Adaptive Protocols to Improve TCP/IP Performance in an LMDS Network using a Broadband Channel Sounder

Todd Jacob Eshler

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Dr. Scott F. Midkiff, Chair
Dr. Charles W. Bostian
Dr. Luiz A. DaSilva

April 22, 2002
Blacksburg, Virginia

Keywords: TCP/IP, LMDS, Broadband Sounder, Forward Error Correction, Bit Error Rate, Automatic Retransmission Request Scheme

Adaptive Protocols to Improve TCP/IP Performance in an LMDS Network using a Broadband Channel Sounder

Todd Jacob Eshler

Dr. Scott F. Midkiff, Chair
Computer Engineering

(ABSTRACT)

Virginia Tech researchers have developed a broadband channel sounder that can measure channel quality while a wireless network is in operation. Channel measurements from the broadband sounder hold the promise of improving TCP/IP performance by trigging configuration changes in an adaptive data link layer protocol. We present an adaptive data link layer protocol that can use different levels of forward error correction (FEC) codes and link layer automatic retransmission request (ARQ) to improve network and transport layer performance.

Using a simulation model developed in OPNET, we determine the effects of different data link layer protocol configurations on TCP/IP throughput and end-to-end delay using a Rayleigh fading channel model. Switching to higher levels of FEC encoding improves TCP/IP throughput for high bit error rates, but increases end-to-end delay of TCP/IP segments. Overall TCP/IP connections with link layer ARQ showed approximately 150 Kbps greater throughput than without ARQ, but lead to the highest end-to-end delay for high bit error rate channels.

Based on the simulation results, we propose algorithms to maximize TCP/IP throughput and minimize end-to-end delay using the current bit error rate of the channel. We propose a metric, carrier-to-interference ratio (CIR) that is calculated from data retrieved from the broadband channel sounder. We propose algorithms using the carrier-to-interference ratio to control TCP/IP throughput and end-to-end delay.

The thesis also describes a monitor program to use in the broadband wireless system. The monitor program displays data collected from the broadband sounder and controls the settings for the data link layer protocol and broadband sounder while the network is in operation.

Acknowledgements

I would like to thank Dr. Midkiff for being my advisor. His advice and guidance have proved to be invaluable. His suggestions have helped me develop this thesis and added to my experience as a graduate student. Also, I would like to thank the other members of my advisory committee, Dr. Bostian and Dr. DaSilva.

I would like to thank Tim Gallagher for guidance in developing forward error correction model and sounder data analysis used in this thesis. I would like to thank Christian Rieser for the use of the broadband sounder.

I would like to thank the National Science Foundation for their support. This work was supported by National Science Foundation's Digital Government program (grant 9983463).

I would thank the faculty, staff, and students of the CWT for their assistance during my graduate career at Virginia Tech.

Finally, I would like to thank my family and friends who have provided support during my graduate and undergraduate career at Virginia Tech.

Table Of Contents

Table Of Figures

Table Of Tables

Chapter 1. Introduction

1.1. Background

The research described in this thesis evolved from the project "Testbed for High-Speed 'End-to-End' Communications in Support of Comprehensive Emergency Management" sponsored by the National Science Foundation's Digital Government program. The goal of this project is to develop a rapidly-deployable, reliable broadband wireless backbone network for emergency and disaster response that supports the Internet Protocol (IP). The wireless network is to be used in areas where the wired communication infrastructure has been damaged, for example, by a hurricane, a tornado, or an earthquake, or in areas without an effective communications system. The network is needed to support common Internet applications such as web browsing and electronic mail. Also, the network is needed to support more advanced applications such as video conferencing and other collaboration tools.

Virginia Tech researchers are investigating, designing, and building a prototype wireless network that will operate in the Local Multipoint Distribution Service (LMDS) frequency range. The bandwidth available with LMDS will support high data rates of 100 megabits per second (Mbps) or more. However, high error rates could reduce network performance significantly, especially if the network must be deployed in a sub-optimal configuration. This is especially true if the Transmission Control Protocol (TCP) is used because of how TCP handles packet loss due to congestion and errors.

The focus of the research described in this thesis is the problem of achieving high levels of network performance for TCP/IP applications over wireless links with potentially varying performance. We investigate existing schemes for adaptive protocols, create a new scheme for changing link-level protocol operation that relies on channel information obtained from a built-in broadband channel sounder, analyze the performance of this scheme using simulation, and develop a design for a prototype implementation. In addition, to support the overall project, we have also developed interfaces between the control program implementing adaptive decision-making and the modem controller and the broadband channel sounder. We have also implemented a prototype user interface to monitor and control overall system operation.

1.2. TCP/IP Performance in Wireless Networks

Over the years, TCP/IP has become the predominate protocol suite for networked applications. Many different physical media have had data link protocols developed that allow the medium to carry TCP/IP traffic. Most of these physical media use some form of electrical wiring or optical cabling to carry the network traffic. Because wired physical media tend to have low bit error rates and loss due to congestion is of greater concern, TCP/IP treats loss due to bit error and loss due to congestion in the same manner.

When a packet is lost on a wired network, TCP/IP assumes that the packet was lost because of network congestion. TCP/IP reacts by retransmitting the lost packet and shrinking the sending window to help relieve the assumed congestion in the network. This reaction tends to improve performance for a wired network because the loss is typically due to buffer overflow resulting from congestion, not a bit error. Since packet loss due to bit errors occur regularly on a wireless network, TCP/IP will continue to shrink the sending window with every packet loss. As TCP/IP shrinks the sending window, it uses the available bandwidth on the network less and less efficiently [3, 19].

Changing the TCP/IP suite itself is typically not feasible since the wireless network needs to be interoperable with the larger Internet. So, to prevent this performance reduction, the physical layer, data link layer or network layer beneath TCP/IP must be changed to reduce or hide bit errors. Reducing or hiding the bit error can be achieved through several methods, which include adaptive forward error correction (FEC) and automatic repeat request (ARQ). The method and layer where the method or methods are employed must have a minimum impact on applications that need to run on the network.

Many solutions have been proposed to improve TCP/IP performance on wireless networks. All of the solutions use adaptive protocols that change their behavior as the bit error rate increases or decreases. The solutions can be divided into three groups: (1) protocols that perform adaptation at the physical layer, (2) protocols that perform adaptation at the data link layer, and (3) protocols that perform adaptation at the network layer. These schemes are reviewed in Chapter 2.

1.3. Rapidly-Deployable Broadband Wireless Network

Figure 1.1 shows a high-level view of the network being developed by Virginia Tech researchers. The network consists of a hub and one or more remote units connected via a rapidly deployed Local Multipoint Distribution Service (LMDS) broadband wireless backbone. The test bed to be deployed in this project will consist of a hub and eight remote units. The hub and the remote units contain routing and other network service functions, as well as a radio and modem, built-in channel sounder, and Geographic Positioning System (GPS) unit. The remote units can provide connections to hosts on a remote local area network (LAN), for example using 10/100 BaseT Ethernet and/or IEEE 802.11 wireless LAN. The hub provides a connection to a wide area network, for example Net.Work.Virginia, via a high data rate connection. Thus, end hosts can gain access to remote Geographic Information System (GIS) and other networked servers using the rapidly-deployed LMDS backbone and existing wide area networks.

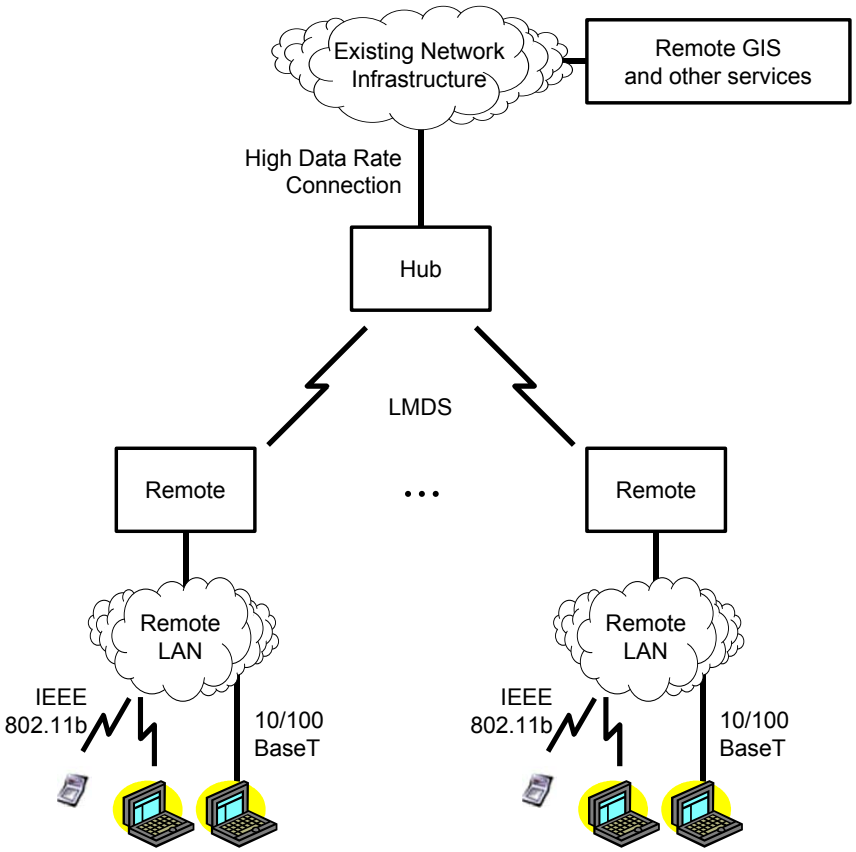

Figure 1.1. High-level view of the system under investigation.

The network supports bi-directional traffic between the hosts and the remotes. The network will functionally serve as an Ethernet bridge. To ensure that the network is usable for standard Internet applications, the following restrictions must be adhered to:

- A network host must be able to use an "out of the box" TCP/IP protocol stack that does not need to be recompiled to run on the network.
- Network applications should not need to be modified or recompiled to run on the network.

Since the network will be used as a communication backbone during disaster situations and maintain communications with possibly varying channel conditions, the network should use an adaptive scheme to improve TCP/IP performance. The adaptive scheme must adhere to the two goals listed above. An adaptive link-layer scheme will be developed to use data from the built-in channel sounder to help improve the performance of TCP/IP.

1.4. Research Goals

In this thesis, we determine if a broadband channel sounder can be used to develop an adaptive link layer scheme to improve TCP/IP performance for the network. The adaptive link layer scheme uses forward error correction with varying coding rates and an automatic repeat request scheme that can be disabled or enabled. Through simulation, TCP/IP performance is measured using different coding rates and with the ARQ scheme turned on and off. An algorithm for controlling the adaptive link layer is proposed that uses the channel data provided by the broad band sounder. The algorithm varies the coding rate and controls the ARQ scheme to improve TCP/IP performance for the network.

1.5. Document Overview

Chapter 2 provides an overview of different approaches to improve TCP/IP performance in broadband wireless networks.

Chapter 3 presents how the adaptive link layer protocols designed for the rapidly deployable LMDS network to improve TCP/IP performance.

Chapter 4 presents the proposed multiple access scheme and data link layer for the LMDS broadband wireless network.

Chapter 5 presents the design of the simulation model and discusses simulation validation and verification results.

Chapter 6 presents the results collected from all the simulation runs and discusses the performance differences of the different FEC coding levels and link layer ARQ scheme.

Chapter 7 summarizes and discusses future research that could stem of this thesis.

Chapter 2. Adaptive Methods to Improve TCP/IP Performance

The proposed system offers high data rates. However, high error rates can substantially reduce system performance. TCP does not respond well to high error rates since it responds to packet loss as a result of error or congestion in the same manner. As the error rate increases, TCP shrinks its sending window size and lowers the overall throughput [3, 4, 14, 19]. There are two basic approaches to address this problem: (i) reduce the error rate at the link or physical level, or (ii) reduce TCP's performance penalty for lost packets. Each of these approaches is discussed below.

2.1. Physical Layer Adaptation

Wireless networks are prone to higher error rates than wired networks. To reduce the effective error rate seen by TCP/IP for the rapidly deployable broadband network, we investigated ways to adapt the wireless transmissions to reduce the error rate or to become more tolerant of the higher bit error rate, but to not sacrifice higher data rates when they can be achieved. The physical layer has several parameters that can be adjusted to improve performance of protocols at higher layers. This section describes adaptation schemes that can be used at the physical layer. These schemes depend on the properties of the radio used for the wireless network. Therefore, some of these schemes may not be able to be implemented because of the radio chosen.

2.1.1. Adaptive Modulation

Zukerman and Gitlis [6] define the term "modulation gain" as the increase in data rate (measured in symbols per second) for a given modulation constellation compared to the data rate for binary modulation. Adaptive modulation adjusts the modulation gain, hence changing the size of the modulation constellation, to get optimal performance for a particular channel quality. Changing the modulation constellation as the channel quality varies can improve performance of the wireless network [6, 8]. Using adaptive modulation together with automatic repeat request (ARQ) has also been shown to improve the performance of a wireless network [6].

2.1.2. Pseudo-Noise Code

A pseudo-noise (PN) code is used in direct-sequence spread-spectrum networks such as IEEE 802.11 [13, 18]. A longer PN code requires more bandwidth than a shorter PN Code [18], but a longer PN code improves the error rate of the wireless connection [13]. The CATER (Code Adepts To Enhance Reliability) [9] MAC protocol increases the length of the PN code after retransmissions fail. The CATER protocol improves performance when the wireless channel degrades, but it does introduce a little overhead even when the channel quality is good [13]. The ADIM-NB/PG protocol (an adaptive DSSS MAC protocol for a non-broadcast multiple access medium with processing gain control) extends the idea of changing the PN-code based on channel quality by changing the PN-code on a per packet basis [19]. ADIM-NB/PG does improve network throughput and delay, but requires a radio with a great deal of functionality [19].

2.1.3. Forward Error Correction

Forward error correction can be used to recover from errors due to noise in a wireless link [7, 1]. Unnecessary overhead can be introduced by FEC in an environment with a low error rate [7]. Also, errors can be masked by FEC in an environment with a high error rate [7]. Eckhardt and Steenkiste suggest adaptive FEC [7]. The amount of error correction is increased as the wireless link degrades; as the link improves, error correction is decreased to improve utilization of bandwidth. Eckhardt and Steenkiste show that FEC improves performance in a WaveLAN wireless LAN [7]. Two disadvantages of adaptive FEC are that the effective bandwidth is reduced as FEC is increased and processing overhead at each wireless node is increased [7].

2.2. Data Link Protocol Adaptation

This section describes protocols that adapt to the network conditions at the data link layer. Some of the protocols discussed are aware of the network layer, meaning that the network layer protocol used must be modified for the data link protocol to be effective. Other data link protocols presented are not aware of the network layer, meaning that the network protocol does not need to be modified for the data link protocol to be effective.

2.2.1. Network-Aware Data Link Protocols

A data-link protocol can improve the wireless connection by caching packets on each side of the wireless link. The SNOOP protocol proposed by Balakrishnan, et al. [3] improves the link in this manner. Packets are cached and retransmitted to recover from losses. In a network with a fixed host, base host, and remote host, SNOOP functions as shown in Figure 2.1.

Figure 2.1. Diagram of a network using SNOOP.

The fixed host represents any node on a wired network wanting to make a connection with the remote host. The network layer at the fixed host does not need to be modified [3]. The base host is the host that connects the mobile host to a wired network. Instead of running transport layer code, the base host runs a SNOOP module that allows the base host to keep track of TCP sequence numbers and cache packets sent to the mobile host. The cached packets are retransmitted to the mobile host if it appears that they have been lost over the wireless link. The mobile host uses TCP with selective acknowledgements (SACK) described in RFC 1323 [10]. The SNOOP module at the base host sends SACKs to the mobile host to force the mobile host to retransmit packets that the fixed host has not received from the mobile host. The SNOOP protocol has been shown to improve throughput up to 20 times over regular TCP [3]. However, it does require that the mobile host use a special form of TCP that has not been commonly implemented [3]. It is also not able to improve performance of other network protocols that might run over the wireless link [3].

2.2.2. Network-Unaware Data Link Protocols

A network-unaware data link protocol has an obvious advantage over network-aware data link layer protocols because the network layer protocol does not need to be modified. If the network layer is modified, then the wireless node will only be able to make connections with nodes that have a modified network layer.

Asymmetric Reliable Mobile Access in Link Layer (AIRMAIL) uses adaptive FEC and ARQ [1]. For AIRMAIL to work, the network must have a base station, which decides the amount of FEC to be used. AIRMAIL changes the amount of FEC used based on the quality of the current channel. AIRMAIL uses a window-based ARQ scheme, meaning that status messages are sent from the mobile units to the base station only after the mobile station has received a set number of packets from the base station [1]. However, the base station sends out status messages to the mobile units periodically [1]. For the system of interest in this research, AIRMAIL's requirement for a base station is not a problem because our design dictates the use of a base station. The advantages of AIRMAIL are that the protocol reduces the number of redundant transmissions and helps conserve power at the mobile stations using comprehensive status messages for groups of packets [1].

Transport Unaware Link Improvement Protocol (TULIP) is a data-link protocol that does not need a base station and does not keep any TCP state information [14]. TULIP can support unidirectional and bi-directional traffic over a half-duplex link [14]. Even though TULIP is network unaware, it is service aware [14]. This means that it provides two different types of data-link packets. The first type is a reliable link packet (RLP) that should be used with TCP/IP or any other protocol that requires reliable service. The second type is an unreliable link packet (ULP) that can be used with protocols such as UDP/IP [14]. Network layer protocols, like TCP/IP, tend to have generous timeouts [14]. TULIP takes advantage of these timeouts to try to recover from packets that are loss to due to error [14]. Unlike AIRMAIL or traditional stop-and-wait ARQ, TULIP only allows for enough time between the transmissions of packets for the receiver to send a link acknowledgement back [14]. Through simulation, Parsa and Garcia-Luna-Aceves have shown that TULIP has better throughput and lower packet delay than SNOOP.

2.3. Network Layer Techniques

Another method to improve TCP/IP performance over wireless media is to use a split-connection protocol, as shown in Figure 2.2. In split-connection protocols, the connection between two hosts is divided into two connections and a special protocol is used across the wireless link.

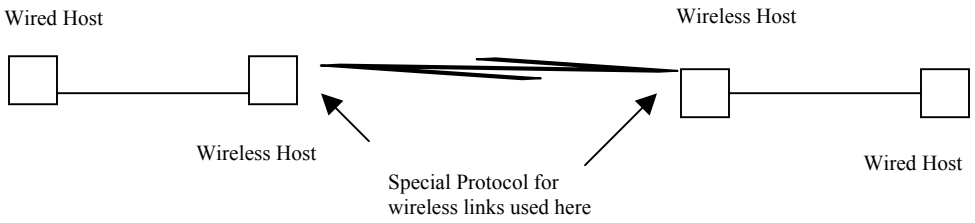

Figure 2.2. Diagram of a network using a split-connection protocol.

Indirect-TCP uses the split-connection technique [2]. Even though Indirect-TCP improves the throughput over wireless links [2, 4], the semantics of end-to-end TCP acknowledgements are broken [4]. Acknowledgements can reach the source host before they reach the destination host on the other side of the wireless link [4]. Another disadvantage of Indirect-TCP is that applications that run at the wireless host must be rewritten and recompiled to use Indirect-TCP socket calls [2, 4]. Balakrishnan, et al. showed that it is not necessary to split the TCP connection to improve performance across a wireless link [4].

2.4. Summary

Many forms of adaptation have been developed to improve TCP/IP performance. All of the adaptation methods perform adaptation at the network layer, data link layer, or physical layer. Network layer adaptation provides performance improvement with very few changes to network hardware. However, modified versions of TCP/IP tend to run on networks using network layer adaptation. Network aware data link layer protocols require fewer modifications of network layer and transport layer protocols. Unmodified versions of TCP/IP can run with networks using network unaware data link layer adaptation with increased modem controller hardware complexity. Using adaptation at the physical layer requires highest complexity in network radios and modems. However, physical layer adaptation requires no changes to TCP/IP.

Chapter 3. Problem Statement

3.1. Introduction

As discussed in Chapter 1, Virginia Tech researchers are developing a rapidly-deployable, reliable broadband wireless backbone to be used for emergency situations. The network needs to be easily setup by emergency crews. Computer systems that connect to the network should be able to use unmodified network applications and an unmodified TCP/IP protocol suite.

The network will be used in to respond to natural disasters such as hurricanes, tornadoes, and earthquakes. Unknown environments and varying weather conditions can lead to high bit error rates. These high error rates can hurt performance, especially for TCP because it responds to loss due to congestion in the same manner it responds to loss to due to error. As presented in Chapter 2, adaptive protocols can improve TCP/IP performance in such situations.

