = (sum(one));
            if (z1 > oldZ1)
                oldZ1 = z1;
                offset = phi;
            end
            phi = phi + (2 * pi) / 24;
            psk1 = cos(2 * pi * carrierFreq/samplingRate * t + phi);
        else
            zero = firstSymbol .* psk0;
            z0 = (sum(zero));
            if (z0 > oldZ0)
                oldZ0 = z0;
                offset = phi;
            end
            phi = phi + (2 * pi) / 24;
            psk0 = -cos(2 * pi * carrierFreq/samplingRate * t + phi);
        end
    end

    psk0 = -cos(2 * pi * carrierFreq/samplingRate * t + offset);
    psk1 =  cos(2 * pi * carrierFreq/samplingRate * t + offset);
end

% PSK demodulation via a correlation receiver
rxPSK = zeros(1, numberOfBits);
for i = 1:numberOfBits
    rcv = packet((i-1)*samplesPerBit + 1:i*samplesPerBit);
    zero = rcv .* psk0;
    one = rcv .* psk1;
    z0 = (sum(zero));
    z1 = (sum(one));
    rxPSK(i) = (z1 > z0);
end

berPSK(iter) = 100 * sum(abs(rxPSK - expectedData)) / numberOfBits;

% Display FSK/PSK bit error rates for a single packet.
if (iter ~= numberOfPackets + 1)
    s = sprintf('%.2f\t%.2f\t%.2f', ...
                berFSK_e(iter), berFSK_q(iter), berPSK(iter));
    disp(s);

    offsetFSK = offsetFSK + samplesPerPacket + length(interPacketSilence);
```

```
        offsetPSK = offsetPSK + samplesPerPacket + length(interPacketSilence);
    end
end

%% Display average bit error rates.
disp('--------------------');
avgBerFSK_e = 0;
avgBerFSK_q = 0;
avgBerPSK = 0;
for i = 1:numberOfPackets
    avgBerFSK_e = avgBerFSK_e + berFSK_e(i);
    avgBerFSK_q = avgBerFSK_q + berFSK_q(i);
    avgBerPSK = avgBerPSK + berPSK(i);
end
avgBerFSK_e = avgBerFSK_e / numberOfPackets;
avgBerFSK_q = avgBerFSK_q / numberOfPackets;
avgBerPSK = avgBerPSK / numberOfPackets;

disp('Transmitted Data');
s = sprintf('BER for FSK modulation (envelope detection): %.2f%%', ...
            avgBerFSK_e);
disp(s);
s = sprintf('BER for FSK modulation (quadrature receiver): %.2f%%', ...
            avgBerFSK_q);
disp(s);
s = sprintf('BER for PSK modulation: %.2f%%', avgBerPSK);
disp(s);
disp('--------------------');

index = numberOfPackets + 1;
disp('Simulation with Impulse Response');
s = sprintf('BER for FSK modulation (envelope detection): %.2f%%', ...
            berFSK_e(index));
disp(s);
s = sprintf('BER for FSK modulation (quadrature receiver): %.2f%%', ...
            berFSK_q(index));
disp(s);
s = sprintf('BER for PSK modulation: %.2f%%', berPSK(index));
disp(s);
```

B.6 Main Function for OMNeT++ Simulation

```
#include <stdio.h>
#include "libchannel.h"
#include "libphy.h"
#include "cownedobject.h"
#include "envirdefs.h"
#include "startup.h"
#include "ver.h"

USING_NAMESPACE

//
// The main() function
//
ENVIR_API int main(int argc, char *argv[])
{
    cStaticFlag dummy;
```

```
    printf(OMNETPP_PRODUCT " Discrete Event Simulation "
           " (C) 1992-2008 Andras Varga, OpenSim Ltd.\n");
    printf("Version: " OMNETPP_VERSION_STR ", build: " OMNETPP_BUILDID
           ", edition: " OMNETPP_EDITION "\n");
    printf("See the license for distribution terms and warranty disclaimer\n");

    // Call MATLAB application initialization function.
    if (!mclInitializeApplication(NULL,0)) {
        fprintf(stderr, "Could not initialize the MATLAB application.\n");
        exit(-1);
    }
    // Call the library initialization functions.
    if (!libchannelInitialize()){
        fprintf(stderr, "Could not initialize libchannel properly.\n");
        exit(-1);
    }
    if (!libphyInitialize()){
        fprintf(stderr, "Could not initialize libphy properly.\n");
        exit(-1);
    }

    int exitcode = setupUserInterface(argc, argv, NULL);

    // Call the library termination functions.
    libphyTerminate();
    libchannelTerminate();
    // Call MATLAB application termination function.
    mclTerminateApplication();

    printf("\nEnd.\n");

    return exitcode;
}
```

Appendix C
Validation of Emulator in Chapter 3

Comparison of bit error rates obtained with packet transmission through the office test environment (Average BER) versus convolution with an impulse response estimate (Simulated BER).

C.1 7.5 kHz, 250 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 12:33 P.M.	10/1/2009 12:33 P.M.	10/1/2009 12:33 P.M.	10/1/2009 12:33 P.M.	10/1/2009 12:37 P.M.
Frequency (Hz)	7500				
Bit Rate (bps)	250				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00
Average BER	0.00	0.00	0.00	0.00	0.00
Simulated BER	0.00	0.00	0.00	0.00	0.00
% Difference	0.00	0.00	0.00	0.00	0.00

C.2 7.5 kHz, 500 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 12:43 P.M.	10/1/2009 12:43 P.M.	10/1/2009 12:43 P.M.	10/1/2009 12:43 P.M.	10/1/2009 12:48 P.M.
Frequency (Hz)	7500				
Bit Rate (bps)	500				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00
21	0.00	0.00	0.00	0.00	0.00
22	0.00	0.00	0.00	0.00	0.00
23	0.00	0.00	0.00	0.00	0.00
24	0.00	0.00	0.00	0.00	0.00
25	0.00	0.00	0.00	0.00	0.00
26	0.00	0.00	0.00	0.00	0.00
27	0.00	0.00	0.00	0.00	0.00
28	0.00	0.00	0.00	0.00	0.00
29	0.00	0.00	0.00	0.00	0.00
30	0.00	0.00	0.00	0.00	0.00
31	0.00	0.00	0.00	0.00	0.00
32	0.00	0.00	0.00	0.00	0.00
33	0.00	0.00	0.00	0.00	0.00
34	0.00	0.00	0.00	0.00	0.00
35	0.00	0.00	0.00	0.00	0.00
36	0.00	0.00	0.00	0.00	0.00
37	0.00	0.00	0.00	0.00	0.00
38	0.00	0.00	0.00	0.00	0.00
39	0.00	0.00	0.00	0.00	0.00

40	0.00	0.00	0.00	0.00	0.00
41	0.00	0.00	0.00	0.00	0.00
42	0.00	0.00	0.00	0.00	0.00
43	0.00	0.00	0.00	0.00	0.00
44	0.00	0.00	0.00	0.00	0.00
45	0.00	0.00	0.00	0.00	0.00
46	0.00	0.00	0.00	0.00	0.00
47	0.00	0.00	0.00	0.00	0.00
48	0.00	0.00	0.00	0.00	0.00
49	0.00	0.00	0.00	0.00	0.00
50	0.00	0.00	0.00	0.00	0.00
Average BER	0.00	0.00	0.00	0.00	0.00
Simulated BER	0.00	0.00	0.00	0.00	0.00
% Difference	0.00	0.00	0.00	0.00	0.00

C.3 7.5 kHz, 1250 bps

Demodulation	FSK (Envelope Detector) Amplitude Comparator	FSK (Envelope Detector) Hard Limiter	FSK (Quadrature Receiver) Default	FSK (Quadrature Receiver) Hard Limiter	PSK
Date/Time	10/1/2009 12:51 P.M.	10/1/2009 12:51 P.M.	10/1/2009 12:51 P.M.	10/1/2009 12:51 P.M.	10/1/2009 12:54 P.M.
Frequency (Hz)	7500				
Bit Rate (bps)	1250				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	6.10	12.99	6.05	4.88	0.00
2	5.96	12.79	6.25	4.74	0.00
3	6.10	12.89	6.30	4.74	0.00
4	5.91	12.84	6.15	4.54	0.00
5	6.01	12.89	6.15	5.03	0.00
6	6.05	12.74	6.10	4.54	0.00
7	5.91	12.79	6.25	4.39	0.00
8	5.96	12.50	6.20	4.39	0.00
9	5.91	12.84	6.01	4.83	0.00
10	5.62	12.55	6.20	4.49	0.00
11	5.76	12.74	6.20	4.69	0.00
12	5.32	12.65	6.10	4.30	0.00
13	5.47	12.55	6.05	4.64	0.00
14	5.52	12.65	6.15	4.54	0.00
15	5.57	12.60	6.10	4.20	0.00
16	5.27	12.65	6.01	4.49	0.00
17	5.47	12.30	6.05	4.59	0.00
18	5.32	12.50	6.05	4.39	0.00
19	5.13	12.30	5.96	4.54	0.00
20	5.18	12.40	5.86	4.05	0.00
21	5.08	12.55	5.96	4.15	0.00
22	5.08	12.26	5.91	4.35	0.00
23	5.22	12.35	5.86	4.30	0.00
24	4.83	12.65	5.96	4.30	0.00

25	4.74	12.35	6.05	4.20	0.00
26	4.74	12.35	5.86	4.20	0.00
27	4.74	12.30	5.71	3.91	0.00
28	4.30	11.72	5.91	4.25	0.00
29	4.49	12.21	5.96	4.10	0.00
30	4.49	11.96	5.91	4.15	0.00
31	4.69	12.16	5.81	3.91	0.00
32	4.10	12.11	5.62	4.00	0.00
33	4.64	12.50	5.57	4.00	0.00
34	4.30	12.06	5.42	4.05	0.00
35	4.15	12.06	5.52	4.30	0.00
36	4.30	12.11	5.62	4.15	0.00
37	3.91	11.62	5.57	3.86	0.00
38	3.76	11.57	5.52	3.71	0.00
39	3.86	11.96	5.57	4.05	0.00
40	3.76	12.11	5.52	4.10	0.00
41	3.37	11.77	5.27	3.76	0.00
42	3.32	11.96	5.47	3.96	0.00
43	3.03	11.77	5.42	3.86	0.00
44	3.37	11.67	5.27	3.81	0.00
45	3.08	11.67	5.32	3.86	0.00
46	3.61	11.77	5.18	3.71	0.00
47	2.98	11.62	5.18	3.86	0.00
48	2.98	11.43	5.08	3.56	0.00
49	2.78	11.18	5.22	3.86	0.00
50	2.88	11.57	5.03	3.66	0.00
Average BER	4.68	12.25	5.79	4.22	0.00
Simulated BER	0.00	1.76	0.00	3.03	0.00
% Difference	4.68	10.49	5.79	1.19	0.00

C.4 7.5 kHz, 2500 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 12:57 P.M.	10/1/2009 12:57 P.M.	10/1/2009 12:57 P.M.	10/1/2009 12:57 P.M.	10/1/2009 1:00 P.M.
Frequency (Hz)	7500				
Bit Rate (bps)	2500				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	5.08	3.86	2.15	0.44	0.00
2	4.15	3.86	2.20	0.44	0.00
3	4.59	3.61	2.10	0.34	0.00
4	4.39	3.86	2.05	0.39	0.00
5	4.98	3.76	2.25	0.34	0.00
6	4.93	4.00	2.20	0.39	0.00
7	4.79	3.96	2.20	0.44	0.00
8	4.54	3.81	2.20	0.44	0.00

9	4.54	3.61	2.20	0.34	0.00
10	4.79	3.81	2.15	0.39	0.00
11	4.64	3.86	2.20	0.34	0.00
12	4.49	3.76	2.10	0.39	0.00
13	4.83	3.91	2.10	0.39	0.00
14	5.03	3.76	2.00	0.34	0.00
15	5.03	3.91	2.15	0.39	0.00
16	4.15	3.61	2.29	0.39	0.00
17	4.44	3.71	2.15	0.34	0.00
18	4.64	3.56	2.34	0.39	0.00
19	4.83	3.86	2.15	0.34	0.00
20	4.49	3.47	2.20	0.39	0.00
21	4.59	3.71	2.15	0.44	0.00
22	4.64	3.52	1.95	0.39	0.00
23	4.49	3.96	2.15	0.34	0.00
24	4.79	3.66	2.15	0.39	0.00
25	4.79	3.66	2.20	0.39	0.00
26	4.79	3.66	2.10	0.34	0.00
27	4.54	3.56	2.15	0.39	0.00
28	4.35	3.61	2.15	0.34	0.00
29	4.25	3.71	2.25	0.34	0.00
30	5.22	3.66	2.20	0.44	0.00
31	4.79	3.42	2.29	0.39	0.00
32	4.20	3.52	2.15	0.24	0.00
33	4.69	3.61	2.05	0.29	0.00
34	4.98	3.42	2.10	0.39	0.00
35	4.20	3.71	2.10	0.34	0.00
36	4.69	3.76	2.00	0.39	0.00
37	4.39	3.86	2.05	0.34	0.00
38	4.39	3.71	2.20	0.34	0.00
39	4.64	3.76	2.10	0.34	0.00
40	4.39	3.76	2.05	0.29	0.00
41	4.39	3.66	2.10	0.29	0.00
42	4.79	3.71	2.05	0.34	0.00
43	4.98	3.71	2.15	0.34	0.00
44	4.54	3.76	2.00	0.29	0.00
45	4.79	3.47	2.05	0.39	0.00
46	5.08	3.47	2.10	0.34	0.00
47	4.39	3.56	2.10	0.34	0.00
48	4.35	3.61	2.10	0.34	0.00
49	4.64	3.66	2.05	0.34	0.00
50	4.79	3.42	1.95	0.29	0.00
Average BER	4.64	3.70	2.13	0.36	0.00
Simulated BER	4.44	2.93	14.60	4.93	0.00
% Difference	0.20	0.77	12.47	4.57	0.00

C.5 7.5 kHz, 3750 bps

Demodulation	FSK (Envelope Detector) Amplitude Comparator	FSK (Envelope Detector) Hard Limiter	FSK (Quadrature Receiver) Default	FSK (Quadrature Receiver) Hard Limiter	PSK
Date/Time	10/1/2009 1:02 P.M.	10/1/2009 1:02 P.M.	10/1/2009 1:02 P.M.	10/1/2009 1:02 P.M.	10/1/2009 1:04 P.M.
Frequency (Hz)	7500				
Bit Rate (bps)	3750				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	2.78	0.68	0.44	7.18	0.05
2	2.83	0.68	0.44	7.13	0.00
3	2.73	0.73	0.44	7.08	0.05
4	2.88	0.73	0.44	7.23	0.00
5	2.83	0.63	0.44	7.23	0.00
6	2.73	0.68	0.44	7.13	0.00
7	2.88	0.68	0.44	7.03	0.00
8	2.78	0.68	0.44	7.23	0.00
9	2.93	0.68	0.44	7.13	0.00
10	2.69	0.68	0.44	7.28	0.00
11	2.78	0.73	0.44	7.18	0.00
12	2.64	0.83	0.44	7.08	0.00
13	2.59	0.73	0.44	7.03	0.00
14	2.59	0.68	0.44	7.28	0.00
15	2.83	0.73	0.44	7.18	0.00
16	2.83	0.73	0.44	6.88	0.00
17	2.69	0.68	0.44	7.03	0.00
18	2.73	0.78	0.44	7.13	0.05
19	2.64	0.78	0.44	7.23	0.05
20	2.83	0.68	0.44	7.03	0.00
21	2.73	0.63	0.44	7.13	0.05
22	2.64	0.68	0.44	7.32	0.05
23	2.73	0.78	0.44	7.08	0.05
24	3.17	0.68	0.44	7.28	0.10
25	2.59	0.68	0.44	7.23	0.20
26	2.59	0.68	0.44	7.13	0.20
27	2.88	0.73	0.44	7.23	0.20
28	2.69	0.68	0.44	7.18	0.20
29	2.98	0.73	0.49	7.32	0.24
30	2.59	0.68	0.44	7.37	0.24
31	2.64	0.73	0.44	7.32	0.24
32	2.69	0.73	0.49	7.47	0.29
33	2.78	0.78	0.44	7.32	0.24
34	2.69	0.73	0.44	7.23	0.24
35	2.64	0.73	0.44	7.18	0.24
36	2.64	0.78	0.44	7.42	0.24
37	2.59	0.73	0.49	7.32	0.24
38	2.88	0.68	0.49	7.23	0.24
39	2.73	0.68	0.44	7.57	0.15

40	2.78	0.68	0.44	7.23	0.15
41	2.78	0.68	0.44	7.42	0.05
42	2.78	0.68	0.49	7.08	0.05
43	3.13	0.73	0.49	7.47	0.05
44	2.64	0.68	0.44	7.42	0.05
45	2.88	0.68	0.49	7.18	0.05
46	2.64	0.73	0.44	7.28	0.00
47	2.73	0.73	0.44	7.47	0.00
48	2.78	0.68	0.44	7.08	0.00
49	2.88	0.68	0.44	7.28	0.00
50	2.73	0.68	0.49	7.37	0.00
Average BER	2.76	0.71	0.45	7.23	0.09
Simulated BER	3.96	0.83	0.54	9.18	0.00
% Difference	1.20	0.12	0.09	1.95	0.09

C.6 12.5 kHz, 250 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 1:47 P.M.	10/1/2009 1:47 P.M.	10/1/2009 1:47 P.M.	10/1/2009 1:47 P.M.	10/1/2009 1:50 P.M.
Frequency (Hz)	12500				
Bit Rate (bps)	250				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	24.46	0.00	0.00	0.00	0.00
2	24.46	0.00	0.00	0.00	0.00
3	24.46	0.00	0.00	0.00	0.00
4	24.46	0.00	0.00	0.00	0.00
5	24.46	0.00	0.00	0.00	0.00
6	24.46	0.00	0.00	0.00	0.00
7	24.46	0.00	0.00	0.00	0.00
8	24.46	0.00	0.00	0.00	0.00
9	24.46	0.00	0.00	0.00	0.00
10	24.46	0.00	0.00	0.00	0.00
11	24.46	0.00	0.00	0.00	0.00
12	24.46	0.00	0.00	0.00	0.00
13	24.46	0.00	0.00	0.00	0.00
14	24.46	0.00	0.00	0.00	0.00
15	24.46	0.00	0.00	0.00	0.00
16	24.46	0.00	0.00	0.00	0.00
17	24.46	0.00	0.00	0.00	0.00
18	24.46	0.00	0.00	0.00	0.00
19	24.46	0.00	0.00	0.00	0.00
20	24.46	0.00	0.00	0.00	0.00
Average BER	24.46	0.00	0.00	0.00	0.00
Simulated BER	24.46	0.00	0.00	0.00	0.00
% Difference	0.00	0.00	0.00	0.00	0.00

C.7 12.5 kHz, 500 bps

Demodulation	FSK (Envelope Detector) Amplitude Comparator	FSK (Envelope Detector) Hard Limiter	FSK (Quadrature Receiver) Default	FSK (Quadrature Receiver) Hard Limiter	PSK
Date/Time	10/1/2009 1:55 P.M.	10/1/2009 1:55 P.M.	10/1/2009 1:55 P.M.	10/1/2009 1:55 P.M.	10/1/2009 2:04 P.M.
Frequency (Hz)	12500				
Bit Rate (bps)	500				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	24.46	0.00	0.00	0.00	0.00
2	24.46	0.00	0.00	0.00	0.00
3	24.46	0.00	0.00	0.00	0.00
4	24.46	0.00	0.00	0.00	0.00
5	24.46	0.00	0.00	0.00	0.00
6	24.46	0.00	0.00	0.00	0.00
7	24.46	0.00	0.00	0.00	0.00
8	24.46	0.00	0.00	0.00	0.00
9	24.46	0.00	0.00	0.00	0.00
10	24.46	0.00	0.00	0.00	0.00
11	24.46	0.00	0.00	0.00	0.00
12	24.46	0.00	0.00	0.00	0.00
13	24.46	0.00	0.00	0.00	0.00
14	24.46	0.00	0.00	0.00	0.00
15	24.46	0.00	0.00	0.00	0.00
16	24.46	0.00	0.00	0.00	0.00
17	24.46	0.00	0.00	0.00	0.00
18	24.46	0.00	0.00	0.00	0.00
19	24.46	0.00	0.00	0.00	0.00
20	24.46	0.00	0.00	0.00	0.00
21	24.46	0.00	0.00	0.00	0.00
22	24.46	0.00	0.00	0.00	0.00
23	24.46	0.00	0.00	0.00	0.00
24	24.46	0.00	0.00	0.00	0.00
25	24.46	0.00	0.00	0.00	0.00
26	24.46	0.00	0.00	0.00	0.00
27	24.46	0.00	0.00	0.00	0.00
28	24.46	0.00	0.00	0.00	0.00
29	24.46	0.00	0.00	0.00	0.00
30	24.46	0.00	0.00	0.00	0.00
31	24.46	0.00	0.00	0.00	0.00
32	24.46	0.00	0.00	0.00	0.00
33	24.46	0.00	0.00	0.00	0.00
34	24.46	0.00	0.00	0.00	0.00
35	24.46	0.00	0.00	0.00	0.00
36	24.46	0.00	0.00	0.00	0.00
37	24.46	0.00	0.00	0.00	0.00
38	24.46	0.00	0.00	0.00	0.00

39	24.46	0.00	0.00	0.00	0.00
40	24.46	0.00	0.00	0.00	0.00
41	24.46	0.00	0.00	0.00	0.00
42	24.46	0.00	0.00	0.00	0.00
43	24.46	0.00	0.00	0.00	0.00
44	24.46	0.00	0.00	0.00	0.00
45	24.46	0.00	0.00	0.00	0.00
46	24.46	0.00	0.00	0.00	0.00
47	24.46	0.00	0.00	0.00	0.00
48	24.46	0.00	0.00	0.00	0.00
49	24.46	0.00	0.00	0.00	0.00
50	24.46	0.00	0.00	0.00	0.00
Average BER	24.46	0.00	0.00	0.00	0.00
Simulated BER	24.46	0.00	24.41	0.00	0.00
% Difference	0.00	0.00	24.41	0.00	0.00

C.8 12.5 kHz, 1250 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 2:07 P.M.	10/1/2009 2:07 P.M.	10/1/2009 2:07 P.M.	10/1/2009 2:07 P.M.	10/1/2009 2:11 P.M.
Frequency (Hz)	12500				
Bit Rate (bps)	1250				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	0.00	15.97	21.29	16.41	0.00
2	0.05	15.77	20.17	16.21	0.00
3	0.00	15.43	19.97	15.92	0.00
4	0.10	15.92	20.70	16.31	0.00
5	0.00	15.92	20.85	16.21	0.00
6	0.00	15.72	20.07	16.11	0.00
7	0.15	15.63	20.26	16.02	0.00
8	0.15	15.77	20.65	16.21	0.00
9	0.05	15.92	20.85	16.26	0.00
10	0.05	15.82	20.26	16.06	0.00
11	0.15	15.67	20.36	16.16	0.00
12	0.10	15.72	20.70	16.16	0.00
13	0.10	15.92	20.80	16.36	0.00
14	0.10	15.77	20.31	16.16	0.00
15	0.10	15.67	20.51	16.11	0.00
16	0.10	15.63	20.80	16.11	0.00
17	0.05	15.77	20.56	16.16	0.00
18	0.10	15.67	20.36	16.02	0.00
19	0.10	15.67	20.46	16.06	0.00
20	0.10	15.72	20.61	16.16	0.00
21	0.05	15.72	20.70	16.11	0.00
22	0.05	15.72	20.51	16.16	0.00

23	0.05	15.67	20.36	16.06	0.00
24	0.05	15.82	20.70	16.26	0.00
25	0.00	15.72	20.70	16.21	0.00
26	0.00	15.77	20.70	16.21	0.00
27	0.00	15.72	20.56	15.97	0.00
28	0.05	15.77	20.61	16.16	0.00
29	0.00	15.67	20.56	15.97	0.00
30	0.00	15.63	20.61	16.02	0.00
31	0.00	15.72	20.56	16.06	0.00
32	0.05	15.63	20.65	15.97	0.00
33	0.00	15.63	20.70	15.92	0.00
34	0.00	15.72	20.65	16.11	0.00
35	0.00	15.67	20.61	16.11	0.00
36	0.00	15.63	20.56	15.97	0.00
37	0.10	15.63	20.65	15.92	0.00
38	0.00	15.82	20.65	16.11	0.00
39	0.44	16.11	21.14	16.5	0.00
40	0.00	15.63	20.02	15.97	0.00
41	0.00	15.33	19.43	15.72	0.00
42	0.00	15.33	19.43	15.63	0.00
43	0.00	15.23	19.29	15.53	0.00
44	0.00	15.23	19.38	15.53	0.00
45	0.00	15.33	19.43	15.63	0.00
46	0.00	15.19	19.48	15.43	0.00
47	0.00	15.53	19.38	15.87	0.00
48	0.00	15.38	19.43	15.67	0.00
49	0.00	15.28	19.53	15.58	0.00
50	0.00	15.23	19.43	15.48	0.00
Average BER	0.05	15.65	20.34	16.02	0.00
Simulated BER	0.29	14.26	20.02	14.94	0.00
% Difference	0.24	1.39	0.32	1.08	0.00