3.2. TCP/IP Behavior on High BER Links

When TCP/IP recognizes that a packet was lost because it did receive an acknowledgement from the receiver in time, TCP/IP assumes the loss is due to network congestion. Then TCP/IP reduces its congestion window size by half and begins to retransmit the lost packets on the link [15]. The assumption of loss due to network congestion is appropriate for wired network media.

On links with high BER values, the assumption that the loss is due to network congestion no longer holds. It is often as or more likely that the packet loss is due to an error in the packet. When the congestion window size is reduced, throughput is reduced unnecessarily as a result. On links with high BER values, TCP/IP will continue to reduce its throughput, as more packets are lost. TCP/IP will never fully utilize the bandwidth available on a broadband wireless link [3][19]. To prevent changing TCP/IP protocol stacks, adaptive approaches can be implemented to “hide” the losses due to bit errors on the channel.

3.3. Choosing an Adaptive Protocol Scheme

For the adaptive scheme for the Virginia Tech system, we chose to develop a network-unaware data link protocol that uses changing coding rates and an automatic repeat request scheme that can be turned on and off. The data link protocol does not require changes to the TCP/IP protocol stack. Network applications also do not need to be modified or recompiled to function. No physical layer adaptation was employed because Virginia Tech researchers had already obtained LMDS modems and radios that are not capable of physical layer adaptation and had a fixed modulation constellation. Researchers wanted to continue using these modems to reduce cost of the system. However, modem controllers still needed to be built for the LMDS modems that Virginia Tech acquired. Since the modem controller was not already built, an adaptive data link layer protocol can be implemented in the controller design that is implemented.

3.4. Using the Broadband Sounder with the Adaptive Scheme

Current adaptive methods and protocols calculate the current bit error rate for packets that are being received. The adaptive scheme uses the current bit error rate to change protocol and channel properties for transmitting packets. These schemes take a reactive approach by measuring the quality of the channel after transmission.

We wanted to investigate whether the broadband channel sounder [17] that is being developed for Virginia Tech's rapidly deployable network can be used to proactively measure the channel quality before transmission of packets. The broadband channel sounder's primary use is to set up remote sites. The sounder measures channel metrics such as coherence bandwidth, mean excess delay, root mean square (RMS) delay spread. Based on channel measurements, the location of the remote units and the aiming of their antennas can be modified to provide the best performance.

Since the broadband channel sounder is used to locate the remote units and aim their antennas for best performance, subsequent measurements by the broadband channel sounder can be used by the adaptive link protocols to improve TCP/IP performance in the system.

3.4.1. Modem Controller and Broadband Channel Sounder Interface

For the adaptive data link layer protocol to use the data provided by the sounder, an interface between the modem controller and broadband sounder needed to be developed. A host computer at the remotes sites will be connected to the modem controller via a serial port and to the broadband sounder via a parallel port.

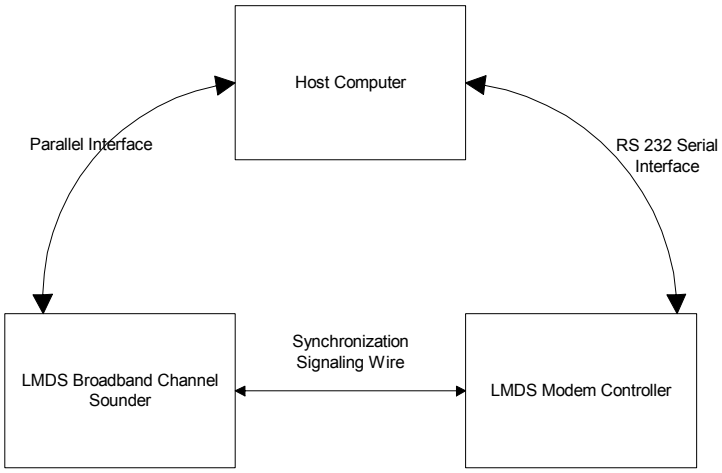

Figure 3.1 Interconnections between host computer, broadband sounder, and modem controller

A program was developed to run on the host computer (see Appendix A,B, and C). The program analyzes the data provided by the sounder to calculate the coherence bandwidth, mean excess delay, and RMS delay spread of the channel. Also, the program can issue modem commands over the serial port. These commands inform the modem controller to switch coding level, turn on/off the ARQ scheme, and change other configuration settings.

3.4.2. Synchronizing the Sounder and Modem Controller

While the broadband sounder sounds the channel, pulses emitted by the broadband sounder interfere with any data transmitted on the channel. To prevent the modem and the broadband sounder from operating at the same time, the modem controller and broadband sounder needed to be synchronized. For this purpose, a signal from the modem controller is sent to the sounder. The modem controller uses the signal to instruct the broadband channel sounder to sound the channel. After the sounder finishes, it uploads data collected to the program running on the host computer. The program analyzes the data. Then it initializes the broadband sounder for the next

sounding interval. The broadband sounder waits until the modem controller signals it to sound the channel.

3.5. Choosing a Multiple Access Scheme

A multiple access scheme is required to allow multiple remote units to access the wireless backbone network. Frequency Division Duplexing (FDD) was chosen to give the nodes on the network the capability for full duplex transmission. A time division multiple access (TDMA) scheme is proposed for the uplink stream so that it can be shared by multiple remote units. To maximize commonality in hardware and software between remote units and the hub, the downlink stream also uses a TDMA scheme, although only one station, the hub, utilizes all time slots. The TDMA scheme is described in detail in Chapter 4.

3.6. Developing a Data Link Layer Protocol

Since the modem controller connects directly to 10/100Mbps Ethernet, the data link layer was designed to encapsulate Ethernet packets. Encapsulating Ethernet packets directly instead of encapsulating network layer packets allows the modem controller to connect to a router or a host computer. A host computer can use an Ethernet card to connect to the modem controller instead of custom equipment. Encapsulating Ethernet packets directly allows the network to serve as a transport bridge to the network layer, allows other network layer protocols to be used instead of IP, leaves routing functions to other devises that can be easily purchased.

For Ethernet packets that are too large, packets are segmented before being transmitted and reassembled at the receiver before being sent to the rest of the network. The data link layer provides multiple coding rates to improve the BER as channel conditions worsen. An ARQ scheme provides reliable transport for packets that cannot be recovered using FEC. The ARQ scheme can be turned on and off when needed. A detail description of the data link layer protocol is provided in Chapter 4.

3.7 Building a Simulation Model

A simulation model of the LMDS backbone network was built using OPNET. A network with one hub and eight remotes was simulated. A simulation was run for each coding rate with the ARQ scheme on and off at various BER values. The average TCP/IP

throughput, congestion window size, receiving window size, end-to-end delay and jitter in end-to-end delay were compared to determine which coding rate should be used to provide the highest throughput for TCP/IP applications. An algorithm is also proposed for the adaptive protocol scheme using the broadband sounder. Details of the simulation model and results are discussed in Chapters 5 and 6, respectively.

3.8. Summary

TCP/IP performance can be hurt by high bit error rates that LMDS network may experience. A network unaware data link layer protocol that uses ARQ and adaptive FEC was chosen to improve TCP/IP performance on the network. This chapter explained how the protocol was integrated into the network to use the broadband sounder.

Chapter 4. Proposed Multiple Access Scheme and Data Link Protocol

This section presents the specification of the multiple access scheme and data link protocol that were developed in this research.

4.1. Multiple Access Scheme

A multiple access scheme is required to allow multiple remote units to access the wireless backbone network. This section describes our proposed multiple access scheme for the baseline network.

Frequency division multiplexing (FDM) is used to separate the downlink (hub-to-remote) and uplink (remote-to-hub) transmissions or streams. The central hub is the only device transmitting on the downlink frequency. All remotes transmit on the same uplink frequency, so a second multiple access scheme is required beyond the FDM scheme used for separating uplink and downlink traffic. The FDM scheme is shown in Figure 4.1, where f_u designates the uplink frequency and f_d designates the downlink frequency.

A time division multiple access (TDMA) scheme is proposed for use in the uplink stream. To maximize commonality in hardware and software between remote units and the hub, the downlink stream also uses a TDMA scheme, although only one station, the hub, will utilize all time slots.

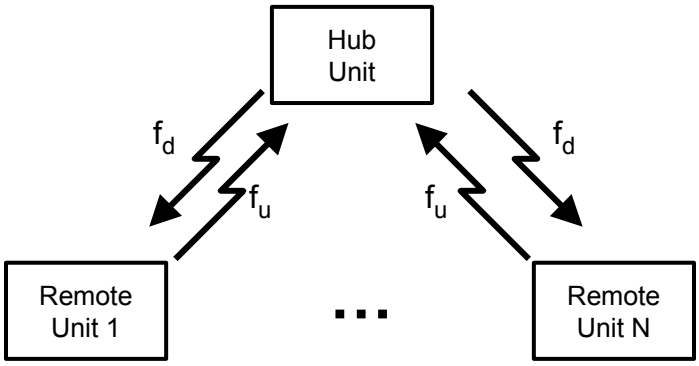

Figure 4.1. Frequency Division for uplink and downlink streams.

4.2. TDMA for the Uplink and Downlink Stream

The TDMA scheme for the uplink and downlink streams employs a "super frame" structure to allow the sounder to operate while the network is running.

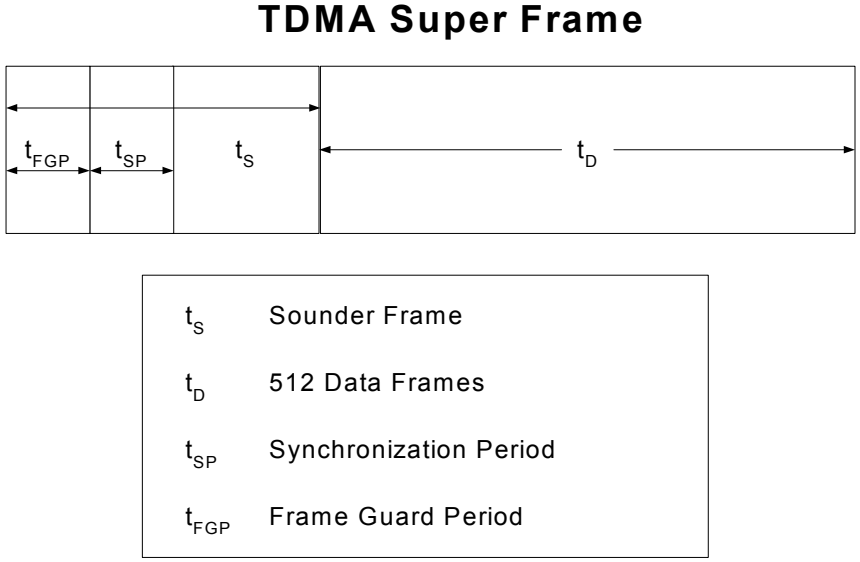

Figure 4.2. TDMA super frame format.

Figure 4.2 shows the format of the proposed TDMA super frame. The following definitions apply.

- Sounder Frame: Time slot in each frame to allow the broadband sounder to sound the channel. The sounder may not utilize every slot, but the slot appears in every super frame to allow flexibility in when sounding is done and to decouple the sounder from the transmit controller.
- Synchronization Period: This field contains synchronization characters used by all remotes and the hub to synchronize to each other. The central hub transmits this signal in downlink frames. The remotes use the position of this field to determine where their time slot lies in uplink frames.
- Frame Guard Period: Time between each super frame to prevent overlapping super frame slots due to clock skew and slow transmitter turn-off.
- Data Frames: A smaller frame containing a frame control header and eight data slots to allow up to eight different remotes transmit data. Data frames are repeated 512 times per super frame.

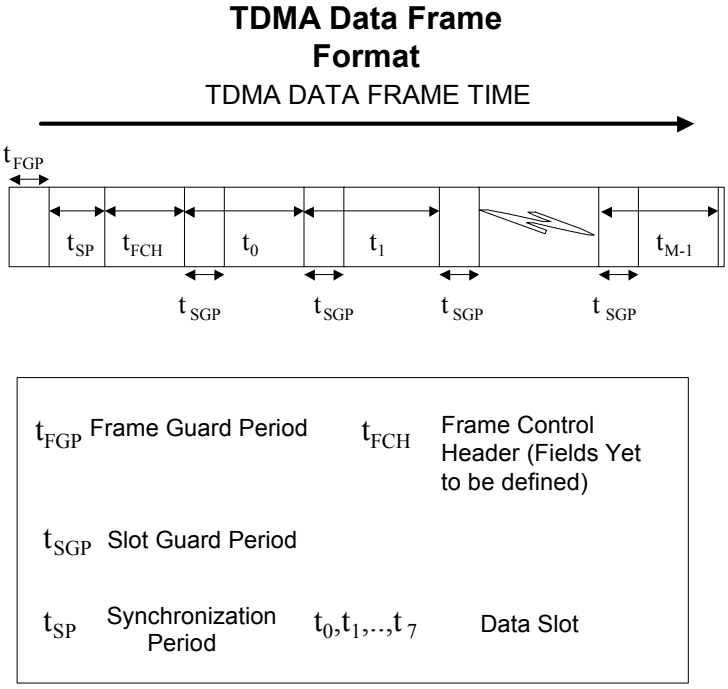

Figure 4.3 TDMA data frame format.

Figure 4.3 shows the data frame format. The components are defined as follows.

- Frame Guard Period: Time between each TDMA data frame. A frame guard period is placed between each frame to prevent frames from overlapping because of clock skew, slow transmit turn-off, or other factors.
- Synchronization Period: This field contains synchronization characters used by all remotes and the hub to synchronize to each other. The central hub transmits this signal in downlink frames. The remotes use the position of this field to determine where their time slot lies in uplink frames. The field is also placed before each data slot to ensure data slots are received correctly.
- Frame Control Header: This header provides fields that are used for control information, e.g., for link-level automatic retransmission request (ARQ) and, in the future, for controlling dynamic assignment of time slots to each remote.
- Slot Guard Period: Time between each slot to prevent remotes from overlapping time slots due to clock skew and slow transmitter turn-off.
- Data Slot: Time slot to carry data. For the baseline design, each remote is statically assigned one or more data slots in uplink frames. All data slots in downlink frames are available to the hub.

As indicated above, only the uplink stream only needs a TDMA scheme. However, the downlink stream also uses the same scheme to simplify programming of the modem controller. Also, the hub is able to help synchronize all the remotes using the

synchronization control field in downlink frames. For the baseline design, the number of time slots and their assignment is fixed. Remotes only use time slots that are assigned to them. The hub has every time slot on the downlink stream assigned to it. Later designs will allow for more efficient dynamic time slot assignment. Data slot lengths and TMDA frame lengths will be chosen to prevent the per user data rate from dropping below 40 Mbps and to maintain reasonable bounds on delay and jitter for transported Ethernet frames (See Ch. 4.4. for the time specifications used in the baseline design.)

To efficiently use bandwidth, data slots are not be padded if data is available for transmission. Instead, the modem performs segmentation and reassembly (SAR). One or more Ethernet frames or Ethernet frame fragments (at the beginning or end) are placed in a data slot. The receiving modem retrieves the Ethernet frame and, if needed, reassembles fragments from the time slots and then pass the complete Ethernet frame out of the modem. The wireless link appear as an Ethernet link to the router and higher layer protocols. This setup allows the modem link operate without requiring an implementation of the Address Resolution Protocol (ARP) for IP. Data slots use a data link protocol that will have header fields to provide information for Ethernet frame reassembly, time slot encoding and link layer retransmission.

4.3. TDMA Timing Issues

4.3.1. Inter-slot and Inter-frame Gap Times

The inter-slot and inter-frame times need to be large enough to prevent data from different users from over lapping. The minimum distance and the maximum distance from the hub are used to calculate the size of the time gap. The following formula is used to calculate the time gap size.

$$TimeGapSize = \frac{(MaxDistance - MinDistance)}{3x10^8\ meter/sec} + T_{OFF} + T_{ON}$$

T_{OFF} and T_{ON} are the times for the radio or signal to turn off and are 25 ns each. For the baseline design, the minimum distance has been set to 1.25 km and the maximum distance has been set to 5 km. These values may change in later stages of the project. The time gap size for these distances needs to be 12.5 μseconds, if turn off/on times are ignored.

4.3.2. Turn Off/On Times

The remotes need to be able to turn off their transmitter signal when their time slot is not available. Originally, it was thought that the radios could be turned off. However, the oscillators in the radios may take up to several seconds to warm up before operating according to specifications. Other methods to turn off the signal to the transmitter using the modem are needed.

4.3.3. Synchronization Preambles

Each modem controller must synchronize with the signal that it is receiving to properly receive packets on the network. For a modem controller to synchronize a preamble must before each time slot in the TDMA frame. The length of the preamble used by the modems acquired by Virginia Tech is 1000 symbols. Since the network is using QPSK modulation, the preamble can be defined as 2000 bits in length.

4.3.4. Number of Synchronization Preambles on Downlink Stream

The synchronization preamble sent by the downlink stream only needs to be sent once per TDMA data frame, since the hub is the only station using the downlink stream. On the uplink stream, a synchronization preamble must be sent with every time slot. It has been proposed that the downlink stream frame format be changed to contain only one synchronization preamble. This would allow for the bandwidth on the downlink stream to be more efficiently used. For the baseline design, the downlink stream uses a synchronization preamble every time slot.

4.3.5. Sounding Interval

The broadband channel sounder (BBS) sounds the LMDS channel for a 20 ms period. While the BBS is sounding, data cannot be transmitted over the channel. Therefore, the BBS has been given a 20 ms frame in the TDMA super frame.

4.4. Current Timing Specifications

The timing specifications used in the baseline design of the protocol are presented in this section. For some of the timing specifications, a bit length and symbol length are provided in parentheses. Since the network will be using QPSK modulation for the baseline design, two bits is equal to one symbol.

$t_{SUPER\ FRAME}$= 713.9 ms = t_S+t_D

t_S=20.0291667 ms

t_D= 693.845 ms = 512 * t_{DATA_FRAME}

t_{FGP}=12.5 μs

$t_{DATA\ FRAME}$ = 1.355 ms = t_{FGP}+t_{SP}+t_{FCH}+8*t_{SGP}+8*t_{SLOT}

t_{SGP} = 12.5 μs

t_{SP}=16.6667 μs (2000bits, 1000 Symbols)

t_{FCH}=400 ns (48bits, 24 Symbols)

For 8 users
t_{SLOT}= 153.2 μs = t_{SP}+136.53 μs (16384b, 8192 symbols for Turbo encoded slot), effective bandwidth per user ~ 10Mbps

4.5. Data Link Protocol

This section describes the formatting of time slots in the TDMA frame. The data link protocol allows for Ethernet frame segmentation and reassembly, automatic repeat request, and multiple levels of forward error correction.

4.5.1. Acronyms

This section shows a listing of the acronyms that will be used for the remainder of the chapter.

ACK	acknowledgment
ACK No	acknowledgement number
ARQ	automatic repeat request
FB	frame border
FIN	finish
FEC	forward error correction
PAD	padding
SEQ No	sequence number
TOS	type of service
V	valid
TM_SLOT	time slot

4.5.2. Timeslot Header Format

This section explains the timeslot header format. See Figure 4.4 for the time slot header format and Ethernet fragment header diagrams.