C.9 12.5 kHz, 2500 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 2:13 P.M.	10/1/2009 2:13 P.M.	10/1/2009 2:13 P.M.	10/1/2009 2:13 P.M.	10/1/2009 2:16 P.M.
Frequency (Hz)	12500				
Bit Rate (bps)	2500				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	7.62	13.04	12.45	13.43	26.37
2	13.43	13.23	12.45	13.33	25.59
3	9.91	13.23	12.45	13.48	25.68
4	12.89	13.13	12.45	13.28	25.68
5	12.01	13.23	12.45	13.43	25.49
6	12.45	13.23	12.45	13.33	24.90

7	10.69	13.23	12.45	13.43	25.10
8	12.74	13.23	12.45	13.48	25.10
9	10.84	13.23	12.45	13.38	25.54
10	12.35	13.23	12.45	13.38	25.93
11	11.08	13.23	12.45	13.48	26.90
12	12.45	13.23	12.45	13.43	12.84
13	12.16	13.23	12.45	13.33	11.87
14	11.18	13.23	12.45	13.48	12.06
15	12.45	13.23	12.45	13.43	11.33
16	11.87	13.23	12.45	13.48	11.18
17	12.30	13.23	12.45	13.33	10.74
18	11.67	13.23	12.45	13.43	10.94
19	12.89	13.23	12.45	13.48	11.47
20	12.06	13.23	12.45	13.48	11.52
21	12.30	13.23	12.45	13.43	11.43
22	12.99	13.23	12.45	13.43	12.11
23	12.50	13.23	12.45	13.48	12.06
24	11.72	13.23	12.45	13.48	11.72
25	12.55	13.23	12.45	13.48	11.96
26	12.55	13.23	12.45	13.48	11.67
27	12.65	13.28	12.45	13.53	11.87
28	12.11	13.28	12.45	13.38	11.87
29	11.82	13.28	12.45	13.53	48.97
30	12.30	13.28	12.45	13.53	49.02
31	12.60	13.28	12.45	13.53	11.04
32	11.91	13.23	12.45	13.48	10.69
33	12.84	13.28	12.45	13.53	10.64
34	12.60	13.28	12.45	13.53	10.79
35	12.65	13.28	12.45	13.53	10.94
36	12.40	13.23	12.45	13.48	11.43
37	12.30	13.28	12.45	13.48	48.58
38	12.50	13.28	12.45	13.53	48.97
39	12.65	13.28	12.45	13.53	48.93
40	13.13	13.28	12.45	13.48	49.02
41	12.16	13.33	12.45	13.53	49.41
42	12.94	13.28	12.45	13.48	49.27
43	12.65	13.28	12.45	13.53	49.22
44	13.33	13.28	12.45	13.43	48.29
45	12.94	13.28	12.45	13.53	48.78
46	13.09	13.28	12.45	13.53	48.44
47	13.33	13.33	12.45	13.53	47.95
48	12.70	13.28	12.45	13.53	30.71
49	12.11	13.33	12.45	13.57	47.71
50	12.74	13.28	12.45	13.53	30.91
Average BER	12.24	13.25	12.45	13.47	25.81
Simulated BER	6.74	12.74	12.45	13.18	39.06
% Difference	5.50	0.51	0.00	0.29	13.25

C.10 12.5 kHz, 3125 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 2:18 P.M.	10/1/2009 2:18 P.M.	10/1/2009 2:18 P.M.	10/1/2009 2:18 P.M.	10/1/2009 2:20 P.M.
Frequency (Hz)	12500				
Bit Rate (bps)	3125				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	42.63	33.25	31.98	34.77	33.11
2	42.63	33.35	32.13	34.81	26.56
3	42.63	33.45	32.03	34.81	27.44
4	42.63	33.50	32.13	34.86	30.08
5	42.63	33.45	32.03	34.81	30.66
6	42.63	33.40	32.28	34.81	30.18
7	42.63	33.45	32.08	34.81	32.28
8	42.63	33.45	32.18	34.81	32.52
9	42.63	33.45	32.08	34.81	32.86
10	42.63	33.45	32.18	34.81	33.74
11	42.63	33.40	32.13	34.81	34.13
12	42.63	33.40	32.18	34.81	34.57
13	42.63	33.50	32.23	34.86	34.62
14	42.63	33.50	32.32	34.81	34.08
15	42.63	33.45	32.23	34.81	34.38
16	42.63	33.45	32.28	34.81	34.77
17	42.63	33.50	32.32	34.86	34.72
18	42.63	33.45	32.32	34.81	34.47
19	42.63	33.40	32.32	34.91	33.69
20	42.63	33.59	32.52	34.86	33.40
21	42.63	33.59	32.57	34.86	32.76
22	42.63	33.59	32.62	34.86	31.88
23	42.63	33.50	32.71	34.86	31.79
24	42.63	33.54	32.62	34.81	31.79
25	42.63	33.50	32.62	34.86	32.32
26	42.63	33.45	32.71	34.86	31.05
27	42.63	33.69	32.62	35.01	32.47
28	42.63	33.59	32.76	34.91	33.74
29	42.63	33.59	32.71	34.96	35.25
30	42.63	33.59	32.86	34.91	36.04
31	42.63	33.59	32.91	34.86	36.28
32	42.63	33.69	32.91	34.96	36.91
33	42.63	33.64	32.86	34.91	37.21
34	42.63	33.69	32.67	35.01	36.87
35	42.63	33.64	32.81	34.91	36.52
36	42.63	33.59	32.91	34.91	36.08
37	42.63	33.64	32.91	34.96	35.94
38	42.63	33.79	32.86	35.06	34.47
39	42.63	33.84	32.91	35.06	35.35

40	42.63	33.79	33.06	35.06	34.52
41	42.63	33.69	32.96	35.01	35.06
42	42.63	33.74	32.91	34.96	34.52
43	42.63	33.84	33.06	35.06	34.62
44	42.63	33.79	33.06	35.01	34.77
45	42.63	33.74	33.01	35.06	34.62
46	42.63	33.74	33.06	34.96	34.28
47	42.63	33.89	33.15	35.06	33.45
48	42.63	33.84	33.15	35.11	33.89
49	42.63	33.89	33.20	35.06	33.25
50	42.63	33.94	33.20	35.11	32.76
Average BER	42.63	33.59	32.61	34.91	33.65
Simulated BER	40.09	31.15	28.61	32.76	0.34
% Difference	2.54	2.44	4.00	2.15	33.31

C.11 17.5 kHz, 250 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 2:24 P.M.	10/1/2009 2:24 P.M.	10/1/2009 2:24 P.M.	10/1/2009 2:24 P.M.	10/1/2009 2:28 P.M.
Frequency (Hz)	17500				
Bit Rate (bps)	250				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00
Average BER	0.00	0.00	0.00	0.00	0.00
Simulated BER	0.00	0.00	0.00	0.00	0.00
% Difference	0.00	0.00	0.00	0.00	0.00

C.12 17.5 kHz, 500 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 2:34 P.M.	10/1/2009 2:34 P.M.	10/1/2009 2:34 P.M.	10/1/2009 2:34 P.M.	10/1/2009 2:39 P.M.
Frequency (Hz)	17500				
Bit Rate (bps)	500				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	0.00	0.29	0.00	1.61	0.00
2	0.00	0.49	0.00	1.71	0.00
3	0.00	0.24	0.00	1.71	0.00
4	0.00	0.49	0.00	1.71	0.00
5	0.00	0.24	0.00	1.76	0.00
6	0.00	0.54	0.00	1.71	0.00
7	0.00	0.20	0.00	1.56	0.00
8	0.00	0.63	0.00	1.86	0.00
9	0.00	0.54	0.00	1.76	0.00
10	0.00	0.49	0.00	1.76	0.00
11	0.00	0.73	0.00	1.86	0.00
12	0.00	0.68	0.00	2.29	0.00
13	0.00	0.44	0.00	2.05	0.00
14	0.00	0.83	0.00	2.05	0.00
15	0.00	0.68	0.00	2.10	0.00
16	0.00	0.73	0.00	2.59	0.00
17	0.00	0.63	0.00	2.39	0.00
18	0.00	0.88	0.00	2.59	0.00
19	0.00	0.78	0.00	2.44	0.00
20	0.00	0.98	0.00	2.98	0.00
21	0.00	0.93	0.00	3.42	0.00
22	0.00	1.03	0.00	3.13	0.00
23	0.00	1.22	0.00	3.32	0.00
24	0.00	1.22	0.00	3.56	0.00
25	0.00	1.22	0.00	3.56	0.00
26	0.00	1.42	0.00	3.22	0.00
27	0.00	1.51	0.00	3.22	0.00
28	0.00	1.81	0.00	3.52	0.00
29	0.00	2.20	0.00	4.79	0.00
30	0.00	2.44	0.00	4.54	0.00
31	0.00	2.54	0.00	4.83	0.00
32	0.00	2.64	0.00	4.79	0.00
33	0.00	2.64	0.00	4.74	0.00
34	0.00	2.98	0.00	5.42	0.00
35	0.00	3.37	0.00	5.18	0.00
36	0.00	3.08	0.00	5.03	0.00
37	0.00	3.17	0.00	5.32	0.00
38	0.00	3.61	0.00	5.42	0.00

39	0.00	3.66	0.00	5.66	0.00
40	0.00	3.96	0.00	6.40	0.00
41	0.00	4.10	0.00	6.45	0.00
42	0.00	4.39	0.00	6.54	0.00
43	0.00	4.15	0.00	6.30	0.00
44	0.00	4.49	0.00	6.74	0.00
45	0.00	4.35	0.00	6.35	0.00
46	0.00	4.93	0.00	6.49	0.00
47	0.00	4.79	0.00	6.54	0.00
48	0.00	4.39	0.00	6.10	0.00
49	0.00	4.49	0.00	6.20	0.00
50	0.00	6.88	0.00	8.20	0.00
Average BER	0.00	2.10	0.00	3.91	0.00
Simulated BER	0.00	0.00	0.00	0.00	0.00
% Difference	0.00	2.10	0.00	3.91	0.00

C.13 17.5 kHz, 1250 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 2:43 P.M.	10/1/2009 2:43 P.M.	10/1/2009 2:43 P.M.	10/1/2009 2:43 P.M.	10/1/2009 2:46 P.M.
Frequency (Hz)	17500				
Bit Rate (bps)	1250				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	24.51	20.46	23.00	19.43	0.00
2	22.51	18.41	21.78	18.26	0.00
3	22.71	18.60	22.12	18.55	0.00
4	23.00	19.19	22.27	19.34	0.00
5	24.46	20.51	22.95	19.38	0.00
6	22.51	18.55	21.88	18.41	0.00
7	22.85	18.65	22.02	18.55	0.00
8	23.00	18.99	22.27	18.99	0.00
9	22.90	19.09	22.22	19.04	0.00
10	22.66	18.85	21.97	18.75	0.00
11	22.80	18.70	22.02	18.60	0.00
12	22.90	18.90	22.17	18.90	0.00
13	22.90	18.85	22.22	18.90	0.00
14	22.80	18.75	22.12	18.70	0.00
15	22.71	18.99	22.17	18.90	0.00
16	22.95	18.95	22.22	18.99	0.00
17	22.90	19.14	22.27	19.24	0.00
18	22.95	18.90	22.17	18.95	0.00
19	22.80	18.99	22.22	19.04	0.00
20	22.90	19.09	22.31	19.14	0.00
21	23.00	18.95	22.22	19.19	0.00
22	22.95	18.85	22.22	19.09	0.00

23	22.95	18.85	22.12	18.95	0.00
24	22.95	18.95	22.12	19.24	0.00
25	23.00	18.80	22.22	19.09	0.00
26	22.90	18.90	22.22	19.04	0.00
27	22.95	18.99	22.17	19.19	0.00
28	23.00	19.14	22.27	19.34	0.00
29	22.95	18.95	22.27	19.29	0.00
30	22.80	19.04	22.22	19.29	0.00
31	22.90	19.34	22.31	19.58	0.00
32	23.00	19.24	22.36	19.48	0.00
33	23.00	19.14	22.36	19.43	0.00
34	22.90	19.34	22.36	19.58	0.00
35	22.95	19.29	22.31	19.53	0.00
36	23.00	19.48	22.51	19.73	0.00
37	23.00	19.48	22.46	19.68	0.00
38	22.90	19.38	22.46	19.63	0.00
39	22.95	19.38	22.41	19.63	0.00
40	23.00	19.48	22.51	19.63	0.00
41	23.00	19.73	22.51	19.78	0.00
42	22.90	19.78	22.51	19.92	0.00
43	23.00	19.68	22.56	19.78	0.00
44	22.90	19.78	22.56	19.92	0.00
45	22.90	19.68	22.51	19.78	0.00
46	22.85	19.63	22.56	19.87	0.00
47	23.00	19.78	22.51	19.73	0.00
48	22.95	19.82	22.56	19.78	0.00
49	22.95	20.02	22.56	20.17	0.00
50	23.00	20.07	22.56	20.07	0.00
Average BER	22.96	19.23	22.32	19.29	0.00
Simulated BER	43.12	46.92	37.55	43.60	0.00
% Difference	20.16	27.69	15.23	24.31	0.00

C.14 17.5 kHz, 2500 bps

Demodulation	FSK (Envelope Detector)		FSK (Quadrature Receiver)		PSK
	Amplitude Comparator	Hard Limiter	Default	Hard Limiter	
Date/Time	10/1/2009 2:48 P.M.	10/1/2009 2:48 P.M.	10/1/2009 2:48 P.M.	10/1/2009 2:48 P.M.	10/1/2009 2:51 P.M.
Frequency (Hz)	17500				
Bit Rate (bps)	2500				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	17.97	21.53	12.45	21.39	6.59
2	18.26	21.48	12.60	21.44	6.59
3	18.26	21.39	12.60	21.29	6.40
4	18.12	21.29	12.55	21.29	6.59
5	18.16	21.48	12.55	21.39	6.59
6	18.12	21.39	12.50	21.29	6.59

7	18.07	21.34	12.50	21.19	5.52
8	18.16	21.39	12.50	21.34	6.59
9	18.16	21.44	12.55	21.39	6.59
10	18.12	21.39	12.45	21.19	6.59
11	18.12	21.34	12.45	21.24	6.59
12	18.12	21.29	12.45	21.14	6.59
13	18.07	21.19	12.55	21.24	6.59
14	18.12	21.48	12.40	21.24	6.59
15	18.12	21.29	12.45	21.29	6.59
16	18.07	21.24	12.45	21.14	6.54
17	18.12	21.34	12.45	21.19	6.25
18	18.07	21.34	12.45	21.19	4.44
19	18.02	21.24	12.40	21.14	2.05
20	18.07	21.19	12.45	20.95	1.61
21	18.07	21.44	12.45	21.34	0.68
22	17.92	21.19	12.45	21.19	1.03
23	17.87	21.19	12.40	21.00	1.27
24	17.97	21.29	12.40	21.14	2.29
25	17.87	21.19	12.40	21.19	15.09
26	17.97	21.14	12.40	21.09	4.88
27	17.97	21.09	12.30	21.09	6.40
28	17.63	20.95	12.35	20.95	6.59
29	17.92	21.00	12.40	20.85	6.59
30	17.82	21.04	12.40	20.85	6.59
31	17.87	20.90	12.40	20.90	6.59
32	18.02	21.04	12.40	20.70	6.59
33	17.77	21.00	12.35	20.90	6.59
34	17.72	20.95	12.35	21.00	6.54
35	17.68	20.90	12.40	20.70	6.59
36	17.87	20.90	12.30	20.70	6.59
37	17.97	20.90	12.30	20.75	6.59
38	17.77	20.85	12.30	20.70	6.45
39	17.77	21.04	12.35	20.80	6.30
40	17.82	20.85	12.35	20.61	5.91
41	17.77	20.85	12.30	20.70	5.62
42	17.68	20.90	12.30	20.70	6.15
43	17.68	20.75	12.35	20.56	6.20
44	17.92	20.70	12.26	20.51	6.45
45	17.92	20.70	12.35	20.41	6.40
46	17.68	20.80	12.35	20.61	6.05
47	17.63	20.80	12.35	20.70	6.25
48	17.58	20.85	12.26	20.70	6.49
49	17.92	20.61	12.26	20.61	6.30
50	17.72	20.65	12.40	20.41	6.54
Average BER	17.94	21.11	12.41	20.99	5.94
Simulated BER	21.97	25.54	15.33	24.71	0.00
% Difference	4.03	4.43	2.92	3.72	5.94

C.15 17.5 kHz, 3500 bps

Demodulation	FSK (Envelope Detector) Amplitude Comparator	FSK (Envelope Detector) Hard Limiter	FSK (Quadrature Receiver) Default	FSK (Quadrature Receiver) Hard Limiter	PSK
Date/Time	10/1/2009 2:53 P.M.	10/1/2009 2:53 P.M.	10/1/2009 2:53 P.M.	10/1/2009 2:53 P.M.	10/1/2009 2:55 P.M.
Frequency (Hz)	17500				
Bit Rate (bps)	2500				
Packet Size (bits)	2048				
Test Number	BER	BER	BER	BER	BER
1	28.08	25.15	24.17	25.15	3.37
2	27.98	25.20	24.17	25.15	2.39
3	28.32	25.20	24.22	25.10	3.47
4	28.03	25.20	24.17	25.05	3.22
5	28.08	25.24	24.22	25.10	3.37
6	28.32	25.20	24.22	25.15	2.20
7	28.03	25.24	24.22	25.15	3.37
8	28.13	25.24	24.22	25.10	4.00
9	28.03	25.20	24.17	25.15	5.32
10	28.03	25.24	24.22	25.05	5.91
11	28.03	25.20	24.17	25.05	4.25
12	28.03	25.10	24.17	25.10	3.81
13	28.03	25.24	24.22	25.05	3.86
14	28.08	25.15	24.17	25.15	4.54
15	28.08	25.24	24.17	25.05	4.15
16	28.03	25.24	24.17	25.10	4.69
17	28.03	25.20	24.17	25.15	4.98
18	28.03	25.24	24.22	25.10	4.69
19	28.03	25.20	24.17	25.15	4.20
20	28.03	25.20	24.17	25.10	4.20
21	28.03	25.20	24.17	25.05	4.39
22	28.08	25.24	24.17	25.10	3.96
23	28.13	25.20	24.17	25.10	3.96
24	28.13	25.15	24.17	25.05	4.30
25	28.03	25.24	24.17	25.05	4.69
26	28.13	25.24	24.17	25.10	3.96
27	28.13	25.24	24.17	25.10	3.66
28	28.13	25.15	24.17	25.10	3.32
29	28.08	25.20	24.17	25.05	3.22
30	28.13	25.20	24.17	25.10	2.88
31	28.08	25.20	24.17	25.10	2.59
32	28.08	25.20	24.17	25.05	2.29
33	28.13	25.15	24.17	25.10	1.90
34	28.13	25.20	24.17	25.05	1.76
35	28.08	25.15	24.17	25.05	1.66
36	28.17	25.15	24.17	25.10	1.66
37	28.03	25.20	24.17	25.10	1.71
38	28.13	25.15	24.17	25.10	1.71
39	28.17	25.20	24.17	25.10	1.76

40	28.08	25.20	24.17	25.05	1.76
41	28.13	25.20	24.17	25.10	1.90
42	28.17	25.15	24.17	25.10	2.34
43	28.13	25.20	24.17	25.10	2.39
44	28.17	25.15	24.17	25.10	2.54
45	28.17	25.20	24.17	25.10	2.93
46	28.13	25.20	24.17	25.10	2.15
47	28.13	25.20	24.17	25.10	2.44
48	28.13	25.20	24.17	25.10	2.29
49	28.13	25.20	24.17	25.10	2.34
50	28.08	25.20	24.17	25.10	2.64
Average BER	28.10	25.20	24.18	25.10	3.22
Simulated BER	28.61	25.29	24.22	25.24	3.22
% Difference	0.51	0.09	0.04	0.15	0.00

Appendix D
Source Code for Chapter 4

D.1 Tunnel Relay Application