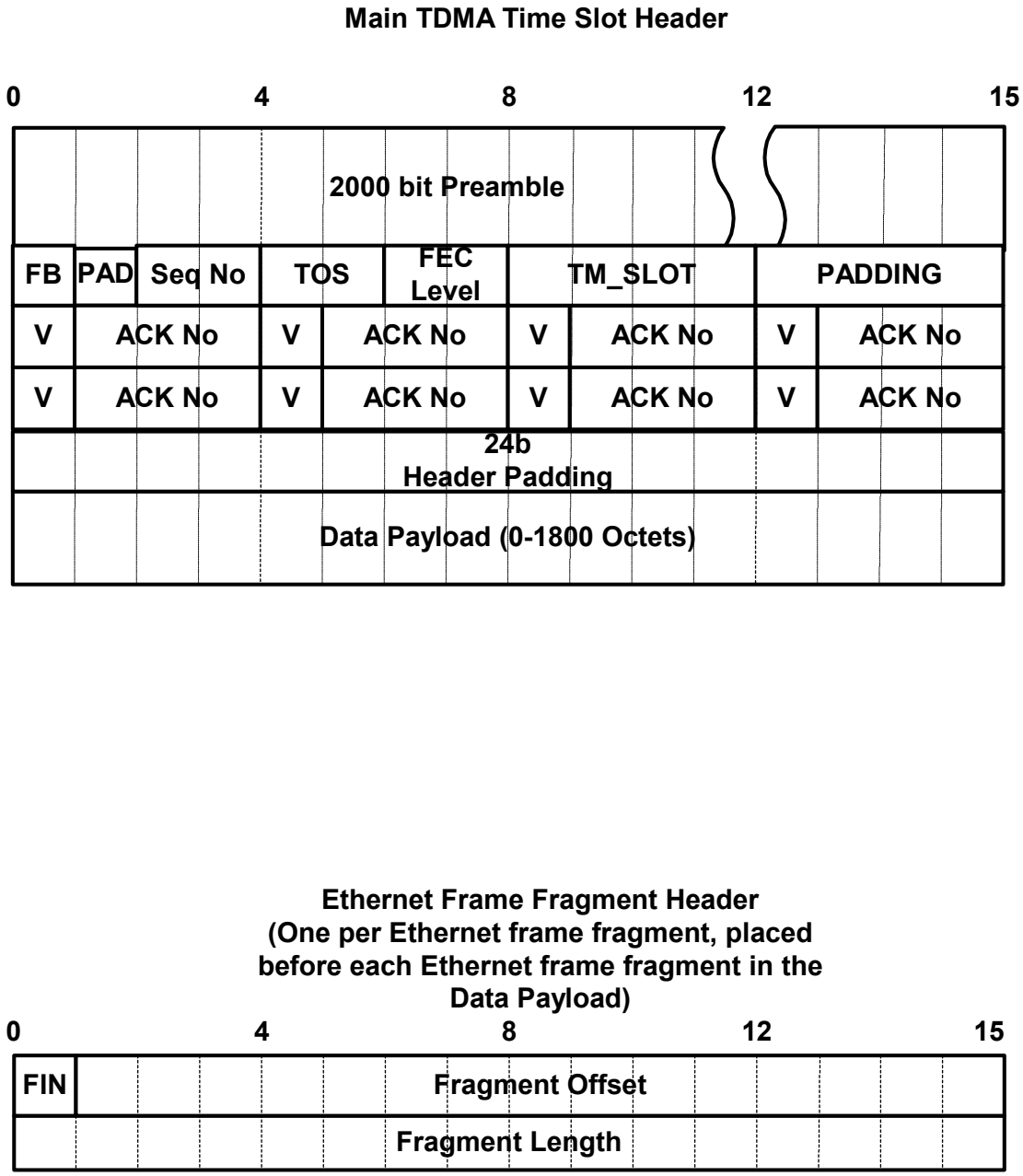

Figure 4.4. TDMA slot format.

4.5.2.1. Frame Border (FB) Field

The FB field marks the beginning of the frame transmitted by the hub on the downlink stream. The FB field is at the beginning of each TDMA frame control header and time slot header. If the FB field contains a 0, the receiver knows that a time slot is being received. If the FB field contains a 1, the receiver knows that a TDMA frame control header is being received and that a new data frame has begun.

4.5.2.2. Sequence Number (Seq No) Field

The Seq No field contains the sequence number of a time slot sent with acknowledged service. The sequence number field provides space for up to seven sequence numbers. However, only four sequence numbers are used for the baseline ARQ scheme for acknowledged service.

4.5.2.3. Type of Service (TOS) Field

The protocol allows for two types of services, acknowledged service and unacknowledged service.

4.5.2.3.1. Acknowledged Service

Time slots that are sent using the acknowledged service are acknowledged by the receiver after successfully receiving the packet. A selective-repeat ARQ scheme is used to retransmit packets that are not acknowledged in the next incoming TDMA frame.

4.5.2.3.2. Unacknowledged Service

The receiver will not acknowledge timeslots with the unacknowledged service indicated in the TOS field. The sender should fill the TOS field with $(00)_2$ when using this service. The sender should not fill the Seq No field.

4.5.2.3.3. Other Services

The TOS field has unused combinations to allow two more types of services. This space has been reserved to allow for future service types to be added to the protocol.

4.5.2.4. Forward Error Correction (FEC) Level Field

The FEC level field specifies the forward error correction coding level for the data payload of the time slot. The time slot header is always encoded with the highest level of encoding. The payload can be encoded with three different levels of encoding. The payload be also be transmitted with no error correction coding if the FEC field is filled with $(00)_2$.

4.5.2.5. Time Slot (TM_SLOT) Field

The TM_SLOT field contains the number of the time slot that the field is sent in. This field is used by nodes on the network when acknowledge service is being performed. The TM_SLOT field helps the receiver know to what time slot the received packet belongs. Since the receiver maintains separate ARQ tables for each time slot, the TM_SLOT field determines which ARQ table should be used.

4.5.2.6. Valid (V) Field

There are eight V fields, one for each time slot in a frame. The sender marks the V field with a 1 if it has filled the acknowledgment number field with a valid sequence number for a given time slot.

4.5.2.7. Acknowledgement Number (ACK No) Field

There are eight ACK No fields, one for each timeslot in a frame. The acknowledgement number always represents the next packet that the receiver is expecting to receive, i.e., it can be interpreted as a request field. The ACK No fields provide enough space for seven sequence numbers. However, only four sequence numbers are used for the ARQ scheme for the acknowledged service in the baseline design.

4.5.2.8. Header FEC Field

This field contains the coding bits to decode the header information at the receiver.

4.5.2.9. Payload FEC Field

This field contains the coding bits to decode the data payload information.

4.5.2.10. Data Payload

The data payload section contains Ethernet frame fragment headers and Ethernet frames. The data payload section can contain multiple Ethernet frame fragments. Each fragment has an Ethernet frame fragment header placed in front of it.

4.5.3. Ethernet Frame Fragment Header Format

This section explains the Ethernet frame fragment header format. See Figure 4.4 for the Ethernet frame fragment format diagram.

4.5.3.1. Finish (FIN) Field

The FIN field marks the ending of an Ethernet frame. The FIN field is filled with a zero for the first to *N*-1 Ethernet frame fragments. For the *N*th Ethernet frame fragment, the FIN field is filled with a one. If an entire Ethernet frame consists of only one fragment, the FIN field is filled with a one.

4.5.3.2. Fragment Offset Field

The fragment offset field provides the position of the Ethernet frame fragment in the original Ethernet frame. The position is given in octets.

4.5.3.3. Fragment Length Field

The fragment length field provides the length of the Ethernet frame fragment in octets.

4.5.4. TDMA Frame Control Header Format

This section explains the TDMA frame control header format. See Figure 4.5 for the TDMA frame control header format.

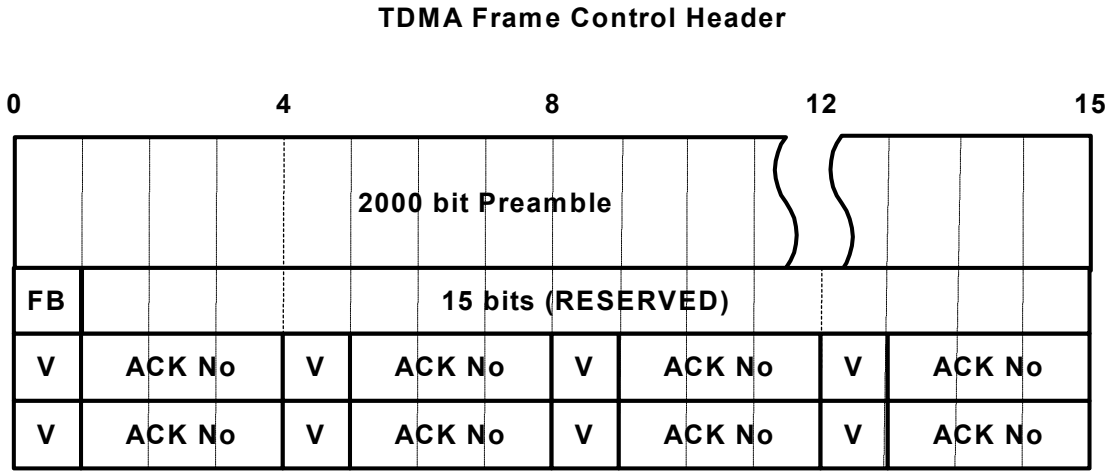

Figure 4.5. TDMA frame control header format

4.5.4.1Frame Border (FB) Field

The FB field marks the beginning of the frame transmitted by the hub on the downlink stream. The FB field is at the beginning of each TDMA frame control header and time slot header. If the FB field contains a 0, the receiver knows that a time slot is being received. If the FB field contains a 1, the receiver knows that a TDMA frame control header is being received and that a new data frame has begun.

4.5.4.2. Valid (V) Field

There are eight V fields, one for each timeslot in a frame. The sender marks the V field with a 1 if it has filled the acknowledgment number field with a valid sequence number for a given time slot.

4.5.4.3. Acknowledgement Number (ACK No) Field

There are eight ACK No fields, one for each timeslot in a frame. The acknowledgement number always represents the next packet that the receiver is expecting to receive. The ACK No fields provide enough space for seven sequence numbers. However, only four sequence numbers are used for the ARQ scheme for acknowledged service in the baseline design.

4.6. Service Functional Definitions

This section explains the operation of the two types of service provided by the protocol.

4.6.1. Acknowledged Service

Acknowledged service uses a go back-*N* ARQ scheme. The sending window size is fixed to four and the receiving window size is fixed to three. An ARQ scheme with these windows sizes performs better than a stop-and-wait ARQ scheme, since the sender does not need to wait for an ACK before sending the next packet. The window size gives the sender more time to detect the need for a retransmission, since it does not receive an ACK for a time slot until a data frame later. The sender should send no more than two time slots before receiving an acknowledgement from the receiver. The receiver fills the ACK No field for a time slot using the following formula.

ACK No = (*Received Sequence Number* + 1) mod 4

The receiver must also mark the V field for that time slot with a one to show that the ACK No field is valid. Upon receiving an acknowledgement for a packet that has already been sent, the sender retransmits the entire sending window of timeslots. Figure 4.6 shows an example of the operation of the go back-*N* ARQ scheme.

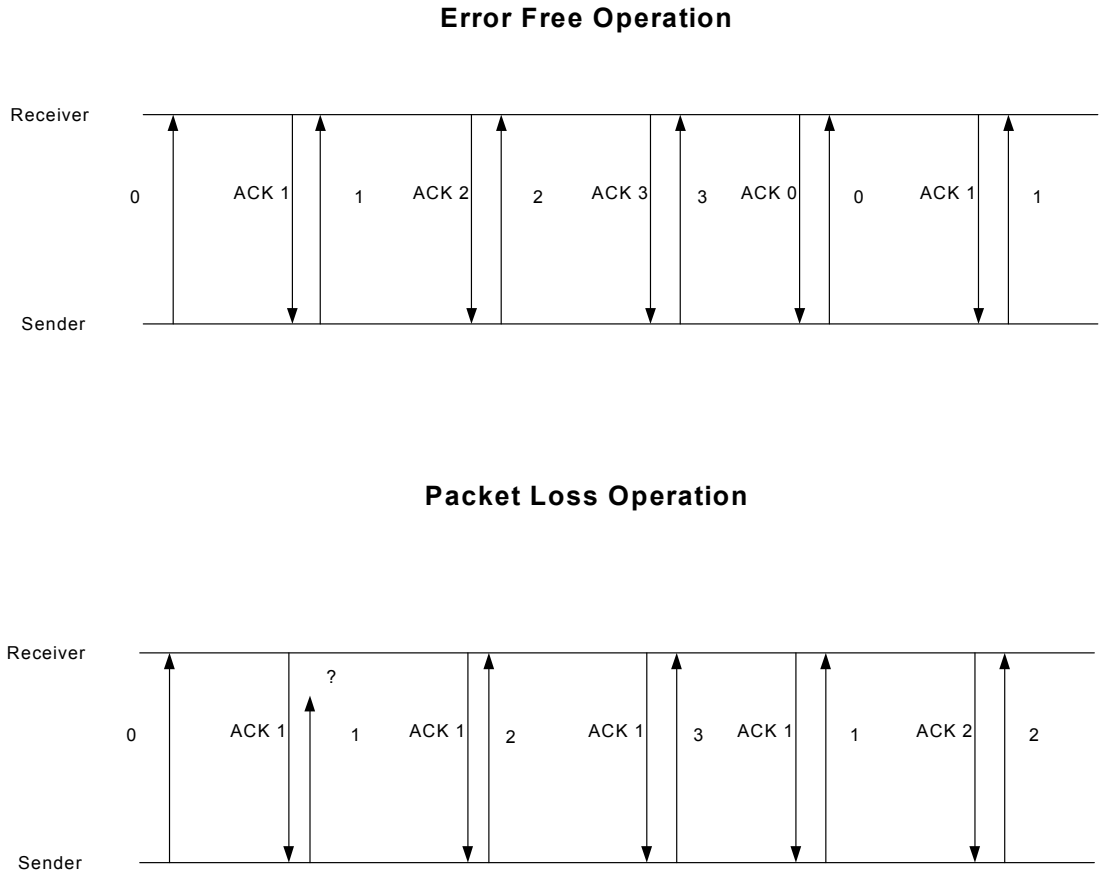

Figure 4.6. Go-Back-*N* ARQ scheme.

4.6.2. Unacknowledged Service

Unacknowledged service does not use an ARQ scheme. The sender of the time slot should fill the Seq No field with zeros. Upon receiving a time slot with unacknowledged service, the receiver must fill the V field for the timeslot with a zero to show that the sender can ignore the ACK No field.

The receiving or sending of a time slot with unacknowledged service must not affect ACK No and Seq No values for timeslots with acknowledged service at the receiver or the sender.

4.7. Proposed Forward Error Correction Codes

Each time slot in a TDMA data frame is encoded using Turbo and Reed-Solomon (RS) Codes [16]. The ordering of the encoding is as follows.

(1) The time slot header is encoded with a fixed RS code.

(2) The payload is then encoded using one of three possible RS codes (see Table 4.2)

(3) The header is placed in front of the encoded payload.

(4) Padding bits are added (see Tables 4.1 and 4.2 for amount of padding).

(5) The time slot is encoded using a fixed Turbo code (see Table 4.1.)

4.7.1. Header Encoding

Each header is encoded using RS(15,9) encoding. The header always uses the same fixed RS encoding. Since the header encoding is fixed, the receiver can decode the header and read the header to determine the encoding for the payload. Three bytes of padding are added to the header before the header is encoded. These three bytes of padding can be used later to extend the data link protocol's functionality.

4.7.2. Suggested Turbo Codes and RS Codes

This section presents the suggested Turbo Codes and RS codes to be used for the timeslot payloads. In this section, The TB Code Numbers refer to the Turbo Code presented in Table 4.1.

Data section sizes presented in Table 4.1 do not take into account the 120bits consumed by the time slot header. One header block and multiple RS code blocks are used to fill the data section of the Turbo code block. RS code block are not segmented across time slots. Padding bits are used to fill the remaining data section of the Turbo code block. Since only 7 complete RS blocks of 255B can fit in the data section of the suggested Turbo code blocks, each Turbo code has the same effect on the end-user

bandwidth. In the case of RS blocks of 200B, 8 complete blocks can fit in the data section of TB Code 1. However, 9 RS blocks of 200B can fit in the data section of Turbo Code 2 and 3.

Table 4.1. Three Suggested Turbo Codes

Code	Block Size (b)	Data Size (b)	Data Size (B)	Coding Rate	Coding Gain (dB)	Padding bits with 255B RS code blocks	Padding bits with 200B RS code blocks
TB Code 1 (128,120)x(128,120)	16384	14400	1800	0.88	6.6	0	1480
TB Code 2 (128,120)x(128,126)	16384	15120	1890	0.923	5.5	720	600
TB Code 3 (128,127)x(128,126)	16384	16002	2000.25	0.977	4.0	1602	1482

Table 4.2. Suggested RS Codes

Code	# of Code Blocks in a Time Slot	Coding Rate	Effective Bandwidth per user with 8 users on the network (Mbps) after Turbo Code has been applied to the entire time slot
(255,233)	7	0.937	9.946
(255,241)	7	0.945	10.03
(255,247)	7	0.9686	10.28
(200,181)	8 (TB Code 1), 9 (TB Code 2,3)	0.905	8.61 (TB Code 1), 9.69 (TB Code 2,3)
(200,191)	8 (TB Code 1), 9 (Code 2,3)	0.955	9.09 (TB Code 1), 10.23 (TB Code 2,3)
(200,195)	8 (TB Code 1), 9 (TB Code 2,3)	0.975	9.28 (TB Code 1), 10.45 (TB Code 2,3)

4.7.3. Recommendations

Turbo Code 2, (128,120) x (128,126), with RS code blocks of 200 B should be used. Code 2 provides more code gain than Turbo Code 3 but maintains the same amount of end user bandwidth. Turbo Code 1 does provide the greatest amount of code gain. However, the code brings bandwidth below acceptable levels for an eight-user network. Only RS(200,181) and RS(200,191) will be used. It was determined after developing the simulation model (described in Chapter 5) that the coding gain difference

between RS(200,195) and RS(200,191) was not large enough to have a noticeable difference in performance.

4.8. Summary

This chapter presented the design for the multiple access scheme and data link layer protocol to be used on the LMDS network and for the simulation model described in Chapter 5.

Chapter 5. Simulation Model

This chapter describes the simulation environment and simulation implementation.

5.1. Simulation Environment

The LMDS broadband network is simulated using OPNET version 8.0.C. One hub and eight remote modem controller units are modeled. The data link layer for the modems is implemented using an OPNET process model. The OPNET-provided TCP/IP Reno protocol stack is used in all the simulations. TCP/IP Reno was chosen since it is used in the BSD kernel [15] and has been used with other simulation models [14]. Simulations were run with various average channel bit error rates, each level of RS payload encoding, and with ARQ and without ARQ.

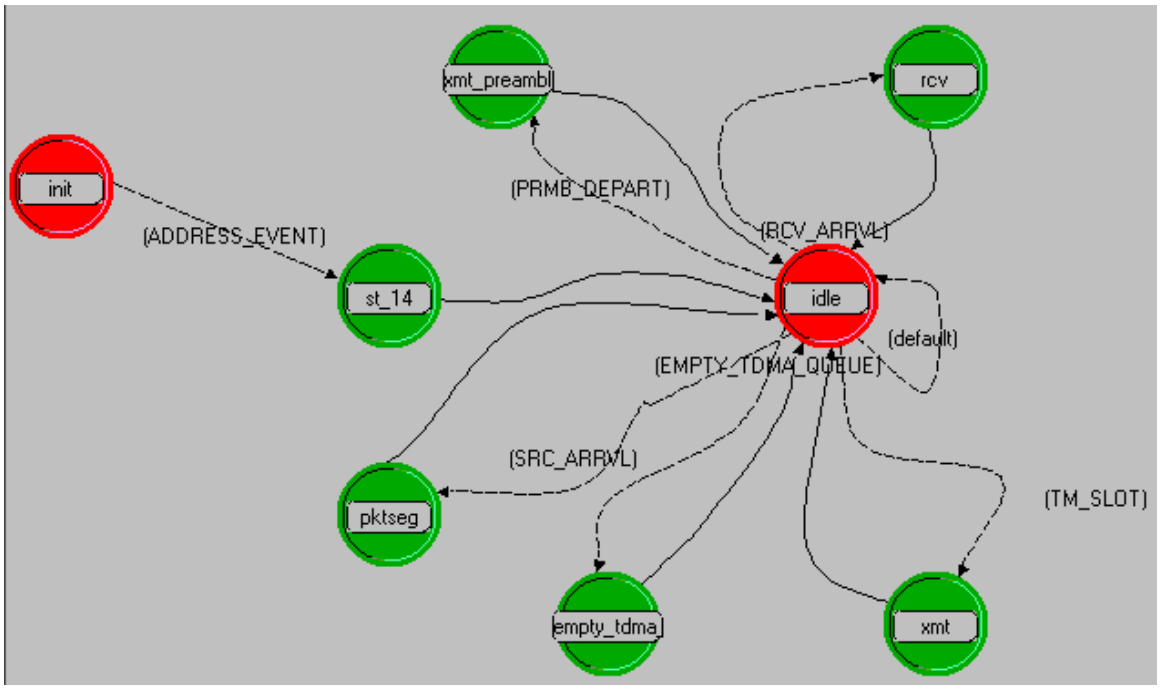

Figure 5.1. OPNET state diagram of the LMDS modem controller.

5.2. Comparison of Simulation Model to Proposed Real System

The simulation model is designed to imitate the proposed LMDS broadband system. The simulation model uses the multiple access scheme described in Chapter 4. The TDMA scheme is implemented using OPNET software interrupts. The software interrupts act like interrupts that would be used within the modem controller. The

forward error correction is simulated using the method described in Section 5.4. In a real system the higher FEC codes would cause a reduction in throughput because the number of data bits is reduced. To capture this effect in the simulation, payloads are padded with zeros that take the place of the number of correction bits used by the correction code. The padding bits reduce the useful data section of a payload packet, as would the correction bits in an FEC code. The data link layer protocol is fully implemented using the process model described in Section 5.3, as it would be in the proposed modem controller.

5.3. Simulated Modem Controller

Figure 5.1 shows the state diagram of the modem controller. The modem controller receives packets from the higher layer and performs any necessary packet segmentation. It then transmits data packets across the link using the data link layer described earlier. Upon receiving packets, the modem controller performs any necessary reassembly and passes data packet up to the higher layer.

5.3.1. PKTSEG State

The PKTSEG state receives packets from the higher layer. The packets are segmented and a small header needed to reassemble the packets is placed at the beginning of each segment. The segments are then placed together to form payloads for a time slot. Once enough data has arrived to form a complete payload, the payload is placed in a queue to wait for transmission. The PKTSEG state then sets a timer that controls the Empty Payload Buffer State.

5.3.2. EMPTY_TDMA State

In low traffic situations and at the beginning of TCP/IP sessions, packets from the higher layer are too small to form an entire time slot payload and may sit in the buffer used to build payload sizes packets for some time. After the timer set by the PPKTSEG state expires, the EMPTY_TDMA state pads the current time slot packet being formed with zeros to the TDMA payload slot size for the current coding scheme and places the time slot packet in a queue to wait for transmission.

5.3.3. XMT State

A time slot timer is set by the RCV state that launches the XMT state in time for that modem unit's time slot. The XMT state transmits a time slot packet from the transmission queue or the retransmission queue if the ARQ scheme is enabled and a retransmission is necessary. A header is built for the payload that contains the payload sequence number, level of the payload encoding, and acknowledgements for received time slot packets. The header and payload are encoded with the appropriate RS codes and then they are encoded in one Turbo code block. Before a packet is transmitted, a preamble is transmitted. The receiving modem controllers synchronize with the transmitted packets using the preambles. The remote modem controllers use the preambles to determine when their time slot for transmission is.

5.3.4. RCV State

The RCV state is awakened every time data arrives on the wireless link. The RCV state determines if a preamble or a packet has arrived. If a preamble arrives, the timers are set for the next transmitting time slot, if necessary. If a time slot packet arrives, the Turbo code block of the packet is decoded. The header is decoded. If the header is determined to be in error, the payload and header are destroyed. If the header is not in error, the payload header is decoded and placed in the reassembly buffer. The ARQ scheme tables are updated with the acknowledgements in the received header. After enough payloads are received to form a complete upper layer packet, the upper layer packet is put together and passed onto the higher layer.

5.3.5 XMT_PREAMBLE State

The modem controller process uses the XMT_PREAMBLE state only if the modem controller is at a hub. While in this state, the hub sets interrupts for when preambles and time slot packets need to be sent. The remote modem controller process relies on the preambles sent from the hub units to set interrupts for time slots.

5.4. Channel Error Model and FEC Model

For a QPSK channel, the bit error rate is equal to $\varepsilon(\gamma) = \frac{1}{2} erfc(\sqrt{\gamma})$; where γ is equal to the E_b/N_0 value of the channel [16]. In a Rayleigh Fading channel, the average

bit error rate is $\bar{\varepsilon} = \int_{-\infty}^{\infty} \varepsilon(\gamma) P(\gamma) d\gamma$ where $P(\gamma) = \frac{1}{\bar{\gamma}} e^{-\frac{\gamma}{\bar{\gamma}}}; \gamma \geq 0$ [16]. For a packet using FEC, a packet error occurs when a minimum number of bit errors occur. Let *N* be the number of bits per packet. Let *n* be the number of bit errors required for a packet error. Then the probability of a packet error is as follows [9].

$$\varepsilon_P(N,\gamma) = \binom{N}{n}(1-\varepsilon(\gamma))^{N-n}\varepsilon(\gamma)^n + \binom{N}{n+1}(1-\varepsilon(\gamma))^{N-(n+1)}\varepsilon(\gamma)^{(n+1)} +$$
$$\binom{N}{n+2}(1-\varepsilon(\gamma))^{N-(n+2)}\varepsilon(\gamma)^{(n+2)} + ... + \binom{N}{N-1}(1-\varepsilon(\gamma))^{(N-(N-1))}\varepsilon(\gamma)^{(N-1)} +$$
$$\binom{N}{N}(1-\varepsilon(\gamma))^0\varepsilon(\gamma)^{(N)} = \sum_{k=n}^{N}\left(\binom{N}{k}(1-\varepsilon(\gamma))^{N-k}\varepsilon(\gamma)^k\right).$$

Therefore, in a Rayleigh Fading Channel the average packet error rate is equal to $\bar{\varepsilon}_P(N,\gamma) = \int_{-\infty}^{\infty} \varepsilon_P(N,\gamma) P(\gamma) d\gamma$. Using the equation for $\bar{\varepsilon}_P(N,\gamma)$ and MATLAB, the average packet error rate was determined for each Eb/No value. The calculated packet error rates are shown in Table 5.1.

Table 5.1. Probability of Packet Errors for each of the FEC codes used in the OPNET Simulation

Eb/No (dB) / FEC Code	20dB	30dB	40dB	50dB	60dB	70dB
Header Encoding RS(15,9)	2.05709e-02	2.07765e-03	2.07972e-04	2.07993e-05	2.07995e-06	2.07995e-07
High FEC RS(200,181)	2.12597e-01	2.37274e-02	2.39956e-03	2.40227e-04	2.40254e-05	2.40257e-06
Low FEC RS(200,191)	2.58275e-01	2.95749e-02	2.99902e-03	3.00321e-04	3.00363e-05	3.00368e-06
No FEC RS(200,200)	3.74852e-01	4.61519e-02	4.71663e-03	4.72694e-04	4.72797e-05	4.72807e-06

To simulate packet errors on the network, uniformly distributed pseudo-random numbers between [0,1) are generated. If the random number is less than or equal to the packet error probability for the current E_b/N_0 value and the coding level of the packet,

then the packet is considered to be in error. This process is performed for both packet headers and payloads. If the header of a packet is found to be in error, the payload of the packet is considered to also be in error.

5.5. Traffic Model

For the simulation experiments, one remote station attempts to send a file to a server at the hub. The file size was chosen so the simulation had time to reach steady state. Only one remote was chosen to perform a file transfer to reduce simulation time. The real system will typically carry more traffic than just one TCP/IP connection. However, the goal of the model is not to simulate traffic conditions that might be present in the real system. Instead, the goal of the model is to determine the relative effect of different FEC codes, ARQ, and varying channel conditions. Also, if more traffic were simulated for the experiments, it would be harder to determine if throughput reductions were caused by packet errors or congestion in the network.

5.6. Simulation Experiments

For each simulation run, a remote station uses TCP/IP to transfer a file that is 2,145 megabytes (MB) in size to a server at the hub station. Simulations were run for the following E_b/N_0 values: 20 dB, 30 dB, 40 dB, 50 dB, 60 dB, and 70 dB. The E_b/N_0 range covers bit error rates from 10^{-2} to 10^{-8}. The E_b/N_0 value was fixed for the duration of a simulation run. Two sets of simulations were run. One set used link layer ARQ and one set did not use link layer ARQ. Within each of these sets, simulations were run for every FEC level and E_b/N_0 value. These simulation factors are summarized in Table 5.2. This approach provided a full factorial experiment [11] with 36 different simulations. Each simulation was run until steady state was reached. Each simulation was replicated three times using different random seeds to calculate 95% confidence intervals.

Table 5.2. Factors for Full-Factorial Simulation Experiment

Factor	*Values*
E_b/N_0 (dB)	20, 30, 40, 50, 60, 70
ARQ	Off, On
FEC	Off, Low, High

5.7. Simulation Validation

Since the modeled system is still in development, the validity of results could not be confirmed with data collected from a real system. Therefore, the validity of exact throughput and end-to-end delay values cannot be confirmed. However, changes in throughput and changes in end-to-end delay values can be validated against expected system behavior.

Section 5.6.4. shows results from the a constant bit rate test using UDP traffic. Throughput achieved at low bit error rates matches the amount of traffic generated at the source (see Figure 5.6.) For a real system operating in low bit error rate channels, the throughput received from any source should approach the throughput generated by that source. As the bit error rate increases, the throughput received is lower than the traffic genratored by the source for UDP traffic using link ARQ and not using link ARQ. However, UDP traffic using link layer ARQ achieves a higher throughput for all bit error rates. On the real system, it is expected that link layer ARQ will improve throughput for protocols that do not use any ARQ schemes.

End-to-End delay for UDP traffic not using link layer ARQ remains about the same for all bit error rates (see Figure 5.5.) On the real system, traffic with a constant bit rate should have the same end-to-end delay, since the interarrival time for the packets does not change and the data rate of the network is fixed. However, for UDP traffic using link layer ARQ the end-to end delay should increase as the bit error rate increases in the real system because more link layer transmissions are performed as the bit error rate increases. Constant bit rate traffic in the simulation model experiences an increase in end-to-end delay as the real system would.

5.8. Simulation Verification

To ensure that the simulation model operates as designed, several tests were performed to verify its operation. This section describes the tests performed and presents results from some of these tests.

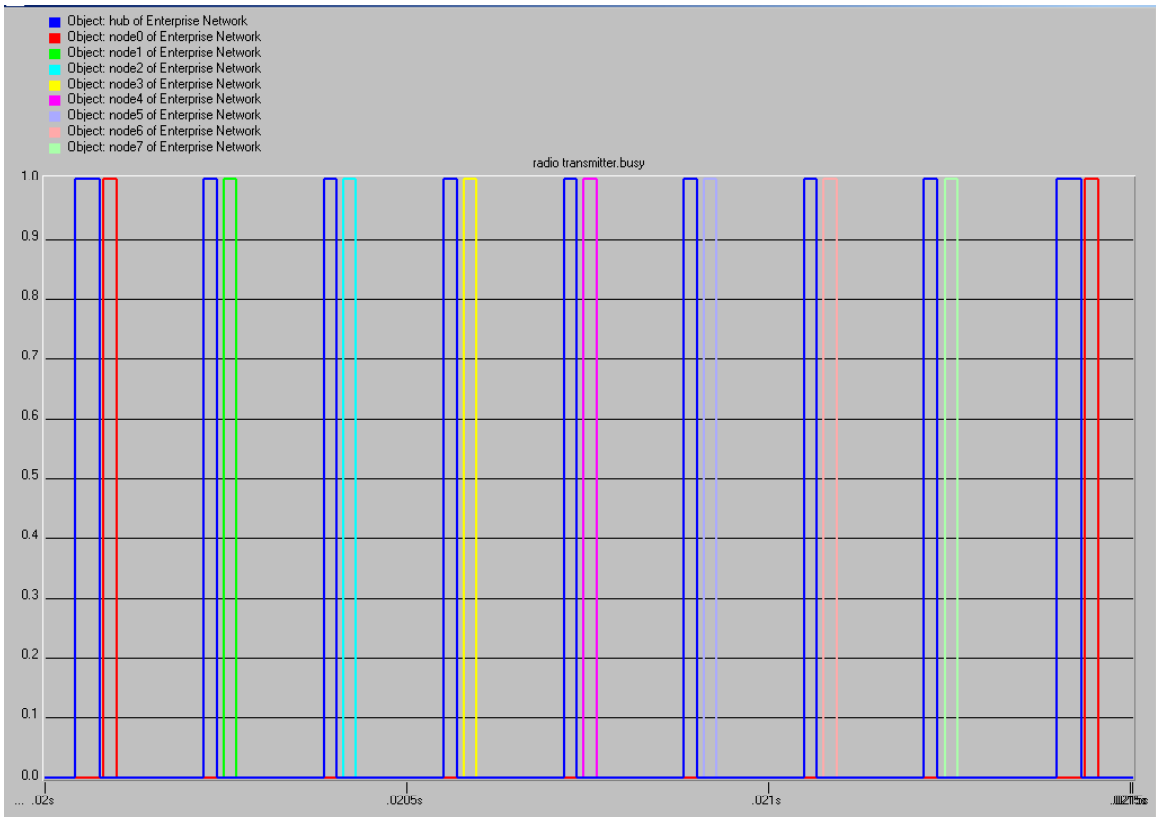

Figure 5.2. Radio transmitter activity on the uplink and downlink channels.

5.8.1. TDMA Scheme Operation

After the TDMA scheme had been built into the modem controller process, the TDMA scheme was verified. In a TDMA scheme each remote must transmit at a different time, or packets will collide on the network. The activity at each transmitter was monitored to see when each node on the network was transmitting. In Figure 5.2, the hub transmitter activity can be seen as the darkest or the blue line. The hub transmits once every time slot because no other stations are on that frequency channel. The other lines show the remote transmitters taking turns transmitting on the channel. The figures also show the effects of propagation delay on the network, since each node lags in time behind the hub station. After all 8 remote nodes have finished transmitting, the cycle starts over at the beginning with remote node 0.

The sounder frame operates within the TDMA scheme as well. The sounder frame is 20 ms. During this time, no nodes on the network can transmit. Figure 5.3 shows the silence in transmitter activity for the sounder frame.

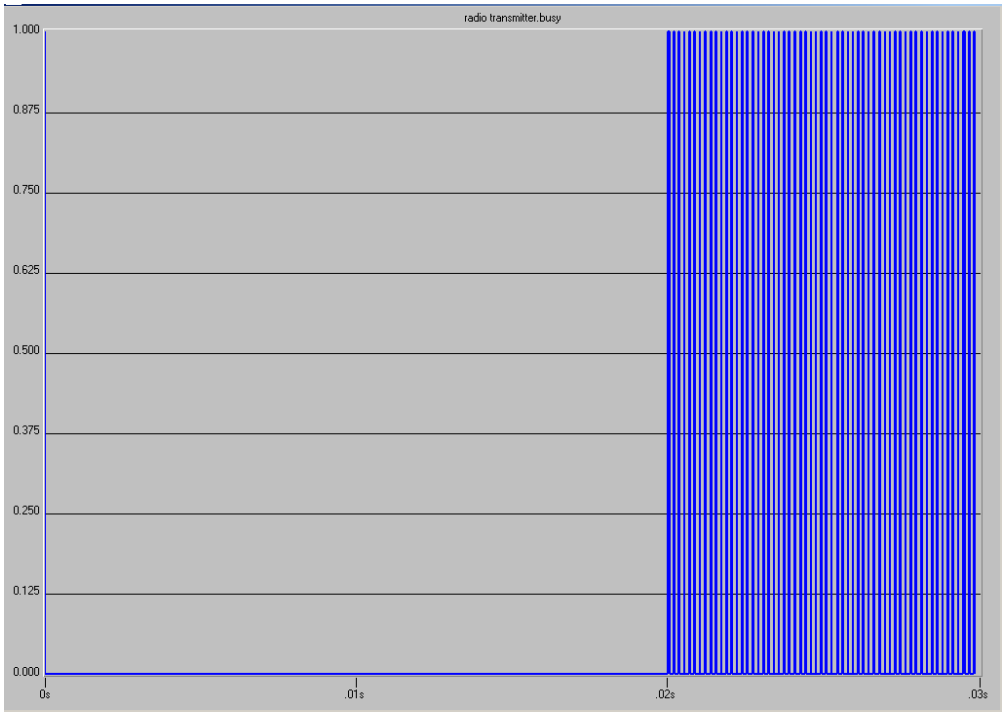

Figure 5.3. 20 ms radio silence for sounder frame.

5.8.2. Throughput Operation

According the design of the MAC and data link layer protocols, each remote can transmit up to 10 Mbps of data under error-free conditions. To confirm that the simulated data link layer met design specifications, a load test was performed on the simulation model. Eight traffic generators programmed to produce 11 Mbps at a constant rate were placed at each of the remotes. At the hub station, a traffic generator was placed that was programmed to generate 81 Mbps of data. Traffic sinks measured the received amount of data at each node on the network. Each node on the network should receive close to 80 Mbps of data, because the hub receives data from all 8 remotes and each node receives data from the hub. During the load test the average throughput received at the node approached 80Mbps in less than 10 seconds, as shown in Figure 5.4.

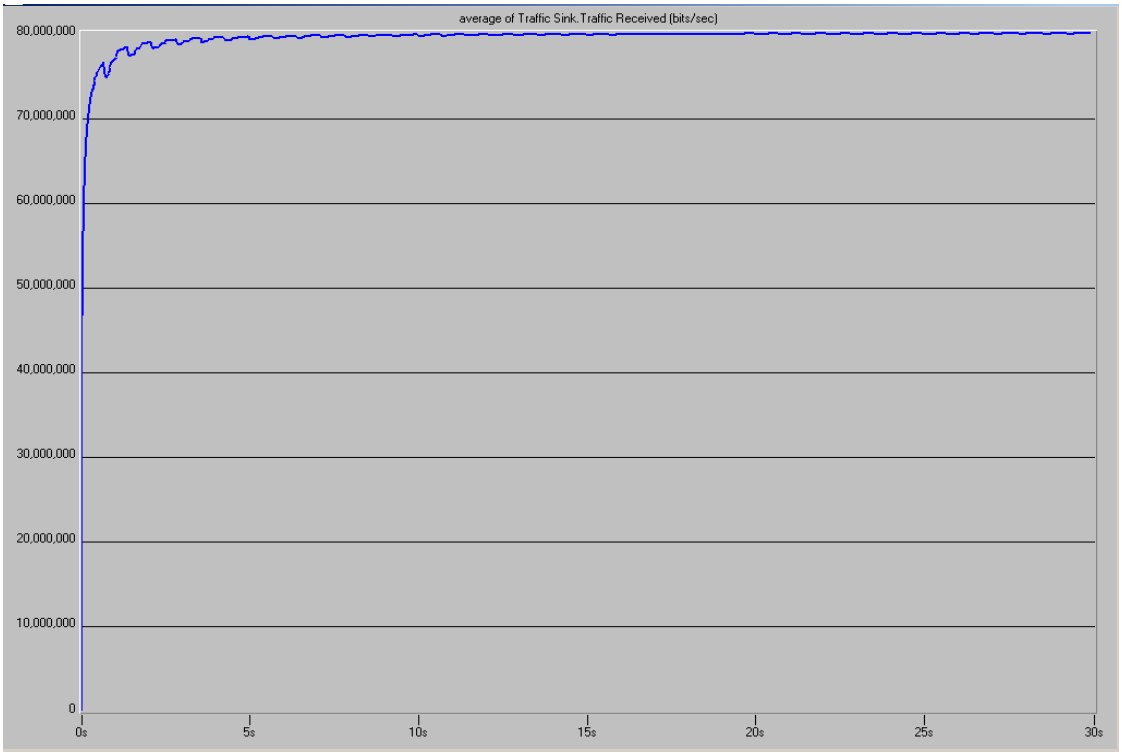

Figure 5.4. Average throughput achieved during load testing.

5.8.3. Automatic Request Scheme Operation

To verify the operation of the automatic request scheme, every node was configured to output the following state information for the ARQ scheme: (i) sequence number of a packet when it arrives, (ii) acknowledgement number for each acknowledgement received, (iii) sequence number of each packet transmitted, and (iv) sequence number of each packet that was transmitted. After examining these logs at each of the remotes, it was determined that the ARQ scheme operated correctly.

During the simulation experiments, it was discovered some packets had unexpected long delays (greater then 6 s) when the ARQ scheme was enabled. Under low traffic conditions when a packet is lost because of an error, it may take several seconds before enough traffic arrives to fill the sending window at the transmitter. The packet that was lost is held during this time at the transmitter and is not retransmitted because the sending window is not full. Once the sending window fills, the sender realizes the packet has been lost and retransmits the packet. To correct this problem, a fast retransmit mechanism was implemented. When the queue at the sender is empty, packets are retransmitted if acknowledgements have not been received for them.

Appendix D shows a log of one the simulations with the fast retransmission fix. After the implementation for the ARQ scheme was corrected, simulations using the ARQ scheme were run again. Chapter 6 presents results for simulations using the corrected ARQ scheme.

5.8.4. Constant Bit Rate Test

To verify end-to-end delay statistics gathered from the simulation, simulations with a constant UDP traffic source were run with no FEC. The traffic source generated UDP packets every 100 ms with a size of 4,800 bits. The service rate on the network is not truly deterministic because the sounder frame interrupts service on the network. Before simulations were run, a rough estimate of delay on the network was calculated.