```
/**
 *  Author: Brian Borowski
 *  Date created: 04/07/2009
 *  Date last modified: 04/19/2010
 *  Relays datagrams from the OS arriving at the tun device to the UDP port
 *  connected to the Java modem, and vice-versa.
 */

#include "util.h"

#include <arpa/inet.h>
#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <getopt.h>
#include <linux/if_tun.h>
#include <net/if.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>

/* Globals */
int is_client, mtu, peer_port, peer_sock, sock, this_port;
short tun_number;
char *dev_ip_address, *ip_address;
struct sockaddr_in serv_addr, peer_addr;
time_t t1, t2;

/* Probe for tun interface availability. */
int probe_tun(int print_to_stderr) {
    int fd;

    if ((fd = open("/dev/net/tun", O_RDWR)) < 0) {
        if (print_to_stderr) {
            fprintf(stderr, "Error: Cannot open '/dev/net/tun'. "
                            "Is the tun kernel module loaded?\n");
        }
        return -1;
    }
    close(fd);
    return 0;
}

/* Delete tun device. */
int del_dev_tun(int fd, int print_to_stderr) {
```

```
    if (ioctl(fd, TUNSETPERSIST, 0) < 0) {
        if (print_to_stderr) {
            fprintf(stderr, "Error: Cannot delete tun device: %s\n",
                    strerror(errno));
        }
        return -1;
    }
    close(fd);
    return 0;
}

/* Allocate and configure tun device. */
int tun_alloc(int *fd,
              char *tun_dev,
              int tun_dev_size,
              int MTU,
              int print_to_stderr) {
    int tmp_fd, sock_opts;
    struct ifreq ifr_tun;
    struct sockaddr_in addr;

    /* Set up tunnel device */
    memset(&ifr_tun, 0, sizeof(ifr_tun));

    ifr_tun.ifr_flags = IFF_TUN | IFF_NO_PI;
    strncpy(ifr_tun.ifr_name, tun_dev, IFNAMSIZ);

    if ((*fd = open("/dev/net/tun", O_RDWR)) < 0) {
        if (print_to_stderr) {
            fprintf(stderr,
                    "Error: Cannot create tun device '/dev/net/tun': %s\n",
                    strerror(errno));
        }
        return -1;
    }

    if ((ioctl(*fd, TUNSETIFF, (void *)&ifr_tun)) < 0) {
        if (print_to_stderr) {
            fprintf(stderr,
                    "Error: Cannot create tun device (TUNSETIFF): %s\n",
                    strerror(errno));
        }
        close(*fd);
        return -1;
    }

    if (ioctl(*fd, TUNSETPERSIST, 1) < 0) {
        if (print_to_stderr) {
            fprintf(stderr,
                    "Error: Cannot create tun device (TUNSETPERSIST): %s\n",
                    strerror(errno));
        }
        close(*fd);
        return -1;
    }

    if ((tmp_fd = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
        if (print_to_stderr) {
            fprintf(stderr,
                    "Error: Cannot can't create tun device (UDP socket): %s\n",
```

```
                strerror(errno));
    }
    del_dev_tun(*fd, 1);
    return -1;
}

/* Set IP of this end point of tunnel */
memset(&addr, 0, sizeof(addr));
addr.sin_addr.s_addr = inet_addr(dev_ip_address);
addr.sin_family = AF_INET;
memcpy(&ifr_tun.ifr_addr, &addr, sizeof(struct sockaddr));

if (ioctl(tmp_fd, SIOCSIFADDR, &ifr_tun) < 0) {
    if (print_to_stderr) {
        fprintf(stderr,
                "Error: Cannot create tun device (SIOCSIFADDR): %s\n",
                strerror(errno));
    }
    del_dev_tun(*fd, 1);
    close(tmp_fd);
    return -1;
}

if (ioctl(tmp_fd, SIOCGIFINDEX, &ifr_tun) < 0) {
    if (print_to_stderr) {
        fprintf(stderr,
                "Error: Cannot create tun device (SIOCGIFINDEX): %s\n",
                strerror(errno));
    }
    del_dev_tun(*fd, 1);
    close(tmp_fd);
    return -1;
}

if (ioctl(tmp_fd, SIOCGIFFLAGS, &ifr_tun) < 0) {
    if (print_to_stderr) {
        fprintf(stderr,
                "Error: Cannot create tun device (SIOCGIFFLAGS): %s\n",
                strerror(errno));
    }
    del_dev_tun(*fd, 1);
    close(tmp_fd);
    return -1;
}

ifr_tun.ifr_flags |= (IFF_UP | IFF_RUNNING);

if (ioctl(tmp_fd, SIOCSIFFLAGS, &ifr_tun) < 0) {
    if (print_to_stderr) {
        fprintf(stderr,
                "Error: Cannot create tun device (SIOCSIFFLAGS): %s\n",
                strerror(errno));
    }
    del_dev_tun(*fd, 1);
    close(tmp_fd);
    return -1;
}

/* Set MTU */
ifr_tun.ifr_mtu = MTU;
```

```
    if (ioctl(tmp_fd, SIOCSIFMTU, &ifr_tun) < 0) {
        if (print_to_stderr) {
            fprintf(stderr,
                    "Error: Cannot create tun device (SIOCGIFMTU): %s\n",
                    strerror(errno));
        }
        del_dev_tun(*fd, 1);
        close(tmp_fd);
        return -1;
    }

    /* Make tun socket non blocking */
    sock_opts = fcntl(*fd, F_GETFL, 0);
    fcntl(*fd, F_SETFL, sock_opts | O_NONBLOCK);

    strncpy(tun_dev, ifr_tun.ifr_name, tun_dev_size - 1);
    close(tmp_fd);

    return 0;
}

void print_datagram(uint8_t *str, int num_bytes) {
    int index = 0, lines = 0;

    if (str) {
        if (num_bytes > 104)  {
            fprintf(stderr, " ");
        }
        for (index = 0; index < 8; index++) {
            fprintf(stderr, "    %2d", index);
        }
        if (num_bytes < 104)  {
            fprintf(stderr, "\n%02d:", lines);
        } else {
            fprintf(stderr, "\n %02d:", lines);
        }
        for (index = 0; index < num_bytes; index++) {
            fprintf(stderr, " %02x", str[index]);
            if (isprint(str[index])) {
                fprintf(stderr, "(%c)", str[index]);
            } else {
                fprintf(stderr, "( )");
            }
            if ((index + 1) % 8 == 0 && (index + 1) < num_bytes) {
                lines += 8;
                if (num_bytes <= 104 || index >= 103) {
                    fprintf(stderr, "\n%02d:", lines);
                } else {
                    fprintf(stderr, "\n %02d:", lines);
                }
            }
        }
        fprintf(stderr, "\n");
    }
    return;
}

void print_help(FILE *descriptor, char *executable_name) {
    fprintf(descriptor,
            "Usage: %s\n"
```

```
            "   -d <tunnel device IP address>\n"
            "   -n <tunnel number>\n"
            "   -b <port of local binding>\n"
            "   -i <IP address of link layer implementation>\n"
            "   -p <port of link layer implementation>\n"
            "   -m <MTU in bytes>\n",
            executable_name);
}

void parse_args(int argc, char *argv[]) {
    char c;
    char *dev_ip_str = NULL,
         *ip_str = NULL,
         *number_str = NULL,
         *mtu_str = NULL,
         *link_port_str = NULL,
         *local_port_str = NULL,
         *executable_name = argv[0];

    /* Do not allow getopt to display messsages automatically. */
    opterr = 0;
    /* Parse command line arguments with getopt. */
    while ((c = getopt(argc, argv, "b:d:i:m:n:p:h")) != -1) {
        switch(c) {
            case 'b':
                local_port_str = optarg;
                break;
            case 'd':
                dev_ip_str = optarg;
                break;
            case 'i':
                ip_str = optarg;
                break;
            case 'm':
                mtu_str = optarg;
                break;
            case 'n':
                number_str = optarg;
                break;
            case 'p':
                link_port_str = optarg;
                break;
            case 'h':
                print_help(stdout, executable_name);
                exit(EXIT_SUCCESS);
            case '?':
                if (isprint(optopt)) {
                    fprintf(stderr, "Unknown option '-%c'.\n", optopt);
                } else {
                    fprintf(stderr, "Unknown option character'\\x%x'.\n",
                            optopt);
                }
                print_help(stderr, executable_name);
                exit(EXIT_FAILURE);
        }
    }

    if (number_str != NULL) {
        if (!is_integer(number_str)) {
            fprintf(stderr, "Error: Invalid tunnel number '%s'.\n",
```

```
                number_str);
        exit(EXIT_FAILURE);
    }
    tun_number = atoi(number_str);
    if (tun_number < 0) {
        fprintf(stderr, "Error: Tunnel number must be non-negative.\n");
        exit(EXIT_FAILURE);
    }
} else {
    fprintf(stderr, "Error: Tunnel number not specified.\n");
    exit(EXIT_FAILURE);
}

if (dev_ip_str != NULL) {
    if (!is_valid_ip_addr(dev_ip_str)) {
        fprintf(stderr, "Error: Invalid device IP address '%s'.\n",
                dev_ip_str);
        exit(EXIT_FAILURE);
    }
    dev_ip_address = strdup(dev_ip_str);
} else {
    fprintf(stderr, "Error: Device IP address not specified.\n");
    exit(EXIT_FAILURE);
}

if (local_port_str != NULL) {
    if (!is_integer(local_port_str)) {
        fprintf(stderr, "Error: Invalid port number '%s'.\n",
                local_port_str);
        exit(EXIT_FAILURE);
    }
    this_port = atoi(local_port_str);
    if (this_port < 1024 || this_port > 65535) {
        fprintf(stderr, "Error: Port %d not in range 1024..65535.\n",
                this_port);
        exit(EXIT_FAILURE);
    }
} else {
    fprintf(stderr, "Error: Local port number not specified.\n");
    exit(EXIT_FAILURE);
}

if (ip_str != NULL) {
    if (!is_valid_ip_addr(ip_str)) {
        fprintf(stderr, "Error: Invalid IP address '%s'.\n", ip_str);
        exit(EXIT_FAILURE);
    }
    ip_address = strdup(ip_str);
} else {
    fprintf(stderr, "Error: Link layer IP address not specified.\n");
    exit(EXIT_FAILURE);
}

if (link_port_str != NULL) {
    if (!is_integer(link_port_str)) {
        fprintf(stderr, "Error: Invalid port number '%s'.\n",
                link_port_str);
        exit(EXIT_FAILURE);
    }
    peer_port = atoi(link_port_str);
```