A UDP packet of 48,000 bits will be segmented into four time slots when using no FEC encoding. If a packet arrives during the 512 data frames section of the super-frame, the maximum service time for a UDP packet would be the length of four data frames plus the transmission delay.

$$4 * t_{DATAFRAME} + T_{TRANSMISSIONDELAY} = 4 * 1.335ms + 120\mu s = 5.54ms$$

For a packet that arrives at the beginning of a sounder frame, the maximum service time would be:

$$t_{FGP} + t_{SP} + t_s + 4 * t_{DATAFRAME} + t_{TRANSMISSIONDELAY} =$$
$$12.5\mu s + 16.7\mu s + 20ms + 4 * 1.355ms = 25.4492ms$$

Since the service time for a single packet is always less than the inter-arrival time, a UDP packet will not have any queuing delay. Over one super frame, seven UDP packets will arrive. Only one of the packets arrives at the beginning of a super-frame. The remaining six packets arrive during a data frame. The average ETE delay for the super frame would be:

$$\frac{25.4492ms + 6 * 5.54ms}{7} = \frac{58.6892ms}{7} = 8.384ms$$

Figure 5.5 shows the end-to-end delay results for UDP traffic simulations using link layer ARQ and not using link layer ARQ. For simulations with low bit error rates, the end-to-end delay approaches the calculated estimate. For connections using link layer ARQ, the end-to-end delay increases because of the number retransmission needed to transmit a packet across the link.

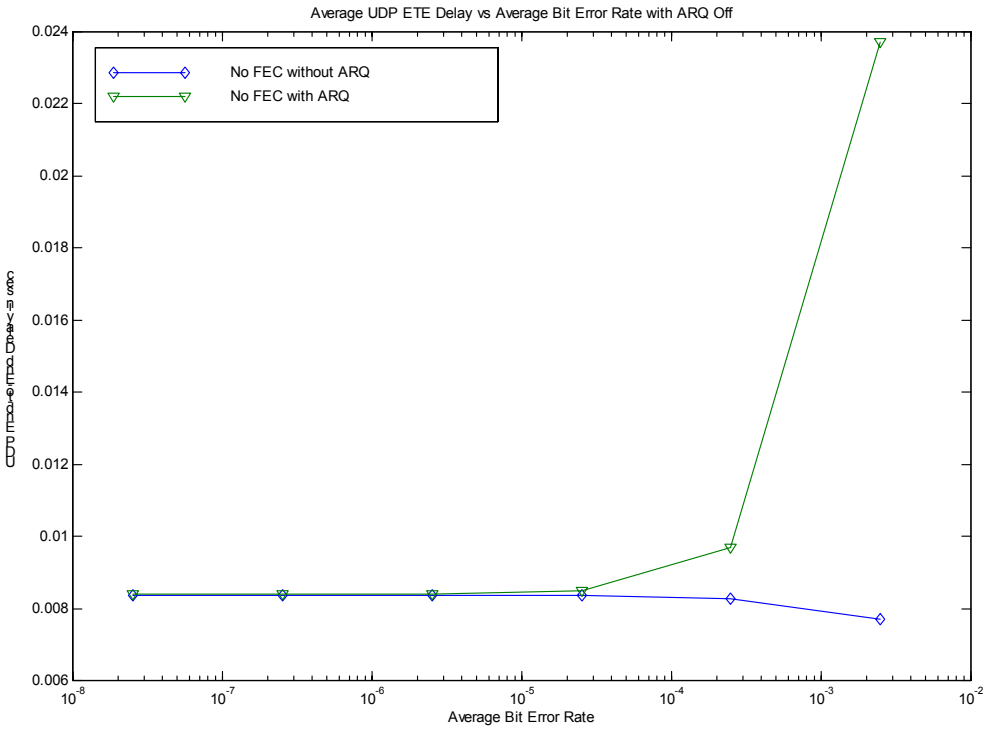

Figure 5.5. End-to-end delay for UDP traffic using no FEC encoding.

The average throughput for simulations with low bit error rate approached 480 kbps generated at the source (see Figure 5.6). As the bit error rate increases, the throughput decreases for traffic using link layer ARQ and without link layer ARQ. However, link layer ARQ provides greater throughput with greater end-to-end delay than traffic without link layer ARQ. Tables 5.3 and 5.4 show simulations results with confidence intervals.

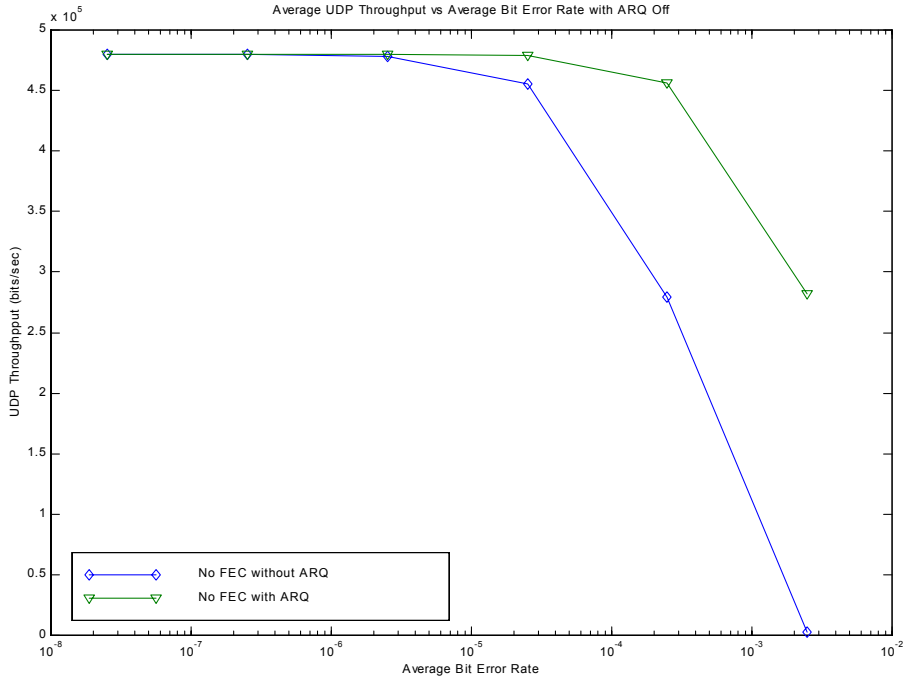

Figure 5.6. Average UDP throughput versus average bit error rate.

Table 5.3. End-to-End Delay Results for UDP Traffic

Eb/No (BER)	Average End-to-End delay (sec)	Average Standard Deviation (sec)	95% Confidence Interval (sec)	ARQ scheme
20dB(2.48e-3)	7.703292e-3	6.578881e-3	±2.968946e-5	Disabled
20dB(2.48e-3)	2.372596e-2	2.024551e-2	±1.479232e-4	Enabled
30dB(2.49e-4)	8.278815e-3	6.148618e-3	±8.093496e-6	Disabled
30dB(2.49e-4)	9.692847e-3	7.511389e-3	±2.742723e-5	Enabled
40dB(2.50e-5)	8.361384e-3	6.091762e-3	±2.268870e-6	Disabled
40dB(2.50e-5)	8.501640e-3	6.225823e-3	±1.337335e-5	Enabled
50dB(2.50e-6)	8.373391e-3	6.094809e-3	±1.167364e-6	Disabled
50dB(2.50e-6)	8.410532e-3	6.146556e-3	±5.986286e-6	Enabled
60dB(2.50e-7)	8.374981e-3	6.094499e-3	±1.953855e-7	Disabled
60dB(2.50e-7)	8.392875e-3	6.124105e-3	±2.085176e-6	Enabled
70dB(2.50e-8)	8.375039e-3	6.094218e-3	±0	Disabled
70dB(2.50e-8)	8.391069e-3	6.123049e-3	±0	Enabled

Table 5.4. Average Throughput Results for UDP Traffic Simulations

Eb/No (BER)	Average Throughput (bps)	Average Standard Deviation(bps)	95% Confidence Interval (bps)	ARQ scheme
20dB(2.48e-3)	2.194831e+3	1.638366e+4	±1.920805e+2	Disabled
20dB(2.48e-3)	2.817159e+5	3.533385e+5	±2.727838e+3	Enabled
30dB(2.49e-4)	2.790754e+5	3.507720e+5	±3.528171e+3	Disabled
30dB(2.49e-4)	4.562080e+5	3.469046e+5	±1.202022e+3	Enabled
40dB(2.50e-5)	4.549278e+5	6.694263e+5	±2.376602e+3	Disabled
40dB(2.50e-5)	4.785597e+5	6.783115e+5	±2.258680e+2	Enabled
50dB(2.50e-6)	4.777595e+5	6.780262e+5	±8.537008e+1	Disabled
50dB(2.50e-6)	4.800000e+5	1.073313e+6	±1.707402e+2	Enabled
60dB(2.50e-7)	4.798400e+5	6.787659e+5	±2.258680e+2	Disabled
60dB(2.50e-7)	4.801067e+5	1.073408e+6	±0.00	Enabled
70dB(2.50e-8)	4.801067e+5	6.788602e+5	±8.840676e-3	Disabled
70dB(2.50e-8)	4.801067e+5	1.073408e+6	±0.00	Enabled

5.9. Summary

This chapter presented the simulation design, validation, and verification. After the simulation was verified and validated, the simulation experiments described in section 5.4 could be run. Chapter 6 presents the results of these simulation experiments.

Chapter 6. Results

This chapter presents results from the simulation model described in Chapter 5. The simulation model used the data link layer protocol described in Chapter 4. An explanation of why current metrics produced from the broadband sounder data are not useful is given. We propose a new metric, carrier-to-interference ratio, to be calculated from the broadband sounder metrics. Algorithms using CIR are proposed to improve TCP/IP performance over wireless networks. Figures for this section are provided in Appendix E. A tables of results is provided in Appendix F.

6.1. Results for Connections Not using Link Layer ARQ

Figure E.1 shows the average throughput without link layer retransmissions for different levels of error coding as a function of average bit error rate. For connections using higher levels of error-correction coding, TCP/IP throughput is higher at high bit error rates. As the bit error rate decreases to 10^{-7}, connections with low FEC outperform connections with high FEC. Therefore, low FEC is sufficient for channels with a bit error rate of 10^{-7}. The connections using low FEC coding have more data bits per packet than connections using high FEC coding. Therefore, the connections with high FEC cannot provide the same throughput. However, as the bit error rate increases, low FEC provides lower throughput than connections with high FEC because the low FEC coding cannot recover from as many errors as the high FEC. These results suggest that for connections not using link ARQ low FEC is sufficient for bit error rates from 10^{-8} to 10^{-6} and high FEC provides the best performance for bit error rates greater than 10^{-6}. TCP/IP throughput could not be measured for bit error rates greater than 10^{-3} because TCP/IP could not maintain a connection under such error-prone channel conditions.

If the sounder were to be used to maximize throughput across the LMDS link, low FEC level should be used for connections until the BER increases beyond 10^{-5}. After the BER is greater than 10^{-5}, the connections should be switched over to a high FEC level to maintain a greater throughput.

Figure E.2 shows the end-to-end delay of packets across the link. The end-to-delay measurements represent the time from when a higher layer packet enters the modem controller at the transmitting end until the higher layer packet is passed to the

higher layer at the receiving end. End-to-end delay includes any queuing delay in the modem and segmentation and reassembly time needed to transmit the packet. Even though connections with higher FEC provide higher throughput, the overall link layer end-to-end delay increases.

As the bit error rate increases, end-to-end delay starts to increase. When TCP/IP Reno, which is used in this model, receives multiple acknowledgements for a particular packet, it performs a fast retransmission for the packet without reducing the window size and, hence, the throughput [15]. For bit error rates, between 10^{-7} and 10^{-5}, it is possible that only one packet out of a group of packets is in error. Therefore, TCP/IP would receive multiple requests for that one packet. To recover from this, TCP/IP would perform a fast retransmission. Since throughput is not reduced during fast retransmission, the queue in the modem controller increases in size. The increase in queue size in the modem controller causes an increase in overall end-to-end delay for packets on the channel. For connections using no FEC and low FEC, TCP/IP Reno performs more fast retransmissions than it does for a connection using high FEC. Therefore, the end-to-end delay is larger for connections using no FEC and low FEC.

As the bit error rate increases beyond 10^{-5}, TCP/IP Reno cannot rely on fast retransmissions to recover from packet errors and the connection undergoes congestion control. Throughput is reduced by half during congestion control [15]. Since TCP/IP reduces throughput, the queue size in the modem controller decreases, causing end-to-end delay to decrease as the bit error rate increases. Since higher levels of FEC coding do not provide as much bandwidth for higher layer packets, connections using higher coding levels experience higher end-to-end delay.

The simulation data also shows that information from the sounder can be used to control end-to-end delay on the network. For bit error rates greater than 10^{-6}, connections with no FEC could be used to reduce delay for higher layer protocols. However, connections using no FEC would experience a larger packet error rate than connections using FEC.

Statistics on the average delay jitter, i.e., variation in end-to-end delay, were also collected (see Figure E.3). Overall, jitter increases as the bit error rate increases. Connections using high FEC experience greater jitter than connections using low FEC or

no FEC coding. Figure E.3 shows that switching to higher levels of FEC increases jitter in end-to-end delay. The results show that switching from one coding scheme to another does not improve jitter as bit error rate increases.

6.2. Results for Connections using Link Layer ARQ

Overall TCP/IP connections with link layer ARQ showed approximately 150 Kbps greater throughput than connections without link layer ARQ. A low level of FEC provides equal or better throughput than a high level of FEC for bit error rates less than 10^{-3}. Low-level FEC provides better throughput because more data bits can fit into a packet with low FEC than a packet with high FEC. However, as the bit error rate increases beyond 10^{-3}, the higher level of FEC coding allows TCP/IP to achieve higher throughput because it can recover from more errors than in a system using a low-level of FEC. Figure E.4 shows TCP/IP performance using link layer ARQ.

The affect of using link layer ARQ can be seen when the congestion window size over time is examined. Figure E.5 shows the congestion window size as a TCP/IP connection starts up for a connection using a high level of FEC at an E_b/N_o of 60 dB. When link layer ARQ is not in use, TCP/IP reduces its congestion window size each time a packet loss is detected. The connection using ARQ is able to hide packet loss from the TCP/IP layer. Therefore, it does not suffer from congestion window size reduction. Since the congestion window size is never reduced, TCP/IP throughput is higher for connections using link layer ARQ.

Even though TCP/IP throughput improves when using link layer ARQ, end-to-end delay for a connection increases as the BER increases, as shown in Figure E.6. The increases becomes more evident as the BER becomes greater than 10^{-4}. The end-to-end delay improves for connections using higher levels of FEC coding because fewer link layer transmissions need to be performed for stronger FEC coding. Jitter in end-to-end delay also increases as the BER increases (see Figure E.7). Like end-to-end delay and average throughput, switching to a higher level of FEC coding does improve average jitter in end-to-end delay.

6.3. Algorithms Using the Initial Sounder Metric

The program developed (see Section 3.4.1) to analyze the broadband sounder data calculates coherence bandwidth, mean excess delay, and RMS delay spread of the channel. The mean excess delay is the expected or first moment of delay for multipath components on the channel [16]. The RMS delay spread is the standard deviation of the multipath components on the channel [16]. The coherence bandwidth delay is the inverse of the RMS delay spread divided by 50 for 90% correlation [16]. However, none of these measurements were found to be useful for controlling the configuration of the data link layer protocol. Each of the measurements from the sounder reflects how far apart or how close multipath components are to each other. These measurements do not reflect how much each multipath component interferes with received signal.

If the sounder could provide an E_b/N_o value, the bit error rate on the channel could be calculated. Having the bit error rate of the channel provides the algorithm for controlling the data link layer protocol with more information of the channel performance for data transmission.

Two algorithms for the adaptive data link layer protocol using the average BER as the main parameter are presented in the following two sections. The first algorithm presented maximizes throughput. The second algorithm minimizes expected end-to-end delay.

6.3.1. Algorithm for Maximizing Throughput

As indicated in Section 6.2, link layer ARQ should be enabled for all bit error rates. The remainder of the algorithm is as follows.

```
while(wireless link is in operation )
{
        Wait for bit error rate from the broadband sounder
        if ( Bit Error Rate  > 10^-3 + ε )
                {
                Set FEC encoding to high FEC.
                }
        else if ( Bit Error Rate < 10^-3 – ε )
                {
                Set FEC encoding to low FEC
                }
}
```

This algorithm will cause the link to use a low level of FEC while the bit error rate is below 10^{-3}. According to simulation results, high FEC provides the best throughput for bit error rates greater than the 10^{-3}. Therefore, the algorithm switches to high FEC for bit error rates greater than 10^{-3}. The value ε is a sensitivity parameter to prevent the algorithm from switching states at a rapid rate when the bit error rate stays near 10^{-3}.

6.3.2. Algorithm for Minimizing End-to-End Delay

The algorithm for minimizing end-to-end delay modifies both the FEC level and selectively enables or disables ARQ. The algorithm is as follows.

```
while ( wireless link is in operation )
{
        Wait for bit error rate from the broadband sounder
        if ( Bit Error Rate < 10^-4 – ε )
                {
                Enable ARQ scheme
                Set FEC encoding to no FEC
                }
        else if ( Bit Error Rate > 10^-4 + ε and Bit Error Rate < 10^-3 – ε)
                {
                Disable ARQ scheme
                Set FEC encoding to no FEC
                }
        else if ( Bit Error Rate > 10^-3 + ε)
                {
                Enable ARQ scheme
                Set FEC encoding to high FEC
                }
}
```

The algorithm is designed to minimize end-to-end delay, however, throughput and packet loss may be adversely affected. Simulation results showed that for a BER less than 10^{-4}, no FEC encoding with ARQ provides the lowest end-to-end delay. For a BER between 10^{-4} and 10^{-3}, no FEC encoding without ARQ provides the lowest end-to-end delay. High FEC encoding with ARQ provides the lowest end-to-end delay for BERs greater than 10^{-3}. The value ε is a sensitivity parameter to prevent the algorithm from switching states at a rapid rate when the bit error rate stays near 10^{-3} and 10^{-4}.

6.4. Algorithms Using an Alternative Sounder Metric

Since RMS delay spread and mean excess delay provide information only about the distance between multipath components, another metric from the broadband sounder is needed. The broadband sounder provides the power profile delay for a pulse sent across the channel [17]. If it is assumed that the LMDS channel is interference limited, then power from multipath components, *I*, is much greater than the power from noise components, *N*, on the channel. Since $I >> N$, power from multipath components can be considered to be have the most effect on channel degradation if the power of the signal, *C*, remains constant [9].

Multipath components on the channel are time shifted reflected images of the original signal [16]. Since these components come from the same transmitter and are interference for the original signal, the multipath components can be viewed as co-channel interference [9]. Co-channel interference is approximated by additive white Gaussian noise [5][12]. Since co-channel interference can be approximated by additive white Gaussian noise, a ratio between the original signal power, *C*, and the power of the multipath components, *I*, can be used to determine a bit error rate [9]. This ratio is called carrier-to-interference ratio, $CIR = C/I$.

CIR can be calculated from sounder data using the algorithm described in Section 6.4.1. The *CIR* can be used to determine the link configuration for the optimum throughput and end-to-end delay using graphs created from the simulation data shown in Figures E.8. and E.9.

6.4.1. Algorithm for Calculating Carrier to Interference Ratio

The carrier-to-interference ratio is calculated using the following algorithm.

Retrieve data collected from broadband sounder and store in SounderData;

```
/*Remove Noise from SounderData*/
For (I=0; I < length(SounderData) ; I=I+1)
{
        if ( SounderDatat[I] < 0.1) /*Removes all points below –10dB*/
                SounderData[I]= 0;
}
```

Maxindex = Search for index of maximum point(SounderData);

```
/*Determine amount of power from carrier signal by summing all the points that
make up the carrier signal*/

Carrier= 0;

/*Pulse Width is given as a parameter to algorithm*/
For (I = Maxindex – PulseWidth/2 ; I < Maxindex + PulseWidth/2; I=I+1)
{
        Carrier = Carrier + SounderData( I);
}

/*Determine amount of power from interference signals by summing all
remaining points*/

Interference = 0;

For ( I= 0; I < Maxindex – PulseWidth/2; I=I+1)
{
Interference = Interference + SounderData(I);
}

For (I= Maxindex + PulseWidth/2; I < Length(SounderData); I =I+1)
{
Interference = Interference + SounderData(I);
}

Absolute_CIR = Carrier /Interference;
CIR = Convert To dB (Absolute_CIR);
```

6.4.2. Algorithm for Maximizing Throughput

CIR, as calculated using the algorithm of Section 6.4.1, can be used to configure the "best" link layer ARQ and FEC configuration. To maximize throughput, ARQ needs to be enabled for all values for *CIR*, as indicated in the results of Figure E.8. The remainder of the algorithm is as follows.

```
while ( wireless link is in operation)
{
        Wait for bit error rate from the broadband sounder
        if ( CIR < 33dB – ε )
                {
                Set FEC encoding to high FEC
                }
        else if ( CIR > 33dB + ε )
                {
                Set FEC encoding to low FEC
```

```
                }
}
```

This algorithm will cause the link to use a low level of FEC while the *CIR* is greater than 33dB. According to simulation results, high FEC provides the best throughput for a *CIR* less than the 33dB. Therefore, the algorithm switches to high FEC for a *CIR* less than 33dB. The value ε is a sensitivity parameter to prevent the algorithm from switching states at a rapid rate when the *CIR* stays near 33dB.