```
        if (peer_port < 1024 || peer_port > 65535) {
            fprintf(stderr, "Error: Port %d not in range 1024..65535.\n",
                    peer_port);
            exit(EXIT_FAILURE);
        }
    } else {
        fprintf(stderr, "Error: Link layer port number not specified.\n");
        exit(EXIT_FAILURE);
    }

    if (mtu_str != NULL) {
        if (!is_integer(mtu_str)) {
            fprintf(stderr, "Error: Invalid MTU '%s'.\n", mtu_str);
            exit(EXIT_FAILURE);
        }
        mtu = atoi(mtu_str);
        if (mtu < 1) {
            fprintf(stderr, "Error: MTU must be a positive number.\n");
            exit(EXIT_FAILURE);
        }
    } else {
        fprintf(stderr, "Error: MTU not specified.\n");
        exit(EXIT_FAILURE);
    }
}

void print_params() {
    printf("Using tunnel parameters:\n");
    printf("  Tunnel #          : %d\n", tun_number);
    printf("  Tunnel IP Address : %s\n", dev_ip_address);
    printf("  Local Port #      : %d\n", this_port);
    printf("  MTU (bytes)       : %d\n", mtu);
    printf("  Peer IP Address   : %s\n", ip_address);
    printf("  Peer Port #       : %d\n", peer_port);
}

void analyze_datagram(uint8_t *datagram) {
    uint8_t protocol;
    uint16_t value;
    int i;

    protocol = datagram[9];
    printf("Datagram Contents\n");
    printf("  Protocol: %d ", protocol);
    if (protocol == 6) {
        printf("(TCP)\n");
    } else if (protocol == 17) {
        printf("(UDP)\n");
    }
    printf("  Source address: ");
    for (i = 12; i <= 15; i++) {
        printf("%d", datagram[i]);
        if (i != 15) {
            printf(".");
        } else {
            printf("\n");
        }
    }
    printf("  Destination address: ");
    for (i = 16; i <= 19; i++) {
```

```
            printf("%d", datagram[i]);
            if (i != 19) {
                printf(".");
            } else {
                printf("\n");
            }
        }
        value = (datagram[20] << 8) | datagram[21];
        printf("  Source port: %d\n", value);
        value = (datagram[22] << 8) | datagram[23];
        printf("  Destination port: %d\n", value);
        if (protocol == 17) {
            value = (datagram[24] << 8) | datagram[25];
            printf("  Length: %d\n", value);
            value = (datagram[26] << 8) | datagram[27];
            printf("  Checksum: %d\n", value);
        } else if (protocol == 6) {
            value = (datagram[33] & 0x20) >> 5;
            printf("  URG: %d, ", value);
            value = (datagram[33] & 0x10) >> 4;
            printf("ACK: %d, ", value);
            value = (datagram[33] & 0x8) >> 3;
            printf("PSH: %d, ", value);
            value = (datagram[33] & 0x4) >> 2;
            printf("RST: %d, ", value);
            value = (datagram[33] & 0x2) >> 1;
            printf("SYN: %d, ", value);
            value = (datagram[33] & 0x1);
            printf("FIN: %d\n", value);
        }
    }

    void config_tun_link() {
        if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
            perror("Error");
            exit(EXIT_FAILURE);
        }
        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(this_port);
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        if (bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
            perror("Error");
            exit(EXIT_FAILURE);
        }
        memset(&peer_addr, 0, sizeof(peer_addr));
        peer_addr.sin_family = AF_INET;
        peer_addr.sin_port = htons(peer_port);
        peer_addr.sin_addr.s_addr = inet_addr(ip_address);
    }

    int main(int argc, char *argv[]) {
        char c, tun_dev[IFNAMSIZ];
        fd_set socket_set;
        int bytes_read, bytes_sent, max_fd, return_code, running, timeout, tun_fd;
        socklen_t peer_len;
        struct timeval select_timeout;
        uint8_t *buffer;

        parse_args(argc, argv);
```

```
print_params();
config_tun_link();

sprintf(tun_dev, "tun%d", tun_number);
timeout = 5;
if ((return_code = probe_tun(1)) < 0) {
    return 1;
}
printf("Status: tun file descriptor probed successfully.\n");
if ((return_code = tun_alloc(&tun_fd, tun_dev, IFNAMSIZ, mtu, 1)) < 0) {
    return 1;
}
printf("Status: '%s' allocated successfully to fd %d.\n", tun_dev, tun_fd);
max_fd = max(max(STDIN_FILENO, tun_fd), sock);

if ((buffer = (uint8_t*)malloc(mtu * sizeof(uint8_t))) == NULL) {
    fprintf(stderr, "Error: Cannot allocate memory for buffer.\n");
    del_dev_tun(tun_fd, 1);
}

printf("Starting server. Type 'q' + ENTER to quit...\n");
running = 1;
while (running) {
    FD_ZERO(&socket_set);
    FD_SET(STDIN_FILENO, &socket_set);
    FD_SET(tun_fd, &socket_set);
    FD_SET(sock, &socket_set);

    select_timeout.tv_sec = timeout;
    select_timeout.tv_usec = 0;

    if (select(max_fd + 1, &socket_set, NULL, NULL, NULL) == 0) {
        printf("No activity for %d seconds. Server still alive.\n",
               timeout);
    } else if (FD_ISSET(tun_fd, &socket_set)) {
        t2 = time(NULL);
        printf("Elapsed time since last datagram: %.2f seconds\n",
               difftime(t2, t1));
        t1 = t2;
        printf("Received data from %s.\n", tun_dev);
        if ((bytes_read = read(tun_fd, buffer, mtu)) < 0) {
            fprintf(stderr, "Error: Cannot read from %s: ", tun_dev);
            perror("");
            continue;
        }
        if (bytes_read > 0) {
            print_datagram(buffer, bytes_read);
            analyze_datagram(buffer);
            if ((bytes_sent = sendto(sock,
                                     buffer,
                                     bytes_read,
                                     0,
                                     (struct sockaddr *)&peer_addr,
                                     sizeof(peer_addr))) != bytes_read) {
                fprintf(stderr, "Error: Sent %d bytes instead of %d.\n",
                        bytes_sent, bytes_read);
            } else {
                fprintf(stderr, "Status: Sent %d bytes.\n", bytes_sent);
            }
        }
```

```
        } else if (FD_ISSET(sock, &socket_set)) {
            printf("Received data from UDP socket.\n");
            peer_len = sizeof(peer_addr);
            if ((bytes_read = recvfrom(sock,
                                       buffer,
                                       mtu,
                                       0,
                                       (struct sockaddr *)&peer_addr,
                                       &peer_len)) < 0) {
                fprintf(stderr, "Error: Cannot read from UDP socket: ");
                perror("");
                continue;
            }
            if (bytes_read > 0) {
                printf("Handling connection with %s.\n",
                       inet_ntoa(peer_addr.sin_addr));
                print_datagram(buffer, bytes_read);
                analyze_datagram(buffer);
                if ((bytes_sent = write(tun_fd,
                                        buffer,
                                        bytes_read)) != bytes_read) {
                    fprintf(stderr, "Error: Sent %d bytes instead of %d.\n",
                            bytes_sent, bytes_read);
                } else {
                    fprintf(stderr, "Status: Sent %d bytes.\n", bytes_sent);
                }
            }
        } else if (FD_ISSET(STDIN_FILENO, &socket_set)) {
            c = getchar();
            if (c == 'q') {
                printf("Shutting down server.\n");
                running = 0;
            }
        }
    }
    del_dev_tun(tun_fd, 1);
    free(buffer);
    return 0;
}
```

D.2 util.h for Tunnel Relay Application

```
#ifndef UTIL_H_
#define UTIL_H_

/**
 * Definitions
 */
#define ERR_TOO_MANY_DOTS       -1
#define ERR_TOO_FEW_DOTS        -2
#define ERR_OCTET_TOO_LONG      -4
#define ERR_OCTET_OUT_OF_RANGE  -8
#define ERR_OCTET_NOT_INTEGER  -16

#define max(A, B) ((A) > (B) ? (A) : (B))

/**
 * Function prototypes
```

```
 */
int is_valid_ip_addr(char *ipAddressPtr);
int is_integer(char *inputPtr);

#endif
```

D.3 util.c for Tunnel Relay Application

```
#include "util.h"

#include <ctype.h>
#include <stdlib.h>
#include <string.h>

/**
 * Returns 0 if the character array represents a valid IPv4 address; otherwise,
 * returns a non-negative error code.
 */
int is_valid_ip_addr(char *ipAddressPtr) {
    unsigned char dotPositions[4];
    char buffer[3];
    unsigned int j;
    int octet;
    int length;
    int i, k;
    int index;

    length = strlen(ipAddressPtr);
    index = 0;
    for (i = 0; i < length; i++) {
        if (ipAddressPtr[i] == '.') {
            if (index < 3) {
                dotPositions[index++] = i;
            } else {
                return ERR_TOO_MANY_DOTS;
            }
        }
    }
    if (index < 3) {
        return ERR_TOO_FEW_DOTS;
    }
    dotPositions[3] = length;
    index = 0;
    for (i = 0; i < 4; i++) {
        k = 0;
        for (j = index; j < dotPositions[i]; j++) {
            if (k < 3) {
                buffer[k++] = ipAddressPtr[j];
            } else {
                return ERR_OCTET_TOO_LONG;
            }
        }
        buffer[k] = '\0';
        index = j + 1;
        if (strlen(buffer) == 1 && buffer[0] == '0') {
            continue;
        }
        if (is_integer(buffer)) {
```

```
            octet = atoi(buffer);
            if (octet <= 0 || octet > 255) {
                return ERR_OCTET_OUT_OF_RANGE;
            }
        } else {
            return ERR_OCTET_NOT_INTEGER;
        }
    }
    return 1;
}

/**
 * Determines whether or not a character array represents an integer.
 */
int is_integer(char *inputPtr) {
    int start, i;

    if (inputPtr[0] == '-') {
        if (inputPtr[1] == '\0') {
            return 0;
        } else {
            start = 1;
        }
    } else {
        start = 0;
    }
    for (i = start; inputPtr[i] != '\0'; i++) {
        if (!isdigit(inputPtr[i])) {
            return 0;
        }
    }
    return 1;
}
```

D.4 SignalProcessor.java for Softwater Modem

```
/**
 * Author: Brian Borowski
 * Date created: 06/23/2009
 * Date last modified: 06/23/2009
 * Computes FFT, cross-correlation, convolution, and rms amplitude.
 * Based on W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery,
 *     Numerical Recipes: The Art of Scientific Computing, Third Edition,
 *     Cambridge University Press: Cambridge, 2007.
 */

public final class SignalProcessor {

    public static int IFFT = -1, FFT = 1, DECONVOLVE = -1, CONVOLVE = 1;
    private float[] ans, temp;
    private int fftPoints;

    public SignalProcessor(int length1, int length2) {
        int max = Math.max(length1, length2),
            upperBound = (max << 1) - 1;
        fftPoints = 2;
        while (fftPoints <= upperBound) {
            fftPoints <<= 1;
```

```
        }
        ans = new float[fftPoints];
        temp = new float[fftPoints];
    }

    public static void realft(float[] data, int n, int isign) {
        int i, i1, i2, i3, i4, np3;
        float c1 = 0.5f, c2, h1r, h1i, h2r, h2i;
        double wr, wi, wpr, wpi, wtemp, theta;

        theta = 3.141592653589793 / (double)(n >> 1);
        if (isign == FFT) {
            c2 = -0.5f;
            four1(data, n >> 1, 1);
        } else {
            c2 = 0.5f;
            theta = -theta;
        }
        wtemp = Math.sin(0.5 * theta);
        wpr = -2.0 * wtemp * wtemp;
        wpi = Math.sin(theta);
        wr = 1.0 + wpr;
        wi = wpi;
        np3 = n + 3;
        int upperBound = n >> 2;
        for (i = 2; i <= upperBound; i++) {
            i2 = 1 + (i1 = i + i);
            i4 = 1 + (i3 = n - i1);
            h1r = c1 * (data[i1] + data[i3]);
            h1i = c1 * (data[i2] - data[i4]);
            h2r = -c2 * (data[i2] + data[i4]);
            h2i = c2 * (data[i1] - data[i3]);
            data[i1] = (float)(h1r + wr * h2r - wi * h2i);
            data[i2] = (float)(h1i + wr * h2i + wi * h2r);
            data[i3] = (float)(h1r - wr * h2r + wi * h2i);
            data[i4] = (float)(-h1i + wr * h2i + wi * h2r);
            wr = (wtemp = wr) * wpr - wi * wpi + wr;
            wi = wi * wpr + wtemp * wpi + wi;
        }
        if (isign == FFT) {
            data[0] = (h1r = data[0]) + data[1];
            data[1] = h1r - data[1];
        } else {
            data[0] = c1 * ((h1r = data[0]) + data[1]);
            data[1] = c1 * (h1r - data[1]);
            four1(data, n >> 1, -1);
        }
    }

    public float[] crossCorrelate(float[] d1, float[] d2) {
        int no2, i,
            len1 = d1.length,
            len2 = d2.length,
            n = fftPoints;

        // Make copies of the data so that the original samples remain
        // preserved.
        System.arraycopy(d1, 0, ans, 0, len1);
        System.arraycopy(d2, 0, temp, 0, len2);
        // Pad with zeros.
```