6.4.3. Algorithm for Minimizing End-to-End Delay

The algorithm to minimize end-to-end delay is as follows.

```
while ( wireless link is in operation )
{
        Wait for bit error rate from the broadband sounder
        if ( CIR > 43dB + ε )
                {
                Enable ARQ scheme
                Set FEC encoding to no FEC
                }
        else if (CIR < 43dB– ε and CIR > 33dB + ε )
                {
                Disable ARQ scheme
                Set FEC encoding to no FEC
                }
        else if ( CIR < 33dB – ε )
                {
                Enable ARQ scheme
                Set FEC encoding to high FEC
                }
}
```

The algorithm is designed to minimize end-to-end delay, however, throughput and packet loss may be adversely affected. Simulation results showed that for a *CIR* greater than 44dB, no FEC encoding with ARQ provides the lowest end-to-end delay. For a *CIR* between 33dB and 44dB, no FEC encoding without ARQ provides the lowest end-to-end delay. High FEC encoding with ARQ provides the lowest end-to-end delay for a *CIR* less than 33dB. The value ε is a sensitivity parameter to prevent the algorithm from switching states at a rapid rate when the *CIR* stays near 33dB and 44dB.

6.5. Summary

This chapter presented results from the simulation model described in Chapter 5. The simulation model used the data link layer protocol described in Chapter 4. The results showed that link layer ARQ improves TCP/IP throughput more than changing the level of FEC. An explanation of why current metrics produced from the broadband sounder data are not useful was given. A new metric, carrier-to-interference ratio, to be calculated from the broadband sounder metrics was proposed. Algorithms using CIR were proposed to improve TCP/IP performance over wireless networks.

Chapter 7. Summary and Conclusions

This chapter provides a summary of the research and suggests future work related to this research.

7.1. Summary

We proposed an adaptive data link layer protocol that uses adaptive forward error correction and an automatic retransmission request scheme at the link layer. These schemes are coupled with a broadband sounder developed by other Virginia Tech researchers. We assumed that the broadband sounder could provide information that is useful in determining a desirable configuration of the data link layer protocol. We developed an OPNET model of the data link layer protocol. TCP/IP behavior with different data link layer properties and channel conditions was estimated and compared. Based on simulation, it was determined that the sounder could not be used to improve the throughput of the data link layer protocol using the initial metrics calculated from data provided by the broadband channel sounder. We proposed a new metric, carrier-to-interference ratio, to be used to aid the adaptive data link layer protocol. Algorithms using bit error rate and carrier-to-interference ratio are proposed to improve both throughput and end-to-end delay for TCP/IP traffic.

7.2. Conclusions

Switching to higher levels of FEC encoding improves TCP/IP throughput for high bit error rates, but increases end-to-end delay of TCP/IP segments. Overall TCP/IP connections with link layer ARQ showed approximately 150 Kbps greater throughput, but exhibited the highest end-to-end delay for high bit error rate channels. We determined that the sounder, coupled with the carrier-to-interference ratio metric, can be used to control and improve both end-to-end delay and throughput for TCP/IP traffic in the network.

We propose algorithms using the current bit error rate of the channel to control end-to-end delay and throughput on the network. We propose a new metric, carrier-to-interference ratio to be calculated from the data gathered from the sounder. We propose algorithms that use the new metric to control end-to-end delay and throughput on the network.

We developed a simulation model of the broadband network being developed by Virginia Tech researchers that could be used later to model changes in the network before the real system is completed. Also, we developed modem controller console software package that can be used to analyze data from the sounder and control data link layer protocol settings.

7.3. Future Work

There are two main areas for possible future work. First, the adaptive data link layer protocol and broadband sounder must be tested in a real system. Since the real system was not available when the simulation model was developed, the results from the simulation model could not be completely validated. After Virginia Tech researchers finish developing the LMDS broadband network for the National Science Foundation-funded Digital Government project, the proposed algorithms for improving throughput and end-to-end delays should be tested with the real system to confirm the effects of different data link layer configurations. The proposed algorithms for controlling end-to-end delay will likely need to be adjusted for the real system. The validity of the proposed metric, carrier-to-interference ratio, for the proposed algorithms needs to be tested against algorithms using the bit error rate of the channel.

The second area for research is network quality of service. In emergency situations certain applications may be considered more mission critical than others. The adaptive features of the data link layer protocol could be used to group different traffic flows to separate mission critical application traffic from application traffic that is not mission critical. Different levels of forward error correction and link layer ARQ could be used to provide mission critical applications with lower packet error rates, lower end-to-end delay and higher throughput than non-mission critical applications. The broadband sounder could be used to determine the data link layer protocol configuration for each traffic flow based on the current conditions of the wireless channel. The simulation model developed for this thesis could be used to test implementations of different quality of service algorithms and determine efficient ways of classifying traffic on the network.

References

[1] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin, "AIRMAIL: A link-layer protocol for wireless networks," *ACM/Baltzer Wireless Networks Journal*, vol. 1, no. ??, pp. 47-60, Feb. 1995.

[2] A. Bakre, B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," *International Conf. on Distributed Computing Systems*, 1995, pp. 136-43.

[3] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," *International Conf. on Mobile Computing and Networking*, 1995, pp. 2-11.

[4] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *ACM. Computer Communication Review*, vol. 26, no. 4, pp. 256-69, Oct. 1996.

[5] Beaulieu NC, Abu-Dayya AA. "Bandwidth efficient QPSK in cochannel interference and fading," IEEE Transactions on Communications, vol.43, no.9, Sept. 1995, pp.2464-74.

[6] C. F. Zukerman and M. Gitlits, "Adaptive transmission parameters optimisation in wireless multi-access communication," *IEEE International Conf. on Networks*, 1999, pp. 91-5.

[7] D. A. Eckhardt and P. Steenkiste, "A trace-based evaluation of adaptive error correction for a wireless local area network," *Mobile Networks and Applications*, vol. 4, no. 4, pp. 273-87, 1999.

[8] N. S. Ericsson, "Adaptive modulation and scheduling of IP traffic over fading channels," *IEEE Vehicular Technology Conference,* vol. 2, pp.849-53, 1999.

[9] Gallagher, Timothy M. Personal Communications. January, 28 2002. January, 29 2002. March 25, 2002. March 26 2002.

[10] V. Jacobson, R. T. Braden, and D. A. Borman, "TCP Extensions for High Perfromance," Internet Engineering Task Force Request for Comments 1323, May 1992.

[11] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, JohnWiley and Sons, 1991.

[12] Kocaturk M, Gupta SC, Arslan H. "Bit-error rate of TDMA links under cochannel interference in overlaid cellular networks." IEEE Transactions on Vehicular Technology, vol.47, no.3, Aug. 1998, pp.808-18.

[13] B. E. Mullins, "CATER: An opportunistic medium access control protocol for wireless local area networks," Ph.D. Dissertation, Virginia Polytechnic Institute and State University, June 1997.

[14] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP performance over wireless networks at the link layer," *Mobile Networks and Applications*, vol. 5, no. 1, pp. 57–71, March 2000.

[15] L. L. Peterson and Bruce S. Davie, *Computer Networks: A Systems Approach*, 2nd edition, Morgan Kaufman Publishers, San Francisco, CA, 2000, pp. 311-219.

[16] Rappaport, Theodore S., *Wireless Communications: Principles & Practice*, Prentice Hall, Inc., Upper Saddle River, New Jersey, 1996. pp. 172-173, 244,346-360.

[17] Rieser, Christian. *Design and Implementation of a Sampling Swept Time Delay Short Pulse (SSTDSP) Wireless Channel Sounder for LMDS.* Thesis. Virgnia Tech. July 17, 2001.

[18] N. Smavatkul and S. F. Midkiff, "Impact of high-end radio functionality in wireless mobile ad hoc networks," *Multiaccess, Mobility, and Teletraffic for Wireless Communications*, 2000, pp. 195-206.

[19] G. Xylomenos and G. C. Polyzos, "TCP and UDP performance over a wireless LAN," *IEEE INFOCOM*, vol. 2, 1999, pp.439-46.

Appendix A: Documentation for the Modem Controller Console Software

This appendix contains the class diagrams and descriptions for the modem controller software that was developed to gather data from the broadband sounder and control the adaptive scheme of the data link layer protocol.

A.1. Class Diagrams

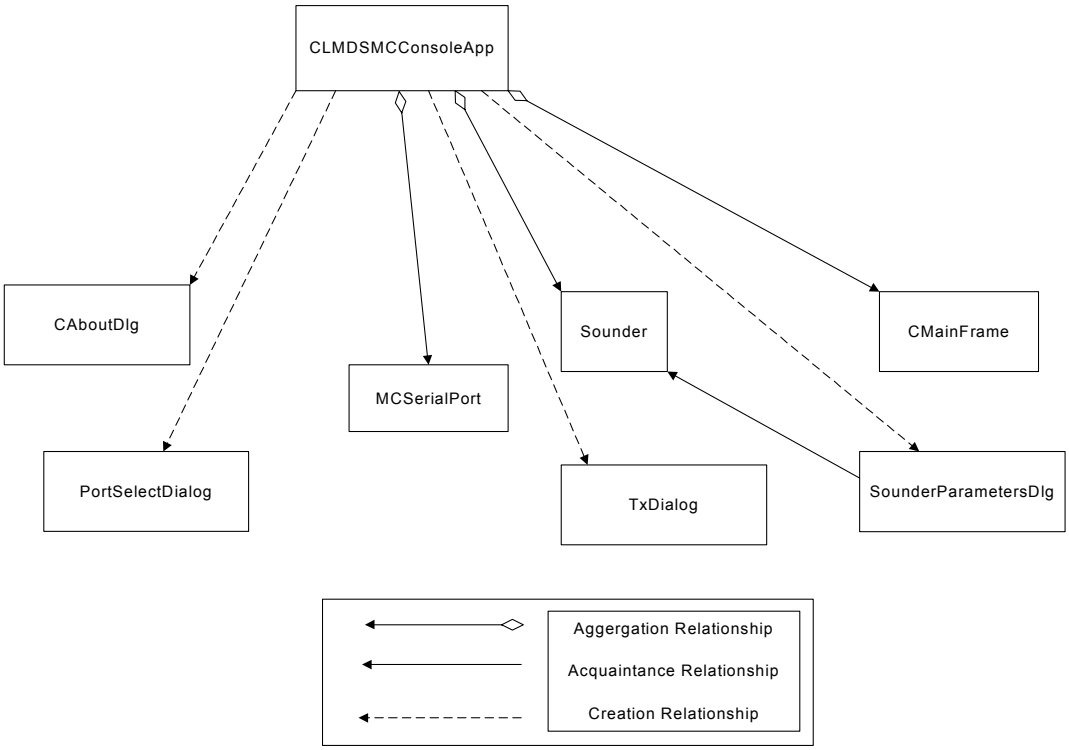

Figure A.1. Class Diagram for Relationships between the Main Application Class and Utility Classes.

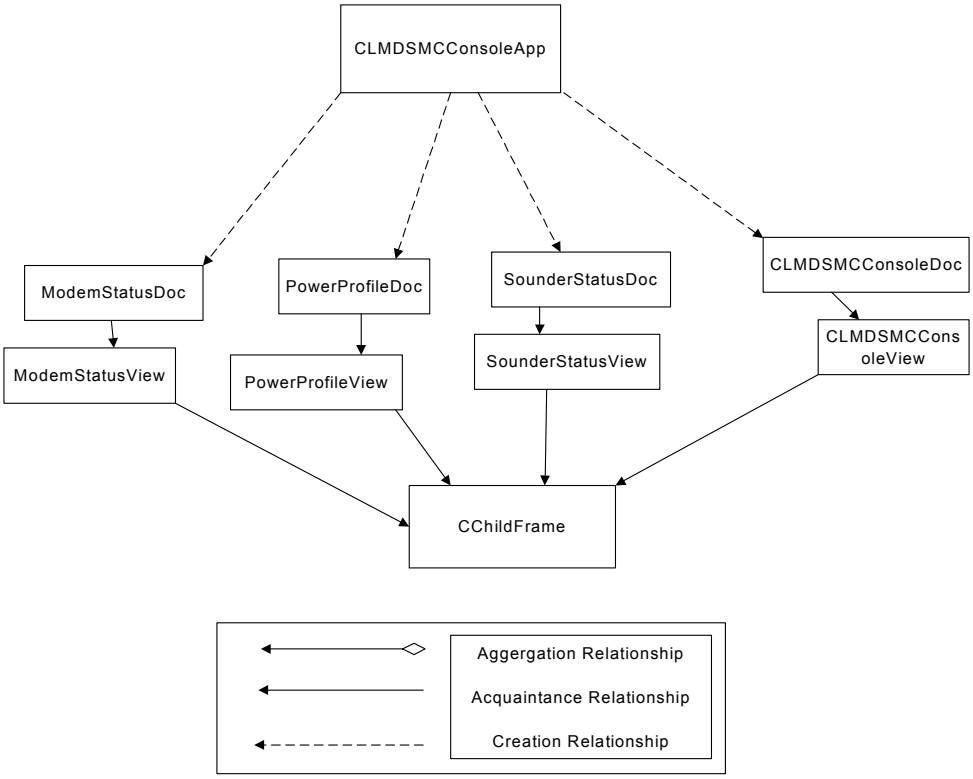

Figure A.2. Class Diagram for Relationships between the Main Application Class and Document/View Classes.

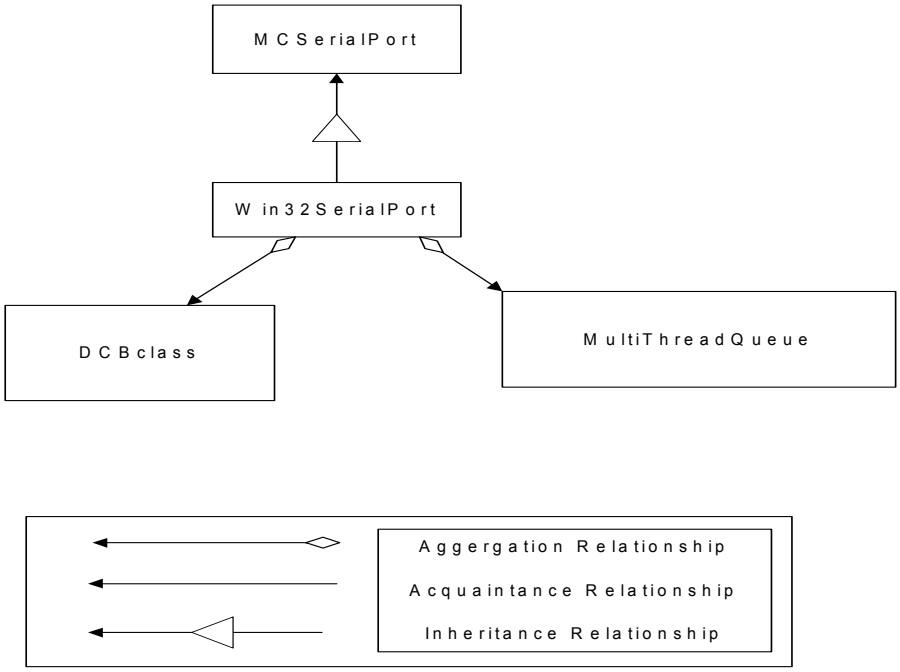

Figure A.3. Class Diagram for Serial Port and Supporting Classes.

A.2. Class Descriptions

A.2.1. CLMDSMCConsoleApp Class

This section contains an explanation of the interface for the CLMDSMCConsoleApp class. This class is the application class for the LMDS Console application. It launches the main thread that controls the application. It encapsulates or uses all the classes defined with the application.

A.2.1.1. Private Data Members

Type and Name	Represents
double ThroughputSwitchPoint	The transition point for the throughput control algorithm in dBc. This value is used to switch between high encoding and low encoding
bool ThroughputControl	Boolean value that determines if the Throughput control algorithm is on or off.
bool DelayControl	Boolean value that determines if the Delay control algorithm is on or off.
double DelayLowSwitchPoint	The low transition point for the delay control algorithm in dBc. This value is used to switch between high encoding with ARQ and no FEC without ARQ
double DelayHighSwitchPoint	The high transition point for the delay control in dBc. This value is used to switch between no FEC with ARQ and no FEC without ARQ.
double Tolerance	Tolerance for delay and throughput control algorithms in dBc. This value is used to prevent the algorithms from switching states too quickly.
double LevelOneHighMark	The transition point for the adaptive data link layer protocol to switch from no FEC to low FEC encoding.
double LevelOneLowMark	The transition point for the adaptive data link layer protocol to switch from low FEC to no FEC encoding.
double LevelTwoHighMark	The transition point for the adaptive data link layer protocol to switch from low FEC to medium FEC encoding.
double LevelTwoLowMark	The transition point for the adaptive data link layer protocol switch from medium FEC to low FEC encoding.
double LevelThreeHighMark	The transition point for the adaptive data link layer protocol switch from medium FEC to high FEC encoding.
double LevelThreeLowMark	The transition point for the adaptive data link layer protocol switch from high FEC to medium FEC encoding.
CMutex SounderStatusMutex	Mutex object used to synchronize the application with the sounder object when the sounder has sounder status information.
CDocument * pConsoleDoc	Pointer to the LMDS Console document. This document presents a transmission log between the Console software and the modem controller
CMultiDocTemplate* pConsoleDocTemplate;	Pointer to the LMDS Console document template for the LMDS modem console. This template contains the view and document classes that control the modem console transmission log.

CDocument * pModemStatusDoc;	Pointer to the modem status document. This document presents current settings in the modem controller
CMultiDocTemplate* pModemStatusDocTemplate;	Pointer to the modem status document template. This template contains the view and document class that control the modem status window
CDocument *pSounderStatusDoc;	Pointer to the sounder status document. This document presents current readings from the sounder.
CMultiDocTemplate* pSounderStatusTemplate;	Pointer to the sounder status template. This template contains the sounder status document and view classes that present the current sounder reading.
CDocument* pPowerProfileDoc;	Pointer to the power profile document. This document presents a graph of the power profile given to it by the sounder.
CMultiDocTemplate* pPowerProfileTemplate;	Pointer to the power profile document template. This template helps control the document and view classes that present the graph of the power delay profile.
CMainFrame* pMainFrame;	Pointer to the main window of the application
TxDialog TxDlg;	Dialog object that present an interface for the user to enter commands to send to the modem controller
SounderParametersDlg SndParDlg	Dialog object that presents an interface for the user to change the settings of the sounder parameters.
MCSerialPort Port;	Serial Port the encapsulates the serial port that connects the computer to the modem controller.
Sounder* pBBSounder;	Pointer to the sounder object that controls the interaction between the software and sounder hardware.

A.2.1.2. Public Method Members

Name	CLMDSMCConsoleApp()
Receives	nothing
Returns	nothing
Purpose	Constructs and initializes an CLMDSMCConsole App object.

Name	int CLMDSMCConsoleApp::ExitInstance()
Receives	nothing
Returns	int //exit code of the main thread when the application exits.
Purpose	The function is called when the application exits. It shuts down any threads in the application and closes the serial port.

Name	BOOL CLMDSMCConsoleApp::InitInstance()
Receives	nothing
Returns	BOOL // FALSE if the application fails to start for any reason.
Purpose	This function initializes the application at startup. It reads the configuration file, opens the serial port, and configures the sounder object.

Name	void CLMDSMCConsoleApp::OnAppAbout()
Receives	nothing
Returns	nothing
Purpose	This function is called by the MFC message map to open the “about” dialog box. It is called when a user selects Help->About

Name	LONG CLMDSMCConsoleApp::OnBBSNotify(UINT wParam,LONG lParam)
Receives	UINT wParam, //message ID for a broad band sounder update LONG lParam // contains nothing, placed here to match standard for custom MFC messages handlers
Returns	LONG //serves no purpose. Kept to maintain standard for custom MFC message handlers
Purpose	Traps the message sent out by the sounder object when the sounder has new readings from hardware and calls the UpdateSounderStatus() function.

Name	void CLMDSMCConsoleApp::OnFileOpen()
Receives	nothing
Returns	nothing
Purpose	This function opens the modem command transmission log. It ensures the serial port is opened and that only one transmission command dialog box is open.