```
        for (i = len1; i < n; i++) {
            ans[i] = 0.0f;
        }
        for (i = len2; i < n; i++) {
            temp[i] = 0.0f;
        }
        realft(ans, n, FFT);
        realft(temp, n, FFT);
        no2 = n >> 1;
        for (i = 2; i < n; i+=2) {
            int iPlusOne = i + 1;
            float ansIPlusOne = ans[iPlusOne],
                  ansI = ans[i],
                  tempIPlusOne = temp[iPlusOne],
                  tempI = temp[i];
            ans[i] = (ansI * tempI + ansIPlusOne * tempIPlusOne) / no2;
            ans[iPlusOne] = (ansIPlusOne * tempI - ansI * tempIPlusOne) / no2;
        }
        ans[0] = ans[0] * temp[0]/no2;
        ans[1] = ans[1] * temp[1]/no2;
        realft(ans, n, IFFT);
        return ans;
    }

    public float[] convolve(float[] data, float[] response, int isign) {
        int no2, i,
            len1 = data.length,
            len2 = response.length,
            n = fftPoints;
        float val, max = 0.0f;
        short maxShort = Short.MAX_VALUE;

        // Make copies of the data so that the original samples remain
        // preserved.
        System.arraycopy(data, 0, ans, 0, len1);
        System.arraycopy(response, 0, temp, 0, len2);
        // Pad with zeros.
        for (i = len1; i < n; i++) {
            ans[i] = 0.0f;
        }
        for (i = len2; i < n; i++) {
            temp[i] = 0.0f;
        }
        realft(ans, n, FFT);
        realft(temp, n, FFT);
        no2 = n >> 1;
        if (isign == CONVOLVE) {
            for (i = 2; i < n; i+=2) {
                int iPlusOne = i + 1;
                float ansIPlusOne = ans[iPlusOne],
                      ansI = ans[i],
                      tempIPlusOne = temp[iPlusOne],
                      tempI = temp[i];
                ans[i] = (ansI * tempI - ansIPlusOne * tempIPlusOne) / no2;
                ans[iPlusOne] =
                    (ansIPlusOne * tempI + ansI * tempIPlusOne) / no2;
            }
            ans[0] = ans[0] * temp[0]/no2;
            ans[1] = ans[1] * temp[1]/no2;
        } else if (isign == DECONVOLVE) {
```

```
            // TODO - but not needed in project
        } else {
            return null;
        }
        realft(ans, n, IFFT);
        for (i = n - 1; i >= 0; i--) {
            val = ans[i];
            val = val > 0 ? val : -val;
            if (val > max) {
                max = val;
            }
        }
        for (i = n - 1; i >= 0; i--) {
            ans[i] = ans[i] / max * maxShort;
        }
        return ans;
    }

    public static float computeRmsAmplitude(short[] samples,
                                            int offset,
                                            int howMany) {
        long rms = 0;
        int upperBound = offset + howMany;
        for (int i = offset; i < upperBound; i++) {
            short val = samples[i];
            rms += (val * val);
        }
        return (float)Math.sqrt((float)rms / howMany);
    }

    public static float indexToFrequency(int numSamples,
                                         int index,
                                         int samplingRate) {
        if (index >= numSamples) return 0.0f;
        if (index <= numSamples/2) {
            return (float)index * samplingRate / (float)numSamples;
        }
        return (float)-(numSamples - index * samplingRate) / (float)numSamples;
    }

    private static void four1(float[] data, int n, int isign) {
        int nn, mmax, m, j, istep, i;
        double wtemp, wr, wpr, wpi, wi, theta;
        float tempr, tempi;

        if (n < 2 || (n & (n-1)) != 0) {
            return;
        }
        nn = n << 1;
        j = 1;
        for (i = 1; i < nn; i+=2) {
            if (j > i) {
                int jMinusOne = j - 1,
                    iMinusOne = i - 1;
                tempr = data[jMinusOne];
                data[jMinusOne] = data[iMinusOne];
                data[iMinusOne] = tempr;
                tempr = data[j];
                data[j] = data[i];
                data[i] = tempr;
```

```
            }
            m = n;
            while (m >= 2 && j > m) {
                j -= m;
                m >>= 1;
            }
            j += m;
        }
        mmax = 2;
        while (nn > mmax) {
            istep = mmax << 1;
            theta = isign * (6.28318530717959 / mmax);
            wtemp = Math.sin(0.5 * theta);
            wpr = -2.0 * wtemp * wtemp;
            wpi = Math.sin(theta);
            wr = 1.0;
            wi = 0.0;
            for (m = 1; m < mmax; m+=2) {
                for (i = m; i <= nn; i+=istep) {
                    j = i + mmax;
                    int jMinusOne = j - 1,
                        iMinusOne = i - 1;
                    float dataJMinusOne = data[jMinusOne],
                          dataJ = data[j];
                    tempr = (float)(wr * dataJMinusOne - wi * dataJ);
                    tempi = (float)(wr * dataJ + wi * dataJMinusOne);
                    data[jMinusOne] = data[iMinusOne] - tempr;
                    data[j] = data[i] - tempi;
                    data[iMinusOne] += tempr;
                    data[i] += tempi;
                }
                wr = (wtemp=wr) * wpr - wi * wpi + wr;
                wi = wi * wpr + wtemp * wpi + wi;
            }
            mmax = istep;
        }
    }
}
```

D.5 LevinsonDurbin.java for Softwater Modem

```
/**
 * Author: Brian Borowski
 * Date created: 06/26/2009
 * Date last modified: 06/26/2009
 * Inverts an impulse response in the time domain.
 * Java port based on 'http://www.musicdsp.org/showone.php?id=188'
 * Original author: Bob Cain, May 1, 2001 arcane[AT]arcanemethods[DOT]com
 */

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Vector;
```

```
public final class LevinsonDurbin {

    private final float[] b, hinv, m, samples;
    private final float[][] a;
    private final int length;
    private final SignalProcessor signalProcesor;

    public LevinsonDurbin(final int length) {
        this.length = length;
        final int numberOfSamples = length << 1;
        samples = new float[numberOfSamples];
        b       = new float[length];
        m       = new float[length];
        hinv    = new float[length];
        a       = new float[length + 1][2];
        signalProcesor = new SignalProcessor(numberOfSamples, numberOfSamples);
    }

    public float[] getInverse(final float[] impulseResponse, final int delay) {
        for (int i = length - 1; i >= 0; --i) {
            samples[i] = impulseResponse[i];
        }
        final int upperBound = length << 1;
        for (int i = length; i < upperBound; ++i) {
            samples[i] = 0.0f;
        }
        final float[] corr = signalProcesor.crossCorrelate(samples, samples);
        for (int i = length - 1; i >= 0; --i) {
            m[i] = corr[i];
        }
        for (int i = delay; i >= 0; --i) {
            b[i] = samples[i];
        }
        for (int i = delay + 1; i < length; ++i) {
            b[i] = 0;
        }
        return solveToeplitz(m, b);
    }

    private float[] solveToeplitz(final float[] r, final float[] q) {
        final int n = length;
        for (int row = n; row >= 0; --row) {
            for (int col = 1; col >= 0; --col) {
                a[row][col] = 0.0f;
            }
        }
        for (int row = n - 1; row >= 0; --row) {
            hinv[row] = 0.0f;
        }
        a[0][0] = 1.0f;

        hinv[0] = q[0]/r[0];

        float alpha = r[0];
        int c = 0,
            d = 1;

        for (int k = 1; k < n; ++k) {
            a[k][c] = 0;
            a[0][d] = 1.0f;
```

```
            float beta = 0.0f;
            for (int j = 1; j <= k; ++j) {
                beta += r[k+1-j]*a[j-1][c];
            }
            beta /= alpha;
            for (int j = 1; j <= k; ++j) {
                a[j][d] = a[j][c] - beta*a[k-j][c];
            }
            alpha *= (1 - beta*beta);
            hinv[k] = q[k];
            for (int j = 1; j <= k; ++j) {
                hinv[k] -= r[k+1-j]*hinv[j-1];
            }
            hinv[k] /= alpha;
            for (int j = 1; j <= k; ++j) {
                hinv[j-1] += a[k+1-j][d]*hinv[k];
            }
            int temp = c;
            c = d;
            d = temp;
        }
        return hinv;
    }

    public static void main(String[] args) {
        if (args.length != 2) {
            System.err.println(
                "Usage: java LevinsonDurbin <input file> <output file>");
            System.exit(1);
        }
        Vector<Float> v = new Vector<Float>();
        String infile  = args[0],
               outfile = args[1];
        BufferedReader in = null;
        String str = null;
        int line = 1;
        try {
            in = new BufferedReader(new FileReader(infile));
            while ((str = in.readLine()) != null) {
                v.add(new Float(str));
                line++;
            }
        } catch (NumberFormatException nfe) {
            System.err.println(
                "LevinsonDurbin: Invalid float '" + str + "' at line "
                + line + ".");
            System.exit(1);
        } catch (FileNotFoundException fnfe) {
            System.err.println(
                "LevinsonDurbin: File '" + infile + "' not found.");
            System.exit(1);
        } catch (IOException ioe) {
            System.err.println(
                "LevinsonDurbin: Error reading file '" + infile + "'.");
            System.exit(1);
        } finally {
            try {
                if (in != null) {
                    in.close();
                }
```

```
            } catch (IOException ioe) { }
        }
        int length = v.size();
        float[] data = new float[length << 1];
        for (int i = 0; i < length; i++) {
            data[i] = v.elementAt(i);
        }
        LevinsonDurbin levinsonDurbin = new LevinsonDurbin(length);

        long startTime = System.currentTimeMillis(), currentTime;
        float[] inverseIR = levinsonDurbin.getInverse(data, 0);
        currentTime = System.currentTimeMillis();
        System.out.println(
            "Computation time: " + (currentTime - startTime)/1000.0f + " s");

        BufferedWriter out = null;
        try {
            out = new BufferedWriter(new FileWriter(outfile));
            for (int i = 0; i < length; i++) {
                out.append(Float.toString(inverseIR[i]));
                out.newLine();
            }
        } catch (IOException ioe) {
            System.err.println(
                "LevinsonDurbin: Error writing '" + outfile + "'.");
        } finally {
            try {
                if (out != null) {
                    out.close();
                }
            } catch (IOException ioe) { }
        }
    }
}
```

References

[Aik 2006] T. B. Aik, Q. S. Sen, Z. Nan, "Characterization of Multipath Acoustic Channels in Very Shallow Waters for Communications," in Proc. OCEANS 2006, pp. 1–8, Singapore, 2006.

[Akyildiz 2005] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater Acoustic Sensor Networks: Research Challenges," Ad Hoc Networks (Elsevier), 3(3):257–279, May 2005.

[Akyildiz 2006] I. F. Akyildiz, D. Pompili, and T. Melodia, "State-of-the-Art in Protocol Research for Underwater Acoustic Sensor Networks," in Proc. WUW-Net'06, 2006.

[Al-Shamma'a 2004] A. I. Al-Shamma'a, A. Shaw., and S Saman, "Propagation of Electromagnetic Waves at MHz Frequencies through Seawater," IEEE Transactions on Antennas and Propagation, vol. 52, pp. 2843–2849, 2004.

[Aqua-Sim 2010] Aqua-Sim, Accessed online: http://ubinet.engr.uconn.edu/mediawiki/index.php/Aqua-Sim, Mar. 2010.

[Au 1998] W. W. L. Au and K. Banks, "The Acoustics of the Snapping Shrimp Synalpheus Parneomeris in Kaneohe Bay," Journal of the Acoustical Society of America 103(1), pp. 41–47, 1998.

[BELLHOP 2010] BELLHOP ray tracing program, Accessed online: http://oalib.hlsresearch.com/Rays/index.html. Last updated Feb. 12, 2010.

[Bello 1963] P. Bello, "Characterization of randomly time-invariant channels," IEEE Trans. Commun. Syst., Vol. CS-11, pp. 361–393, Dec. 1963.

[Benthos 2010] Benthos, "Acoustic telesonar modems, transducers, surface, undersea – Topside options." Accessed online: http://www.benthos.com/acoustic-telesonar-modem-product-comparison.asp, Mar. 2010.

[Bhattacharyya 1943] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society*, Vol. 35, pp. 99–109, 1943.

[Borowski 2008] B. Borowski, A. Sutin, H.-S. Roh, and B. Bunin, "Passive Acoustic Threat Detection in Estuarine Environments," in Proc. of SPIE Vol. 6945, Mar. 2008.

[Borowski 2009] B. Borowski and D. Duchamp, “Short Paper: The Softwater Modem - A Software Modem for Underwater Acoustic Communication,” in Proc. WUWNet’09, 2009.

[Brekhovskikh 2003] L. M. Brekhovskikh and Y. P. Lysanov, *Fundamentals of Ocean Acoustics, Third Edition*, Springer, 2003.

[Butler 1987] L. Butler, “Underwater Radio Communication,” *Amateur Radio*, Apr. 1987. Published online: http://www.qsl.net/vk5br/UwaterComms.htm.

[Cella 2009] U. M. Cella, R. Johnstone, N. Shuley, “Electromagnetic Wave Wireless Communication in Shallow Water Coastal Environment: Theoretical Analysis and Experimental Results,” in Proc. WUWNet’09, 2009.

[Chitre 2004] M. Chitre, J. Potter, and O. S. Heng, “Underwater Acoustic Channel Characterization for Medium-Range Shallow Water Communications,” in Proc. OCEANS 2004, Vol. 1, pp. 40–45, Nov. 2004.

[Chitre 2008] M. Chitre, S. Shahabudeen, L. Freitag, and M. Stojanovic, “Recent Advances in Underwater Acoustic Communications & Networking,” in Proc. MTS/IEEE OCEANS 2008, Quebec City, Canada, Sept. 2008.

[Coates 1989] R. Coates, *Underwater Acoustic Systems*, New York: Wiley, 1989.

[Comaniciu 2003] D. Comaniciu, V. Ramesh, and P. Meer, “Kernel-Based Object Tracking,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 25, No. 5, pp. 564–577, May 2003.

[Cook 1998] G. Cook and A. Zaknich, “Chirp Sounding the Shallow Water Acoustic Channel,” in Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 4, pp. 2521–2524, Seattle, WA, May 1998.

[Dessalermos 2005] S. Dessalermos, “Undersea Acoustic Propagation Channel Estimation,” Master's Thesis, Naval Postgraduate School, Monterey, CA, 2005.

[Diamant 2005] R. Diamant and L. Chorev, “Emulation System for Underwater Acoustic Channel,” UDT Europe convention, Jun. 2005.

[DSPComm 2010] DSPComm, “AquaComm: Underwater wireless modem,” Accessed online: http://www.dspcomm.com/products_aquacomm.html, Apr. 2010.

[Ettus 2010] Ettus Research LLC, “Downloads”, Accessed online: http://www.ettus.com/download, Apr. 2010.

[FFTW 2010] FFTW, “Fastest Fourier Transform in the West,” Accessed online: http://www.fftw.org/, Apr. 2010.

[Freitag 2005] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball, "The WHOI Micro-Modem: An Acoustic Communications and Navigation System for Multiple Platforms," Proc. IEEE/MTS OCEANS Conf. Exhib., Washington, D.C., Sept. 2005.

[Freitag 2009] L. Freitag and S. Singh, "Performance of Micro-Modem PSK Signaling Under Variable Conditions during the 2008 RACE and SPACE Experiments," in Proc. IEEE OCEANS 2009, Biloxi, Oct. 2009.

[Fu 2006] T. Fu, D. Doonan, C. Utley, R. Iltis, R. Kastner, and H. Lee, "Design and Development of a Software-Defined Underwater Acoustic Modem for Sensor Networks for Environmental and Ecological Research," in Proc. OCEANS 2006, Sept. 2006.

[FuNLab 2010] NS2 UAN Simulator, Accessed online: http://ee.washington.edu/research/funlab/uan/uansim.html, Mar. 2010.

[GEBCO 2010] The General Bathymetric Chart of the Oceans, Accessed online: http://www.gebco.net, Mar. 2010.

[GNU 2010] GNU Radio: the gnu software radio, Accessed online http://gnuradio.org/redmine/wiki/gnuradio, Feb. 2010.

[Goldsmith 2005] A. Goldsmith, *Wireless Communications*, Cambridge University Press: Cambridge, 2005.

[Grund 2006] M. Grund, L. Freitag, J. Preisig, K. Ball, "The PLUSNet Underwater Communications System: Acoustic Telemetry for Undersea Surveillance," in Proc. OCEANS 2006, pp. 1–5, Sept. 2006.

[Guerra 2009] F. Guerra, P. Casari, and M. Zorzi, "World Ocean Simulation System (WOSS): A Simulation Tool for Underwater Networks," in Proc. WUWNet'09, Nov. 2009.

[GulfBase 2008] GulfBase, "South Florida Ocean Measurement Center (SFOMC)," Accessed online: http://www.gulfbase.org/organization/view.php?oid=sfomc, Oct. 2008.

[Hanson 2008] F. Hanson and S. Radic, "High Bandwidth Underwater Optical Communication," Applied Optics, Vol. 47, No. 2, Jan. 2008.

[Harris 2007] A. F. Harris III and M. Zorzi, "Modeling the Underwater Acoustic Channel in ns2," NSTools '07, Nantes, France, Oct. 2007.

[Hwang 2003] J.-K. Hwang, "Innovative communication design lab based on PC sound card and Matlab: a software-defined-radio OFDM modem example," in Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Vol. 3, pp. 761–4, Apr. 2003.

[ITC 2010] International Transducer Corp ITC-6050C Preamplified Hydrophone Specifications, Accessed online: http://www.itc-transducers.com/itc_page.asp?productID=29, Jan. 2010.

[Jacobsen 2008] E. Jacobsen, "Re: Reed-Solomon error correction capacity, 2008-11-07", DSPRelated.com, Accessed online: http://www.dsprelated.com/showmessage/105082/1.php, Nov. 2008.

[Jakes 1975] W. C. Jakes (ed.), *Microwave Mobile Communications*. John Wiley & Sons: New York, 1975.

[Jones 1963] J. Jones, "Hard-Limiting of Two Signals in Random Noise," IEEE Trans. on Information Theory, Vol. 9, Iss. 1, pp. 34–42, Jan. 1963.

[JRat 2010] JRat, The Java Runtime Analysis Toolkit. Accessed online: http://jrat.sourceforge.net/, Apr. 2010.

[Kang 2009] T. Kang, H. Song, and W. S. Hodgkiss, "OFDM Underwater Acoustic Communications in KAM08," in Proc. WUWNet'09, Nov. 2009.

[Kärkkäinen 2007] K. Kärkkäinen. Phase Optimized PN Code Sets for Numerical Analysis and Simulation of DS-CDMA Systems, Accessed online: http://www.ee.oulu.fi/~kk/optim_codes_info.html. Last updated Jan. 17, 2007.

[Kim 2009] S.-M. Kim, S.-H. Byun, S.-G. Kim, and Y.-K. Lim, "Experimental Analysis of Stastical Properties of Underwater Channel in a Very Shallow Water Using Narrow and Broadband Signals," in Proc. IEEE Oceans 2009, Biloxi, Oct. 2009.

[Krasnyansky 2010] M. Krasnyansky, "Universal TUN/TAP Driver," Accessed online: http://vtun.sourceforge.net/tun/, Apr. 2010.

[Kullback 1951] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, Vol. 22, No. 1, pp.79–86, 1951.

[LinkQuest 2010] LinkQuest, "Underwater Acoustic Modem Models," Accessed online: http://www.link-quest.com/html/uwm_hr.pdf, Apr. 2010.

[Linnartz 2009] Jean-Paul M.G. Linnartz (ed.), "Comparing Rician and Nakagami Fading," Wireless Communication Reference Website, Accessed online:

http://www.wirelesscommunication.nl/reference/chaptr03/ricenaka/ricenaka.htm, 2009.

[Linnartz 2009a] Jean-Paul M.G. Linnartz (ed.), "Rician fading," Wireless Communication Reference Website, Accessed online: http://www.wirelesscommunication.nl/reference/chaptr03/ricepdf/rice.htm, 2009.

[Loubet 1993] G. Loubet and G. Jourdain, "Characterization of the Underwater Medium as an Acoustical Horizontal Transmission Channel," in Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 1, pp. 329–332, Apr. 1993.

[Marrow 2002] R. Marrow, *Bluetooth Operation and Use*, McGraw-Hill Professional, 2002.

[Michalopoulou 2001] Z. Michalopoulou, "Estimating the Impulse Response of the Ocean: Correlation versus Deconvolution," *Inverse Problems in Underwater Acoustics*, Chapter 5, Springer, 2001.

[Miller 2009] P. Miller, "Paul's Softrock GNU Radio Experiments, 2009-02-15," Accessed online: http://voltar.org/gnuradio/, Feb. 2009.

[MIRACLE 2010] NS-MIRACLE: Multi-InteRfAce Cross-Layer Extension library for the Network Simulator, Accessed online: http://telecom.dei.unipd.it/pages/read/58, Mar. 2010.

[Munk 1974] W. H. Munk, "Sound channel in an exponentially stratified ocean with applications to SOFAR," Journal of the Acoustical Society of America 55, pp. 220–226, 1974.

[Nakagami 1960] M. Nakagami, "The *m*-Distribution – A General Formula of Intensity Distribution of Rapid Fading," in W. C. Hoffman (ed.): *Statistical Methods in Radio Wave Propagation*, Pergamon Press: New York, 1960, pp. 3–36.

[NATO 2008] NATO, "Reconnaissance, Surveillance & Undersea Networks (RSN)," Accessed online: http://www.nurc.nato.int/research/rsn.htm, Oct. 2008.

[NGDC 2010] National Geophysical Data Center: Seafloor Surficial Sediment Descriptions, Accessed online: http://www.ngdc.noaa.gov/mgg/geology/deck41.html, Mar. 2010.

[NI 2010a] National Instruments PCI-6123 DAQ Specifications, Accessed online: http://sine.ni.com/nips/cds/view/p/lang/en/nid/201938, Jan. 2010.

[NI 2010b] National Instruments USB-6221 DAQ Specifications, Accessed online: http://sine.ni.com/nips/cds/view/p/lang/en/nid/203093, Jan. 2010.

[NI 2010c] National Instruments, "Characteristics of Different Smoothing Windows," Accessed online: http://zone.ni.com/reference/en-XX/help/371361E-01/lvanlsconcepts/char_smoothing_windows, Feb. 2010.

[NPS 2008] Naval Postgraduate School, "Seaweb Port Surveillance," Accessed online: www.nps.edu/Research/mdsr/Docs/Seaweb2008trials.ppt, Oct. 2008.

[NS2 2010] The Network Simulator – ns-2, Accessed online: http://nsnam.isi.edu/nsnam/index.php/User_Information, Mar. 2010.

[NYHOPS 2009] Urban Ocean Observatory at the Center for Maritime Systems, "NY-HOPS Present Conditions Time Series and Downloads," Accessed online: http://hudson.dl.stevens-tech.edu/maritimeforecast/PRESENT/data.shtml, Aug. 2009.

[OCEANEARS 2010] OCEANEARS – *The World's Premier Online Source for high quality, hi-fidelity synchronised swimming sound systems!* Accessed online: http://www.oceanears.com, Jan. 2010.

[Olds 2008] J. Olds, "J-QAM: A QAM soundcard modem," Accessed online: http://homepages.paradise.net.nz/peterfr2/QAM.htm, Apr. 2010.

[OMNeT 2010] *OMNeT++ Community Site*, Accessed online: http://www.omnetpp.org, Mar. 2010.

[OPNET 2010] Application and Network Performance with OPNET, Accessed online: http://www.opnet.com/, Mar. 2010.

[Parrish 2007] N. Parrish, S. Roy, W. L. J. Fox, and P. Arabshahi, "Rate-Range for an FH-FSK Acoustic Modem," in Proc. WUWNet'07, Sept. 2007.

[Partan 2006] J. Partan, J. Kurose, and B. N. Levine, "A Survey of Practical Issues in Underwater Networks," in Proc. WUWNet'06, 2006.

[Pati 1993] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition," in Proc. 27th Asilomar Conference on Signals, Systems and Computers, A. Singh, ed., IEEE Comput. Soc. Press, Los Alamitos, CA, 1993.

[Pelekanakis 2003] C. Pelekanakis, M. Stojanovic, and L. Freitag, "High Rate Acoustic Link for Underwater Video Transmission," in Proc. OCEANS 2003, Vol. 2, pp. 1091–1097, Sept. 2003.

[Porter 1994] M. B. Porter and Y. C. Liu, "Finite-Element Ray Tracing," *Theorertical and Computational Acoustics – Volume 2*, D. Lee and M. H. Schultz (ed.), pp. 947–956, World Scientific, 1994.

[Preisig 2006] J. Preisig, "Acoustic Propagation Consideration for Underwater Acoustic Communications Network Development," in Proc. WUWNet'06, 2006.

[Preisig 2009] J. Preisig, personal communication at WUWNet'09, Nov. 3, 2009.

[Press 2007] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Third Edition, Cambridge University Press: Cambridge, 2007.

[Proakis 2007] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Application*, Fourth Edition, Pearson: Upper Saddle River, 2007.

[Proakis 2008] J. G. Proakis and M. Salehi, *Digital Communications*, Fifth Edition, McGraw-Hill: Boston, 2008.

[Rappaport 2002] T. S. Rappaport, *Wireless Communications: Principles and Practice*, Second Edition, Prentice Hall PTR: Upper Saddle River, 2002.

[Riad 1986] S. M. Riad, "The Deconvolution Problem: An Overview," in Proc. of the IEEE, Vol. 74, Issue 1, pp. 82–85, 1986.

[Rice 1944] S. O. Rice, "Mathematical analysis of random noise," Bell Systems Technical Journal, Vol. 23, pp. 282–332, 1944.

[Rice 1945] S. O. Rice, "Mathematical analysis of random noise-- conclusion," Bell Systems Technical Journal, Vol. 24, pp. 46–156, 1945.

[Rice 2001] J. A. Rice, R. K. Crebei, C. L. Fletcher, P. A. Baxley, D. Davison, and K. E. Rogers, "Seaweb Underwater Acoustic Nets," SSC San Diego Biennial Review 2001, SSC San Diego Technical Document TD 3117, pp. 234–250, Aug. 2001.

[Rice 2008] J. A. Rice, "Seaweb Network for FRONT Oceanographic Sensors," FY02 Annual Project Report – National Oceanographic Partnership Program, Accessed online: www.coreocean.org/nopp/project-reports/reports/02rice.pdf, Oct. 2008.