Name	void CLMDSMCConsoleApp::OnModemStatusMenuItem()
Receives	nothing
Returns	nothing
Purpose	This function opens the modem status window and ensures that only one status window is open.

Name	void CLMDSMCConsoleApp::OnParameters()
Receives	nothing
Returns	nothing
Purpose	This function opens the sounder parameters dialog box. The dialog allows the user to change sounder parameters.

Name	void CLMDSMCConsoleApp::OnPowerProfile()
Receives	nothing
Returns	nothing
Purpose	This function opens the window that presents a graph of the power delay profile. It ensures that only one power delay profile graph window is open.

Name	LONG CLMDSMCConsoleApp::OnRxNotify(UINT wParam,LONG lParam)
Receives	UNIT wParam //number of bytes received LONG lParam //not used by OnRxNotify, however, required for custom MFC messages
Returns	LONG //kept to maintain standard for custom MFC messages
Purpose	Displays message received over serial port from the modem controller onto the modem transmission log window.

Name	void CLMDSMCConsoleApp::OnSerialPortClose()
Receives	nothing
Returns	nothing
Purpose	Closes the serial port that is connected to the modem controller

Name	void CLMDSMCConsoleApp::OnSerialPortOpen()
Receives	nothing
Returns	nothing
Purpose	Opens the serial port that is connected to the modem controller

Name	void CLMDSMCConsoleApp::OnSerialPortProperties()
Receives	nothing
Returns	nothing
Purpose	Opens a dialog box that allows the serial port properties to be changed such as baud rate and parity.

Name	void CLMDSMCConsoleApp::OnSounder()
Receives	nothing
Returns	nothing
Purpose	This function turns the sounder on and off. When it is “on”, threads are launched and the sounder begins to collect data. When it is “off”, the sounder threads are shut down and the sounder stops collecting data.

Name	void CLMDSMCConsoleApp::OnSounderStatusMenuItem()
Receives	nothing
Returns	nothing
Purpose	This function opens the sounder status window that presents data readings from the sounder. It ensures that only one sounder status window is open

Name	void CLMDSMCConsoleApp::OnTxOpen()
Receives	nothing
Returns	nothing
Purpose	This function opens the transmission command dialog. This dialog is a modular dialog box that can be open while the user performs other functions.

Name	void CLMDSMCConsoleApp::OnTxSend()
Receives	nothing
Returns	nothing
Purpose	This function is called by the transmission command dialog box and grabs the text message entered by the user. It sends the message to the serial port object so it can be sent to the modem controller.

Name	void CLMDSMCConsoleApp::OnUpdateFileOpen(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disables and enables the File->Open menu item. If the modem transmission command log window is open, the menu item is disabled

Name	void CLMDSMCConsoleApp::OnUpdateModemStatusMenuItem(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disable and enables the View->Modem Status menu item. If the modem status window is open, the menu item is disabled.

Name	void CLMDSMCConsoleApp::OnUpdatePowerProfile(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disables and enables the View->Channel Power Profile menu item. If the channel power profile window is already open, the menu item is disabled.

Name	void CLMDSMCConsoleApp::OnUpdateSerialPortClose(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disables and enables the Serial Port->Close Port menu item. If the port is already closed, the menu item is disabled.

Name	void CLMDSMCConsoleApp::OnUpdateSerialPortOpen(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disables and enables the SerialPort->Open Port menu item. If the port is already open, the menu item is disabled.

Name	void CLMDSMCConsoleApp::OnUpdateSerialPortProperties(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disables and enables the Serial Port->Properties menu. If the port is open the menu item is enabled.

Name	void CLMDSMCConsoleApp::OnUpdateSounder(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function sets the text for the menu item Sounder->Sounder. If the sounder is off, the text is “Enable Sounder.” If the sounder is on, the text is “Disable Sounder”.

Name	void CLMDSMCConsoleApp::OnUpdateSounderStatusMenuItem(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disables and enables the View->Sounder Status menu item. If the sounder status window is open, the item is disabled.

Name	void CLMDSMCConsoleApp::OnUpdateTxOpen(CCmdUI* pCmdUI)
Receives	CCmdUI* pCmdUI //pointer to the user interface command object
Returns	nothing
Purpose	This function disables and enables the View->Transmit Command Dialog menu item. If the dialog box is already visible, the menu item is disabled.

Name	void CLMDSMCConsoleApp::UpdateSounderStatus()
Receives	nothing
Returns	nothing
Purpose	This function uses a mutex to synchronize itself with the sounder object. It then gets the latest readings from the sounder. It updates the sounder status window and power profile window if they are open.

Name	void CLMDSMCConsoleApp::OnUpdateThroughput(CCmdUI* pCmdUI)
Receives	CcmdUI* pCmdUI
Returns	Nothing
Purpose	This function controls the throughput control algorithm menu item under the sounder menu

Name	void CLMDSMCConsoleApp::OnDelay()
Receives	Nothing
Returns	Nothing
Purpose	This function disables and enables the delay control algorithm

Name	void CLMDSMCConsoleApp::OnThroughput()
Receives	Nothing
Returns	Nothing
Purpose	This function disables and enables the throughput control algorithm

Name	void CLMDSMCConsoleApp::OnUpdateDelay(CCmdUI* pCmdUI)
Receives	CcmdUI* pCmdUI //pointer to user interface control
Returns	Nothing
Purpose	This function controls the delay control algorithm control menu item under the sounder menu item.

Name	void CLMDSMCConsoleApp::ThroughputControlAlgorithm()
Receives	Nothing
Returns	Nothing
Purpose	This function contains the throughput control algorithm that updates every time the broadband channel sounder sounds.

Name	void CLMDSMCConsoleApp::DelayControlAlgorithm()
Receives	Nothing
Returns	Nothing
Purpose	This function contains the end-to-end delay control algorithm that updates every time the broadband channel sounder sounds

A.2.1.3. Private Methods

Name	void CLMDSMCConsoleApp::KillSounderThreads(BOOL wait)
Receives	BOOL wait //if the BOOL ==TRUE the function waits to ensure threads have exited instead of just trying to make the threads exit
Returns	nothing
Purpose	To shutdown the threads within the sounder object.

Name	void CLMDSMCConsoleApp::LaunchSounderThreads()
Receives	nothing
Returns	nothing
Purpose	Launches the threads for the sounder. Launching threads causes the sounder to begin data collection

Name	bool CLMDSMCConsoleApp::LoadConfigFile()
Receives	nothing
Returns	bool //returns TRUE if the configuration file is successfully loaded
Purpose	This function loads the configuration file for the application. The configuration file contains the parameters and settings for the modem controller and sounder

Name	void CLMDSMCConsoleApp::SendCommand(CString message)
Receives	CString message //message to be sent to the modem controller
Returns	nothing
Purpose	This function sends a message entered from the send command dialog box to the serial port object. The serial port object then sends it to the modem controller.

A.2.2. CAboutDlg Class

This section contains an explanation of the interface for the CAboutDlg Class. The CAboutDlg Class creates an "About" Dialog box. The dialog box presents the title of the program and the name of the author who wrote the program.

A.2.2.1 Public Method Members

Name	CAboutDlg()
Receives	Nothing
Returns	Nothing
Purpose	Creates a CAboutDlg object. Upon calling this function, the "about" dialog box is created.

Name	DoDataExchange(CDataExchange* pDX)
Receives	CDataExchange *pDX //pointer to a CDataExchange object
Returns	nothing
Purpose	This function performs any data exchange operations between the screen and the dialog object. However, this function does provide any useful operation for this class. It is only provide for completeness.

A.2.3. CChildFrame Class

This file contains an explanation of the interface for the CChildFrame class. This class was created by C++ Visual Studio as part of the document/view architecture of MFC. The class was not modified from its original implementation.

A.2.3.1. Public Method Members

Name	void CChildFrame::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	Validates the CChildFrame object. This function was created by C++ Visual Studio and was not modified.

Name	CChildFrame::CChildFrame()
Receives	Nothing
Returns	Nothing
Purpose	Default constructor for the CChildFrame class.

Name	CChildFrame::~CChildFrame()
Receives	Nothing
Returns	Nothing
Purpose	Default destructor for the CChildFrame class.

Name	void CChildFrame::Dump(CDumpContext& dc) const
Receives	CDumpContext &dc // dump context object
Returns	Nothing
Purpose	This function is required by the document/view architecture and was created by C++ Visual Studio.

Name	BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
Receives	CREATESTRUCT & cs //create structure
Returns	BOOL //true if the initialization for the window is successful.
Purpose	This function is called before a CChildFrame window is created. This function was created by C++ Visual Studio and was not modified.

A.2.3.2. Protected Method Members

Name	int CChildFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
Receives	LPCREATESTRUCT lpCreateStruct //pointer to create structure
Returns	int
Purpose	This function is required by the document/view architecture and was created by C++ Visual Studio.

A.2.4. CLMDSMCConsoleDoc Class

This section contains an explanation of the interface for the CLMDSMCConsoleDoc Class. The CLMDSMCConsoleDoc Class is the document for the document/view MFC architecture. It is used to help control the command transmission log window.

A.2.4.1. Public Method Members

Name	void CLMDSMCConsoleDoc::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function calls CDocument::AssertValid(); function. The default implementation provided by MFC Visual Studio was used for this function

Name	CLMDSMCConsoleDoc::~CLMDSMCConsoleDoc()
Receives	nothing
Returns	nothing
Purpose	Destroys a CLMDSMCConsoleDoc(). The default implementation provided by MFC Visual Studio was used for this function.

Name	void CLMDSMCConsoleDoc::Dump(CDumpContext& dc) const
Receives	CDumpContext &dc //device context for the document object

Returns	nothing
Purpose	Calls CDocument::Dump(CDumpContext& dc). The default implementation provided by MFC Visual Studio was used for this function.

Name	BOOL CLMDSMCConsoleDoc::OnNewDocument()
Receives	nothing
Returns	BOOL //TRUE if the object is initialized correctly
Purpose	This function initializes the CLMDSMCConsoleDoc object. The default implementation provided by MFC Visual Studio was used for this function

Name	void CLMDSMCConsoleDoc::Serialize(CArchive& ar)
Receives	CArchive& ar //archive object that the document object will be archived into
Returns	nothing
Purpose	This function was supplied by the MFC Visual Studio and was not used in this program.

A.2.5. CLMDSMCConsoleView Class

This section contains an explanation of the interface for the CLMDSMCConsoleView class. The CLMDSMCConsoleView Class presents the transmission log for modem commands on the string. This class publicly inherits the CEditView class.

A.2.5.1. Public Method Members

Name	void CLMDSMCConsoleView::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	void CLMDSMCConsoleView::Dump(CDumpContext& dc) const
Receives	CDumpContext & dc //device context object
Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	CLMDSMCConsoleDoc* CLMDSMCConsoleView::GetDocument()
Receives	nothing
Returns	CLMDSMCConsoleDoc* //pointer to CLMDSCConsoleDoc object currently associated with the view object
Purpose	To return a pointer to the document object attached to the view document.

Name	CLMDSMCConsoleView::~CLMDSMCConsoleView()
Receives	nothing
Returns	nothing
Purpose	Default destructor for the CLMDSMCConsoleView class. This function was created by C++ Visual Studio.

Name	void CLMDSMCConsoleView::OnDraw(CDC* pDC)
Receives	CDC* pDC //device context
Returns	nothing
Purpose	This function is called when the power profile is displayed or printed. It uses the data points collected by the sounder and draws them on the device context.

A.2.5.2. Protected Method Members

Name	CLMDSMCConsoleView::CLMDSMCConsoleView()
Receives	nothing
Returns	nothing
Purpose	Default constructor for the CLMDSMCConsoleView class. This function was created by C++ Visual Studio.

Name	BOOL CLMDSMCConsoleView::OnPreparePrinting(CPrintInfo* pInfo)
Receives	CPrintinfo* pInfo //pointer to print information object. This object contains any user selected printing information
Returns	nothing
Purpose	This function prepares the view object for printing.

Name	int CLMDSMCConsoleView::OnCreate(LPCREATESTRUCT lpCreateStruct)
Receives	LPCREATESRUCT lpCreateStruct
Returns	int
Purpose	This function is called when the view object is created. This function was supplied by C++ Visual Studio.

Name	void CLMDSMCConsoleView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
Receives	CDC*pDC //pointer to printer device context CPrintInfo* pInfo // pointer to printing information object.
Returns	nothing
Purpose	This function performs any tasks needed at the end of a print job. This function currently overrides the default CEditView::OnEndPrinting(pDC, pInfo);

Name	BOOL CLMDSMCConsoleView::OnPreparePrinting(CPrintInfo* pInfo)
Receives	CPrintInfo *pInfo
Returns	BOOL
Purpose	This function prepares the CLMDSMCConsoleView object for printing. This function currently overrides the default CEditView::OnPreparePrinting(pInfo);

A.2.6. CMainFrame Class

This section contains an explanation of the interface for the CMainFrame class. This class controls the main window for the application. This class inherits from CMDIFrameWnd.

A.2.6.2. Private Data Members

Type and Name	Represents
CWinApp * App;	Pointer to the application object that created the CMainFrame object

A.2.6.3. Protected Data Members

Type and Name	Represents
CStatusBar m_wndStatusBar	Object that controls the main windows status bar at the bottom of the window. This object was added by Visual C++ Studio
CToolBar m_wndToolBar	Object that controls the tool bar for the main window. This Object was added by Visual C++ Studio.

A.2.6.4. Public Method Members

Name	void CMainFrame::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function only performs the default implementation in CMDIFrameWnd::AssertValid() and was created by Visual C++ Studio

Name	void CMainFrame::AttachApp(CWinApp *app)
Receives	CWinApp * app //pointer to the CWinApp that created this CMainFrame Class
Returns	nothing
Purpose	To attach the CMainframe object to the application object that created.

Name	void CMainFrame::OnTxSend()
Receives	nothing
Returns	nothing
Purpose	This function calls OnTxSend from the attached application object when the transmit command dialog has a command ready to be sent.

Name	LRESULT CMainFrame::OnBBSNotify(UINT wParam, LONG lParam)
Receives	UINT wParam LONG lParam //these two arguments are required for custom MFC messages and are not used by OnBBSNotify
Returns	LRESULT //not used, however LRESULT is a required return for custom MFC messages
Purpose	To call BBSNotify() from the attached application object when the sounder object has finished collecting data from the sounder.

Name	LRESULT CMainFrame::OnRxNotify(UINT wParam, LONG lParam)
Receives	UINT wParam //number of bytes received on the port. LONG lParam //this argument is not used by OnRxNotify, however, it is required for custom MFC messages
Returns	LRESULT //not used, however, LRESULT is a required return value for custom MFC applications.
Purpose	This function calls OnRxNotify() from the attached application object when the data has arrived from the modem controller.

Name	BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
Receives	CREATESTRUCt &cs
Returns	BOOL

Purpose	This function performs any necessary tasks before the window for the CMainFrame object is created. The default implementation provided by Visual C++ Studio was used for the function.

A.2.6.5. Protected Method Members

Name	int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
Receives	LPCREATESTRUCt lpCreateStruct
Returns	int
Purpose	This function is called when the CMainFrame object is created. The default implementation provided by Visual C++ was used for this function.

A.2.7. CommandParser Class

This file contains an explanation of the interface for the CommandParser Class. The CommandParser object is attached to a ModemStatusView object. This object parses any commands received over the serial port. It then displays then on the screen and displays them on the attached ModemStatusView object. See Appendix B for the Modem Command Reference.

A.2.7.1. Private Data Members

Type and Name	Represents
strstream command_stream	string stream that contains the string passed to it by ParseCommand() function.
ModemStatusView * pStatusView	pointer to the ModemStatusView object

A.2.7.2. Public Method Members

Name	void CommandParser::AttachToView(ModemStatusView *view)
Receives	ModemStatusView* view //pointer to the ModemStatusView object.
Returns	nothing
Purpose	This function attaches CommandParser to the ModemStatusView object currently in use by the application.

Name	CommandParser::CommandParser()
Receives	nothing
Returns	nothing
Purpose	Constructs a CommandParser object and sets pStatusView member to NULL.

Name	CommandParser::~CommandParser()
Receives	nothing
Returns	nothing
Purpose	Default destructor for the CommandParser. Provided for completion. It does not perform any tasks.

Name	bool CommandParser::ParseCommand(const char *command_string)
Receives	const char* command_string
Returns	bool //true if the command was a properly formatted command

Purpose	This function parses the command pointed to by command_string. If the command has a formatting error, false is returned. Otherwise, true is returned.

A.2.7.3. Private Method Members

Name	bool CommandParser::ParseARQ()
Receives	nothing
Returns	bool //true if a valid ARQ command was received.
Purpose	This function parses command_stream object for ARQ command. If an ARQ command is valid, true is returned.

Name	bool CommandParser::ParseFECLevel()
Receives	nothing
Returns	bool //true, if the FEC Level command is valid
Purpose	This function parses the command_stream object for an FEC level command. If the FEC Level command is valid, true is returned.

Name	bool CommandParser::ParseNumberOfRemotes()
Receives	nothing
Returns	bool //true if the command was parsed correctly
Purpose	This function parses the Number Of Remotes command and determines the number of remotes in the system.

Name	bool CommandParser::ParseRemoteID()
Receives	nothing
Returns	bool // true if the command was parsed correctly
Purpose	This function parses the RemoteID command and determines the ID of the modem controller.

Name	bool CommandParser::ParseStatus()
Receives	nothing
Returns	bool // true if the command was parsed correctly
Purpose	This function parses a Status command received from the modem. The function returns true, if the status command is valid.

Name	bool CommandParser::ParseTDMAMode()
Receives	nothing
Returns	bool //true if the TDMA Mode command was parsed correctly
Purpose	This function parses a TDMA Mode command from the modem. The function returns true, if the TDMA Mode is valid.

Name	bool CommandParser::ParseTransmit()
Receives	nothing
Returns	bool //true if the command was parsed correctly
Purpose	This function parses a Transmit Mode command from the modem. The function returns true, if the Transmit Mode is valid.

A.2.8. DCBclass Class

This section contains an explanation of the interface for the DCBclass class. This class is a wrapper for the DCB structure used for Win32 Serial IO API. This class simplifies the uses of this structure by providing member functions to update the DCB structure data members.

A.2.8.1. Private Data Members

Type and Name	Represents
DCB dcb_struct	DCB structure. This holds all the settings for a serial port.

A.2.8.2. Public Method Members

Name	LPDCB DCBclass::CopyFrom(LPDCB otherDCB)
Receives	LPDCB otherDCB //DCB structure to copy
Returns	LPDCB //pointer to the DCB structure within the object.
Purpose	To copy the settings from one DCB structure to this object.

Name	LPDCB DCBclass::CopyFrom(LPDCB otherDCB)
Receives	LPDCB otherDCB //DCB structure to copy
Returns	LPDCB //pointer to the DCB structure within the object.
Purpose	To copy settings from DCB class to the given DCB structure

Name	DCBclass::DCBclass(DCBclass &Rhs)
Receives	DCBclass &Rhs
Returns	nothing
Purpose	Copy constructor to allow the class to be passed as an argument for function correctly.

Name	DCBclass::DCBclass()
Receives	Nothing
Returns	Nothing
Purpose	To create and initialize a DCB data structure

Name	DCBclass::~DCBclass()
Receives	Nothing
Returns	Nothing
Purpose	Default destructor for the DCBclass

Name	LPDCB DCBclass::operator = (DCB &Rhs)
Receives	DCB &Rhs
Returns	LPDCB //pointer to the DCB structure in the class
Purpose	To allow the DCBclass to use the assignment operator with DCB structures. It copies the settings in Rhs to the DCBclass.

Name	DCBclass::operator LPDCB()
Receives	nothing

Returns	LPDCB pointer to the DCB data structure
Purpose	returns the pointer to the DCB data structure, so the DCBclass can be used with SetCommState and GetCommState and other Win32 Serial IO API functions

Name	void DCBclass::SetBaudRate(DWORD rate)
Receives	DWORD rate //integer value of the baud rate if the value SAME is passed the baud rate will not be changed
Returns	nothing
Purpose	To change the baud rate setting in the DCB structure

Name	void DCBclass::SetDtr(DWORD value)
Receives	DWORD value Raises or lowers the DTR line. It can have the following values SET raise the DTR line UNSET lower the DTR line SAME leaves the DTR line in the previous sate
Returns	nothing
Purpose	To raise or lower the DTR line

Name	void DCBclass::SetDtrDsr(DWORD value)
Receives	DWORD value This value enables or disable DTR/DSR handshaking ENABLE handshaking on DISABLE handshaking off SAME leaves DTR/DSR handshaking unchanged
Returns	Nothing
Purpose	To turn DTR/DSR handshaking on or off

Name	void DCBclass::SetParity(BYTE parity)
Receives	BYTE parity Amount of parity to be used for the port. The parameter can take on the following values EVENPARITY Even MARKPARITY Mark NOPARITY No parity ODDPARITY Odd SPACEPARITY Space SAME leaves parity unchanged
Returns	Nothing
Purpose	To change or set the parity value in the DCBclass.