[Rodríguez 2008] O. C. Rodríguez, "General description of the BELLHOP ray tracing program, Version 1.0, Jun. 13, 2008," Accessed online: http://oalib.hlsresearch.com/Rays/GeneralDescription.pdf, Mar. 2010.

[Roh 2008] H.-S. Roh, A. Sutin, and B. Bunin, "Determination of acoustic attenuation in the Hudson River Estuary by means of ship noise observations," JASA Express Letters, May 2008.

[Sailer 2000] T. Sailer, "Soundmodem on Modern Operating Systems (2000)," Accessed online: http://www.baycom.org/~tom/ham/dcc2000/soundmodem.pdf, Feb. 2010.

[Schill 2004] F. Schill, U.R. Zimmer, and J. Trumpf, "Visible Optical Communication and Distance Sensing for Underwater Applications," in Proc. Australian Conference Robotics and Automaton Association, 2004.

[Schomer 1972] P. D. Schomer, "Measurement of Sound Transmission Loss by Combining Correlation and Fourier Techniques," Journal of the Acoustical Society of America, Volume 51, Number 4 (Part 1), 1972.

[Scussel 1997] K. F. Scussel, J. A. Rice, and S. Merriam, "A New MFSK Acoustic Modem for Operation in Adverse Underwater Channels," in Proc. IEEE Oceans '97, Halifax, Nova Scotia, Canada, Oct. 1997.

[Shaw 2006] A. Shaw, A.I. Al-Shamma'a, S.R. Wylie, and D. Toal, "Experimental Investigations of Electromagnetic Wave Propagation in Seawater," in Proc. 36th European Microwave Conference, Manchester UK, Sept. 2006.

[Shin 2008] S. Y. Shin and S. H. Park, "Omnet++ Based Simulation for Underwater Environment", in Proc. 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, pp. 689–694, 2008.

[Siegel 1973] M. Siegel and R. W. P. King, "Electromagnetic Propagation between Antennas Submerged in the Ocean," IEEE Trans. on Antennas and Propagation, Vol. 4, pp. 507–513, 1973.

[Simulink 2010] Simulink, "Simulation and Model-Based Design," Accessed online: http://www.mathworks.com/products/simulink/, Apr. 2010.

[Sklar 2001] B. Sklar, *Digital Communications: Fundamentals and Applications*, Second Edition, Prentice Hall PTR: Upper Saddle River, 2001.

[Smith 2003] S. Smith, *Digital Signal Processing, A Practical Guide for Engineers and Scientists*, Newnes: Amsterdam, 2003.

[Sozer 1999] E. M. Sozer, J. G. Proakis, M. Stojanovic, J. A. Rice, A. Benson, M. Hatch, "Direct Sequence Spread Spectrum Based Modem for Underwater Acoustic Communication and Channel Measurements," in Proc. of OCEANS'99, Nov. 1999.

[Sozer 2006] E. Sozer and M. Stojanovic, "Reconfigurable Acoustic Modem for Underwater Sensor Networks," in Proc. WUWNet'06, Sept. 2006.

[Stojanovic 1993] M. Stojanovic, J. Catipovic, and J. G. Proakis, "Adaptive multichannel combining and equalization for underwater acoustic communications," J. Acoust. Soc. Am., Vol. 94, No. 3, pp. 1621–1631, Sept. 1993.

[Stojanovic 1994] M. Stojanovic, J. A. Catipovic, and J. G. Proakis, "Phase-Coherent Digital Communications for Underwater Acoustic Channels," IEEE Journal of Oceanic Engineering, Vol. 19, No. 1, Jan. 1994.

[Stojanovic 2003] M. Stojanovic, "Acoustic (Underwater) Communications," entry in Encyclopedia of Telecommunications, John G. Proakis (ed.), John Wiley and Sons, 2003.

[Stojanovic 2006] M. Stojanovic and L. Freitag, "Multichannel Detection for Wideband Underwater Acoustic CDMA Communications," IEEE Journal of Oceanic Engineering, Volume 31, Issue 3, pp. 685–695, July 2006.

[Stojanovic 2006a] M. Stojanovic, "Low Complexity OFDM Detector for Underwater Channels," in Proc. OCEANS'06, Sept. 2006.

[Strutt 1880] J. W. Strutt (Lord Rayleigh), "On the resultant of a large number of vibrations of the same pitch and of arbitrary phase," Philos. Mag. 10, pp. 73–78, 1880.

[Torres 2009] D. Torres, J. Friedman, T. Schmid, and M. B. Srivastava, "Software-Defined Underwater Acoustic Networking Platform," in Proc. WUWNet'09, Nov. 2009.

[Tritech 2010] Tritech, "AM-300 Acoustic Modem," Accessed online: http://www.tritech.co.uk/products/datasheets/am-300-acoustic-modem.pdf, Apr. 2010.

[URI 2008] University of Rhode Island, *Discovery of Sound in the Sea*, "Cylindrical vs. Spherical Spreading." Accessed online: http://www.dosits.org/science/adv/cvss1.htm, Sept. 2008.

[Urick 1996] R. J. Urick, *Principles of Underwater Sound 3rd Edition*, Peninsula Publishing, 1996.

[Venezia 2003] W. Venezia, W. Baxley, P. Tatro, M. Dhanak, F.R. Driscoll, P. Beaujean, S. Shock, S. Glegg, E. An, M. Luther, B. Weisberg, H. DeFerrari, N. Williams, H. Nguyen, N. Shay, J. Van Leer, R. Dodge, D. Gilliam, A. Soloviev, S. Pomponi, M. Crane, and K. Carter, "SFOMC, A Successful Navy And Academic Partnership Providing Sustained Ocean Observation Capabilities in the Florida Straits," Marine Technology Society Journal, Vol. 37, pp. 81–91, 2003.

[WIS 2009] Wave Information Studies (WIS) Direction Convention, Accessed online: http://www.frf.usace.army.mil/wis/datadefs.html, Aug. 2009.

[WJC 1980] Watkins-Johnson Company, "FSK: Signals and Demodulation, technotes," Vol. 7, No. 5, Sept./Oct. 1980.

[WOD 2010] World Ocean Database and World Ocean Atlas Series, Accessed online: http://www.nodc.noaa.gov/OC5/indprod.html, Mar. 2010.

[Yacoub 2005] M. D. Yacoub, G. Fraidenraich, and J.C.S. Santos Filho, "Nakagami-m phase-envelope joint distribution," Electronics Letters, Vol. 41, Issue 5, pp. 259–261, Mar. 2005.

[Yan 2007] H. Yan, S. Zhou, Z. J. Shi, and B. Li, "A DSP Implementation of OFDM Acoustic Modem," Proc. WUWNet'07, Sept. 2007.

[Yip 2000] K.-W. Yip and T.-S. Ng, "A Simulation Model for Nakagami-m Fading Channels, m < 1," IEEE Trans. on Comm., Vol. 48, No. 2, Feb. 2000.

[Yang 2004] T. C. Yang, "Environmental effects on phase coherent underwater acoustic communications: A perspective from several experimental measurements," in Proc. High Frequency Ocean Acoustics, Vol. 728, pp. 90–97, La Jolla, 2004.

[Yang 2008] T. C. Yang and W.-B. Yang, "Performance analysis of direct-sequence spread-spectrum underwater acoustic communications with low signal-to-noise-ratio input signals," J. Acoust. Soc. Am., Vol. 123, No. 2, pp. 842–855, 2008.

[Zhou 2009] S. Zhou, Z. J. Shi, J.-H. Cui, H. Zhou, J. Liu, and P. Carroll, "AquafModem: A Stand-alone Underwater Acoustic Modem Based on OFDM Technology," in Proc. WUWNet'09, Nov. 2009.

VITA

Brian S. Borowski

Date of birth: February 26, 1979
Place of birth: Ridgewood, NJ USA

Education:

Stevens Institute of Technology, Hoboken, NJ
Doctor of Philosophy in Computer Science, June 2010
Graduate Certificate in Distributed Systems, January 2008
Graduate Certificate in Computer Systems, May 2007

Stevens Institute of Technology, Hoboken, NJ
Master of Science in Computer Science, May 2004 GPA 4.0/4.0
Graduate Certificate in Database Systems, May 2004

Seton Hall University, South Orange, NJ
Bachelor of Science in Computer Science, May 2001 GPA 4.0/4.0
Minor: Mathematics

Publications:

- Brian Borowski and Dan Duchamp, *Measurement-based Underwater Acoustic Physical Layer Simulation*, in Proceedings of MTS/IEEE OCEANS 2010, September 2010, Seattle, Washington (to appear).
- Brian Borowski and Dan Duchamp, Short Paper: *The Softwater Modem - A Software Modem for Underwater Acoustic Communication*, in Proceedings of the ACM International Workshop on Underwater Networks (WUWNet'09), November 2009, Berkeley, California.
- Brian Borowski, *Characterization of a Very Shallow Water Acoustic Communication Channel*, in Proceedings of MTS/IEEE OCEANS 2009, October 2009, Biloxi, Mississippi.
- Brian Borowski, Alexander Sutin, Heui-Seol Roh, and Barry Bunin, *Passive Acoustic Threat Detection in Estuarine Environments*, in Proceedings of SPIE Vol. 6945, March 2008, Orlando, Florida.
- Brian Borowski, Heui-Seol Roh, Barry Bunin, and Alexander Sutin, *Estimation of Passive Acoustic Threat Detection Distances in Estuarine Environments*, in Proceedings of the 153rd Meeting of the Acoustical Society of America, June 2007, Salt Lake City, Utah.
  (Placed second in the Best Student Paper competition of the Engineering Acoustics section)

Presentations:

- *The Softwater Modem - A Software Modem for Underwater Acoustic Communication*, ACM International Workshop on Underwater Networks (WUWNet'09), November 3, 2009, Berkeley, California.

- *Characterization of a Very Shallow Water Acoustic Communication Channel*, MTS/IEEE OCEANS 2009, October 29, 2009, Biloxi, Mississippi.
- *Characterization of a Very Shallow Water Acoustic Communication Channel*, Maritime Security Laboratory at Stevens Institute of Technology, October 5, 2009, Hoboken, NJ. (End-of-year review presentation given to ONR sponsor)
- *Elements of Channel Characterization*, Maritime Security Laboratory at Stevens Institute of Technology, January 6, 2009, Hoboken, NJ.
- *A Software-Based Approach to Communication in Underwater Acoustic Sensor Networks*, Stevens Institute of Technology, November 24, 2008, Hoboken, NJ. (Presentation used at thesis proposal defense)
- *Passive Acoustic Threat Detection in Estuarine Environments*, Stevens Institute of Technology, March 28, 2008, Hoboken, NJ. (Presentation used at oral qualifying examination)
- *Estimation of Passive Acoustic Threat Detection Distances in Estuarine Environments*, 153rd Meeting of the Acoustical Society of America, June 5, 2007, Salt Lake City, Utah.

Honors:

- Stanley Fellowship, September 2009 - May 2010 (tuition, fees, and stipend)
  Competitively awarded across all fields at Stevens Institute of Technology; total of 8 awards granted
- Stanley Fellowship, September 2008 - May 2009 (tuition, fees, and stipend)
  Competitively awarded across all fields at Stevens Institute of Technology; total of 10 awards granted
- Upsilon Pi Epsilon - the Honor Society in Computing and Information Disciplines, December 2006
- Technogenesis Fellowship, September 2005
- Outstanding Computer Science Teaching Assistant, May 2004
- First in Class, Summa Cum Laude, and Computer Science Departmental Honors Citation, May 2001
- Pi Mu Epsilon - the Honorary National Mathematics Society, May 2000
- Seton Hall Provost Scholarship (4-year, full tuition), September 1997

Certification: Sun Certified Programmer for the Java 2 Platform, May 2002

Skills:

Programming Languages: Java, C++, C, Visual Basic, MATLAB, PHP, Perl, Scheme, Bash, and SQL
Operating Systems: Microsoft Windows 3.1 - Vista, and Ubuntu Linux
Databases: MySQL, SQL Server, and Access
Web Technologies: HTML/XHTML, CSS, JavaScript, JSP, ASP, XML, XSLT, and XSL-FO
Design Technologies: UML, Rational Rose, and ER diagrams
Version Control: Rational ClearCase/ClearQuest and MKS Source Integrity

Software: IIS, Microsoft Office, Dreamweaver, Corel Paint Shop Pro, WinRunner, Apache Tomcat, and Apache FOP
Hardware: Proficient at building, upgrading, and troubleshooting Intel and AMD-based PCs

Experience:
Stevens Institute of Technology, Hoboken, NJ
Research Assistant 08/05 - Present

- Extend OMNeT++ with an underwater channel model implemented in MATLAB and exported as a shared library
- Design and implement a configurable acoustic software modem in Java/C that integrates with the sockets interface for easy deployment of network applications
- Characterize the Hudson River estuary as a communications channel by generating the scattering function and all derived views
- Build PC104-based computers for use in an underwater sensor network
- Research diver detection using passive sonar

Teaching Assistant 01/03 - 05/04; 08/07 - 12/07

- Created syllabus, chose required textbook and supplementary materials, and devised and graded assignments for a new course in concurrent programming in conjunction with my adviser (Fall 2007)
- Taught object-oriented software design and programming techniques (Spring 2003), introduction to computer science (Fall 2003), and data structures and algorithms (Spring 2004) under the guidance of the professor
- Led recitation sessions, held office hours, and devised and graded assignments

Cargo Manager Systems, Union, NJ
Web Developer Consultant 10/07 - 10/08

- Enhanced n-tier web applications (JSP/XHTML - Java beans - JDBC) that manage imports, exports, transportation, and warehousing for the supply chain industry
- Modified functionality of a web application that performs government filing of import shipments

Syncsort Incorporated, Woodcliff Lake, NJ
Associate Software Engineer 06/04 - 08/05

- Enhanced DMExpress, an application for sorting, aggregating, copying, joining, and merging extremely large quantities of data
- Utilized MFC to add new front-end features
- Developed back-end infrastructure in standard C++ to run DMExpress tasks in parallel
- Wrote and executed WinRunner scripts to ensure program stability
- Created Perl and Bash scripts to facilitate source code management procedures

KPMG LLP, Montvale, NJ
Programmer Analyst 10/02 - 12/02

- Debugged, maintained, and enhanced KPMG/Link Enterprise, an application that manages expatriate employees and related tax issues
- Performed and tested software builds

Prudential Financial, Iselin, NJ
Web/Application Developer 07/01 - 07/02

- Worked in a team to develop an award-winning application for content management and desktop publishing using ASP 3.0
- Developed a Visual Basic tool that tests the business logic of asset allocation software written in XML
- Designed and coded an ASP 3.0 user interface for Asset Allocation Online, a web application that enables a client to perform his or her own asset allocation by answering questions over the Internet
- Redesigned a series of web pages that contains the monthly performance review and daily unit values of variable life insurance products

ADP, Roseland, NJ
Web Developer Co-op 06/00 - 08/00

- Created an intranet site for the PCPI - Internet Payroll for the PC - department to keep all team members aware of their project's status
- Devised a JSP application that dynamically creates links to files within specific directories so that information can easily be added to the site without maintenance

Seton Hall University, South Orange, NJ
Software Developer 05/00 - 07/00; 05/99 - 07/99

- Provided new software for professors seeking teaching tools
- Proposed, designed, and developed a sorting algorithms demo in Java for use in the CDI - Curriculum Development Initiative – project
- Designed and developed a truth table constructor in Java for use in the CDI project

Other Experience:
Saint Bonaventure Church, Paterson, NJ
Organist/Pianist/Cantor/Director of Music 11/91 - Present

Seton Hall University, South Orange, NJ
Assistant Organist/Cantor 09/97 - 05/01

Memberships: ACM, IEEE, ASA