Name	void DCBclass::SetRts(DWORD value)
Receives	DWORD value Raises or lowers the RTS line. It can have the following values SET raise the DTR line UNSET lower the DTR line SAME leaves the state of the RTS line unchanged
Returns	Nothing
Purpose	To raise or lower the RTS line.

Name	void DCBclass::SetRtsCts(DWORD value)
Receives	DWORD value This value enables or disable RTS/CTS handshaking ENABLE handshaking on DISABLE handshaking off SAME leaves RTS/CTS handshaking unchanged
Returns	Nothing

Purpose	To enable or disable RTS/CTS handshaking.

Name	void DCBclass::SetStopBits(BYTE stop_bits)
Receives	BYTE stop_bits number of stop and start bits. The following values can be used. ONESTOPBIT 1 stop bit ONE5STOPBITS 1.5 stop bits TWOSTOPBITS 2 stop bits SAME leaves the number of stopbits unchanged
Returns	Nothing
Purpose	To change the number of stop bits being used

Name	void DCBclass::SetWordLength(BYTE word_length)
Receives	BYTE word_length word length to be sent across the link, 5,7, or 8 for example SAME leaves the word size unchanged
Returns	Nothing
Purpose	To set the size of the data word

Name	void DCBclass::SetXonXoff(DWORD value)
Receives	DWORD value This value enables or disable XON/XOFF handshaking ENABLE handshaking on DISABLE handshaking off SAME leaves XON/XOFF handshaking unchanged
Returns	nothing
Purpose	To enable or disable XON/XOFF handshaking

A.2.9. MCSerialPort Class

This section contains an explanation of the interface for the MCSerialPort class. This class inherits all the behavior of the Win32SerialPort class in Win32SerialPort.lib.

A.2.9.1. Private Data Members

Type and Name	Represents
HWND m_MainWnd;	Handle for the main window that serial port is being used by. This is needed for messaging to the window frame.

A.2.9.2. Public Method Members

Name	void MCSerialPort::AttachToWindow(HWND hWnd)
Receives	HWND hWnd //handle to the window needed for messaging
Returns	nothing
Purpose	This function attaches the serial port object to the handle of the main window of the application. Once attached, the serial port is able to send and receiver messages to the application.

Name	BOOL MCSerialPort::Close()
Receives	nothing
Returns	BOOL //true if the serial port successfully closes
Purpose	This function calls the Win32SerialPort::Close() function to close the serial port.

Name	MCSerialPort::MCSerialPort()
Receives	nothing
Returns	nothing
Purpose	This is default constructor for the MCSerialPort class and was only implemented for completeness.

Name	MCSerialPort::~MCSerialPort()
Receives	nothing
Returns	nothing
Purpose	This is the default desctructor for the MCSerialPort class and was only implemented for completeness.

A.2.9.3. Protected Method Members

Name	void MCSerialPort::NotifyOfRx(DWORD BytesRecieved)
Receives	DWORD BytesRecieved //number of bytes received over the port
Returns	nothing
Purpose	This function was written to over ride the virtual function NotifyOfRx(). It posts a message to the main window of the application and passes the number of bytes when it posts the message.

A.2.10. ModemStatusDoc Class

This section contains an explanation of the interface for the ModemStatusDoc Class. The ModemStatusDoc Class is the document for the document/view MFC architecture. It is used to help control the sounder status window.

A.2.10.1. Public Method Members

Name	void ModemStatusDoc::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function calls CDocument::AssertValid(); function. The default implementation provided by MFC Visual Studio was used for this function

Name	ModemStatusDoc::~ModemStatusDoc()
Receives	nothing
Returns	nothing
Purpose	Destroys a ModemStatusDoc(). The default implementation provided by MFC Visual Studio was used for this function.

Name	void ModemStatusDoc::Dump(CDumpContext& dc) const
Receives	CDumpContext &dc //device context for the document object
Returns	nothing
Purpose	Calls CDocument::Dump(CDumpContext& dc). The default implementation provided by MFC Visual Studio was used for this function.

Name	BOOL ModemStatusDoc::OnNewDocument()

Receives	nothing
Returns	BOOL //TRUE if the object is initialized correctly
Purpose	This function initializes the ModemStatusDoc object. The default implementation provided by MFC Visual Studio was used for this function

Name	void ModemStatusDoc::Serialize(CArchive& ar)
Receives	CArchive& ar //archive object that the document object will be archived into
Returns	nothing
Purpose	This function was supplied by the MFC Visual Studio and was not used in this program.

A.2.11. ModemStatusView Class

This section contains an explanation of the interface for the ModemStatusView class. The ModemStatusView Class presents the current settings of the adaptive data link layer protocol and the modem controller. This class is apart of the document/view architecture of MFC.

A.2.11.1. Public Data Members

Type and Name	Represents
CEdit m_TransmitMode	This CEdit object displays the transmit mode of the modem controller.
CEdit m_TDMAMode	This CEdit object displays the TDMA mode of the modem controller.
CEdit m_RemoteID	This CEdit object displays the ID of the modem controller. Each modem controller uses an ID determine its TDMA slot.
CEdit m_NumberOfRemotes	This CEdit object displays the number of remotes currently in the system.
CEdit m_FECLevel	This CEdit object displays the current FEC code used by the modem controller
CEdit m_RMSDelaySpread;	This CEdit object displays the RMS Delay Spread of the channel
CEdit m_ARQState	This CEdit object displays if the ARQ scheme has been turned on or off in the modem controller.

A.2.11.2. Public Method Members

Name	void ModemStatusView::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	void ModemStatusView::DoDataExchange(CDataExchange* pDX)
Receives	CDataExchange* pDX //pointer to CData Exchange object
Returns	nothing
Purpose	Displays the values contained in the private CEdit objects of this view class on the screen.

Name	void ModemStatusView::Dump(CDumpContext& dc) const

Receives	CDumpContext & dc //device context object
Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	BOOL ModemStatusView::OnPreparePrinting(CPrintInfo* pInfo)
Receives	CPrintInfo* Info
Returns	BOOL //returns result of DoPreparePrinting. If successful, TRUE is returned.
Purpose	To prepare the view for printing. This function was created by C++ Visual Studio and was not used by this program.

Name	ModemStatusView::ModemStatusView()
Receives	nothing
Returns	nothing
Purpose	Default constructor for the ModemStatusView class. This function was created by C++ Visual Studio.

Name	ModemStatusView::~ModemStatusView()
Receives	nothing
Returns	nothing
Purpose	Default destructor for the ModemStatusView class. This function was created by C++ Visual Studio.

A.2.12. MultiThreadQueue Class

This section contains an explanation of the interface for the MultiThreadQueue. This class provides a multithread byte queue. The queue is protected from more than one thread using it at a time by a mutex. This class is useful for a queue of bytes needs to be shared amongst threads. This queue uses traditional FIFO method of storage and removal.

A.2.12.1. Private Data Members

Type and Name	Represents
CMutex m_Mutex	Mutex object controls access to the queue. It insures the only one thread is accessing the queue at a time.
deque<char> m_Queue	STL deque<char>. This queue holds chars that are protected against multiple threads accessing it at a time.
int m_size	Maximum size of the queue.

A.2.12.2. Public Method Members

Name	void MultiThreadQueue::Flush()
Receives	Nothing
Returns	Nothing
Purpose	This method is used to empty the queue

Name	int MultiThreadQueue::FreeSpace()

Receives	nothing
Returns	int //number of bytes that are free in the buffer
Purpose	This member function is used to determine the number of bytes free remaining in the queue.

Name	MultiThreadQueue::MultiThreadQueue()
Receives	nothing
Returns	nothing
Purpose	Creates a MultiThreadQueue Object with the default max size of 500. It also setups the mutex used to protect the queue against simultaneous multiple access.

Name	MultiThreadQueue::~MultiThreadQueue()
Receives	Nothing
Returns	Nothing
Purpose	Default destructor for the MultiThreadQueue class.

Name	MultiThreadQueue::MultiThreadQueue(int size)
Receives	int size //max size to the set the queue to
Returns	nothing
Purpose	to create a multithreadqueue object with the maximum given size.

Name	int MultiThreadQueue::pop(char *buffer, int length)
Receives	char * buffer //buffer to place removed data in int length //length of buffer given
Returns	int //number of bytes placed in the buffer
Purpose	To remove a given number of bytes at once from the queue.

Name	BOOL MultiThreadQueue::pop(char &c)
Receives	char &c //single byte to placed popped byte in from queue
Returns	BOOL TRUE if the byte could be placed in the char &c
Purpose	To remove a single byte from the queue and place it in the given char &c.

Name	int MultiThreadQueue::push(const char *buffer, int length)
Receives	const char * buffer // buffer of bytes to place in the queue int length //number of bytes to place in the queue
Returns	int number of bytes placed in the queue
Purpose	To place an entire buffer in the queue at once.

Name	BOOL MultiThreadQueue::push(char c)
Receives	char c byte to place in the queue
Returns	BOOL TRUE if the char could be placed in the queue
Purpose	To place a single byte in the queue.

A.2.13. PortSelectionDialog Class

This section contains an explanation of the interface for the PortSelectionDialog class. This dialog box allows the users to select the serial port for the application. This class inherits from the CDialog class.

A.2.13.1. Private Data Members

Type and Name	Represents
CString m_PortName	CString Object that contains the name of the serial port entered by the user.

A.2.13.2. Public Method Members

Name	PortSelectionDialog::PortSelectionDialog(CWnd* pParent /*=NULL*/)
Receives	CWnd* pParent //parent window in which the dialog box was created
Returns	nothing
Purpose	This constructor creates a PortSelectionDialog box and displays it to the user.

A.2.13.3. Protected Method Members

Name	void PortSelectionDialog::DoDataExchange(CDataExchange* pDX)
Receives	CDataExchange *pDX //pointer to a CDatatExchange Object
Returns	nothing
Purpose	Transfers data entered to the user to the CString m_PortName object.

Name	int PortSelectionDialog::OnCreate(LPCREATESTRUCT lpCreateStruct)
Receives	LPCREATESTRUCT lpCreateStruct
Returns	int
Purpose	This function was implemented by Visual C++ Studio.

Name	void PortSelectionDialog::OnOK()
Receives	nothing
Returns	nothing
Purpose	This function is called when the “OK” button is pressed. It calls the provided CDialog::OnOK() function.

A.2.14. PowerProfileDoc Class

This section contains an explanation of the interface for the PowerProfileDoc Class. The PowerProfileDoc Class is the document for the document/view MFC architecture. It is used to help control the modem status window.

A.2.14.1. Private Data Members

Type and Name	Represents
vector<double> DataPoints	STL vector template that contains the list of data points obtained from the sounder.

A.2.14.2. Public Method Members

Name	void PowerProfileDoc::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function calls CDocument::AssertValid(); function. The default implementation provided by MFC Visual Studio was used for this function

Name	PowerProfileDoc::~PowerProfileDoc()
Receives	nothing
Returns	nothing
Purpose	Destroys a PowerProfileDoc(). The default implementation provided by MFC Visual Studio was used for this function.

Name	void PowerProfileDoc::Dump(CDumpContext& dc) const
Receives	CDumpContext &dc //device context for the document object
Returns	nothing
Purpose	Calls CDocument::Dump(CDumpContext& dc). The default implementation provided by MFC Visual Studio was used for this function.

Name	BOOL PowerProfileDoc::OnNewDocument()
Receives	nothing
Returns	BOOL //TRUE if the object is initialized correctly
Purpose	This function initializes the PowerProfileDoc object. The default implementation provided by MFC Visual Studio was used for this function

Name	void PowerProfileDoc::Serialize(CArchive& ar)
Receives	CArchive& ar //archive object that the document object will be archived into
Returns	nothing
Purpose	This function was supplied by the MFC Visual Studio and was not used in this program.

A.2.15. PowerProfileView Class

This section contains an explanation of the interface for the PowerProfileView class. The PowerProfileView Class presents the current settings of the adaptive data link layer protocol and the modem controller. This class is apart of the document/view architecture of MFC.

A.2.15.1. Protected Method Members

Name	void PowerProfileView::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	void PowerProfileView::DoDataExchange(CDataExchange* pDX)
Receives	CDataExchange* pDX //pointer to CData Exchange object
Returns	nothing
Purpose	Displays the values contained in the private CEdit objects of this view class on the screen.

Name	void PowerProfileView::Dump(CDumpContext& dc) const
Receives	CDumpContext & dc //device context object

Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	BOOL PowerProfileView::OnPreparePrinting(CPrintInfo* pInfo)
Receives	CPrintInfo* Info
Returns	BOOL //returns result of DoPreparePrinting. If successful, TRUE is returned.
Purpose	To prepare the view for printing. This function was created by C++ Visual Studio and was not used by this program.

Name	PowerProfileView::PowerProfileView()
Receives	nothing
Returns	nothing
Purpose	Default constructor for the PowerProfileView class. This function was created by C++ Visual Studio.

Name	PowerProfileView::~PowerProfileView()
Receives	nothing
Returns	nothing
Purpose	Default destructor for the PowerProfileView class. This function was created by C++ Visual Studio.

Name	void PowerProfileView::OnDraw(CDC* pDC)
Receives	CDC* pDC //device context
Returns	nothing
Purpose	This function is called when the power profile is displayed or printed. It uses the data points collected by the sounder and draws them on the device context.

A.2.15.2. Public Method Members

Name	void PowerProfileView::ConvertTodBc(vector<double> &v,double MultiPathComponentThresholdDB)
Receives	vector<double> &v, //vector of data points collect from the sounder double MultiPathComponentThresholdDB //the voltage noice threshold in dB of the sounder
Returns	nothing
Purpose	Converts all the values in v to dBc. All values below the MultiPathComponentThresholdDB are set to the MultiPathComponentThesholdDB.

A.2.15.3. Private Method Members

Name	int PowerProfileView::GetMinValueIndex(const vector<double> &v)
Receives	const vector<double> &v // STL vector of data points from the sounder
Returns	int //index of the minimum value in the vector
Purpose	Finds the minimum value in the v and returns its index.

Name	void PowerProfileView::ZeroExtend(vector<double> &v)
Receives	vector<double> &v
Returns	nothing
Purpose	This function zero extends the v, so its length is 0 mod 5.

A.2.16. Sounder Class

This section contains an explanation of the interface for the Sounder class. This class performs most of the functionality for the program when the sounder is enabled. It launches two threads. One is a control thread that controls scripts that control the sounder. The second thread is a calculation thread that processes data from the sounder after a complete power profile snap shot.

A.2.16.1. Private Data Members

Type and Name	Represents
double ActualBinTimeSize	The actual bin time size used by the sounder in secs.
double CIR;	carrier to interferance ratio
double alpha	The filter used by the linear filter for graphing the results.
CWinApp * app	Pointer to the main application. This used to help the sounder object transfer data to and from the main application.
double BandwidthOccupiedByPulseMhz	Amount of bandwidth occupied by the sounder pulse in Mhz.
double BcHighMhz	This is the amount of frequency with 90% correlation in Mhz
double BcLowMhz	This is the amount of frequency with 50% correlation in Mhz
double DataCollectionTimeMin	The time to collect data in minutes.
double DataCollectionTimeSec	The amount of time to collect data in seconds
double HostDataRateRequiredBPS	This the required host data rate in bits per seconds.
double Impedence	This is the amount of impedance in the sounder system in Ohms.
CMutex m_SndMutex	This mutex for the sounder. It used to sync the control thread and the calculation thread. It prevents the control thread from over writing the data in the SounderDataVector while the calculation thread is reading it.
double MemBufferSize	This is the sounders memory buffer size in bits.
double MultiPathComponentThresholdVoltage	This is the multipath component threshold in voltage. It is used to remove noise from the data received by the sounder.
double MXD	Mean Excess delay in seconds of the channel
CWinThread* pSounderCalcThread	Pointer to the sounder calculation thread that determines the mean excess delay and RMS delay spread of the channel.
CWinThread* pSounderControlThread	Pointer to the sounder control thread. The control thread launches the sounder scripts that make the sounder gather data on the channel.
CEvent QuitSounderCalcEvent	CEvent object that is used by the sounder object to terminate the calculation thread.
CEvent QuitSounderControlEvent	CEvent object that is used by the sounder object to terminate the control thread
double RDS	RMS delay spread of the channel in seconds.
double RXF	Sampling frequency of the receiving sounder.
CEvent SounderCalcEvent	CEvent object used by the control thread to signal the calculation thread that data is ready from the sounder.

vector<double> SounderDataVector	vector of data points collected from the sounder. This vector is only used by the control thread. All the data in it is transferred to vector<double> DataPoints to perform calculations.
BOOL SounderIsOn	Boolean switch. If set to TRUE, the sounder calculation and control threads have been launched.
double SounderRangeMeters	The sounder range in meters.
ostrstream TimeDiff	This is out string stream used in storing the time elapse for a sounder snapshot.
ofstream TimeDiffFile	Output file that stores the length of time needed for a complete sounder snap shot.
struct _timeb TimeEnded	_timeb structure that stores the end time of a sounder snapshot. It maintains time to millisecond accuracy.
struct _timeb TimeLaunched	_timeb structure that stores the start time of a sounder snapshot. It maintains time to millisecond accuracy.
double TXF	Transmitting frequency of the sounder in Hz.

A.2.16.2. Public Data Members

Type and Name	Represents
CMutex ChSndMutex	Mutex allows the sounder calculation thread to sync with the application. It prevents the calculation thread from changing vector<double> DataPoints while the application is updating the screen with the new data.
vector<double> DataPoints	Data vector containing final calculations of sounder output. These values have been normalized and channel noise has been removed from them.
CString dir	Name of the directory where sounder control scripts are located.
int Mode	Collection mode of the sounder. If it is set to LIVE, then sounder uses sounder control scripts to collect data.
double MultiPathComponentThresholdDB	Multipath Component Threshold in dB. The function RemoveNoise uses this value to remove all noise from the collected data.
int NumberOfIterations	Number of snapshots the sounder object should collect when started. If this value is set to –1, the sounder will collect an infinite number of snapshots.
double RequestedBinTimeSize	Requested bin time in seconds for sounder data.
double TXPulseRepetitionRate	Transmitter repetition rate in Hz.

A.2.16.3. Private Method Members

Name	double Sounder::CalculateCarrierInterferanceRatio(const vector<double> &datapoints)
Receives	const vector<double> &datapoints //points collected from the sounder
Returns	double //Carrier to Intererance Ratio
Purpose	Calculates Carrier to Interferance Ratio. However, if the Interefance is calculated to be zero, 0 is returned. The data points vector must have the noise remove from the data points and the max value should have already been rotated 2 pulsewidths from the beginning to prevent the algorithm from exceeding the indicies of the vector

Name	inline void Sounder::AbsValue(vector<double> &vec)
Receives	vector<double> &vec //vector of data points
Returns	nothing

Purpose	This function takes the absolute value of every value in vec and places it back in vec.

Name	void Sounder:: CalcTime()
Receives	nothing
Returns	nothing
Purpose	dumps the begin and ending time of the sounder snapshot to the ostrstream TimeDiff

Name	inline double Sounder::CalculateFirstMoment(vector<double> &power,vector<double> &timebins)
Receives	vector<double> &power //values collected from the sounder converted to power in Watts vector<double> &timebins // the time value in seconds of each time bin.
Returns	double //The first moment of the vector power weighted by the vector timebins
Purpose	To calculate the first moment of the vector power weighted by the vector timebins.

Name	inline double Sounder::CalculateRMSDelaySpread(double &FirstMoment, double &SecondMoment)
Receives	double &FirstMoment //First moment of the data vector double &SecondMoment // Second moment of the data vector
Returns	double //the standard deviation or RMS delay spread based on the first moment and second moment given.
Purpose	to calculate the standard deviation or RMS delay spread based on the first moment and second moment given.

Name	inline double Sounder::CalculateSecondMoment(vector<double> &power,vector<double> &timebins)
Receives	vector<double> &power //data points from the sounder converted to Power in Watts vector<double> &timebins //the time value in seconds of each time bin
Returns	The second moment of the given vector power weighted by the vector timebins.
Purpose	To calculate the second moment of the vector power weighted by the vector timebins

Name	void Sounder::EditDSPConfigFiles()
Receives	nothing
Returns	nothing
Purpose	Updates the DSP configuration files for the needed memory buffer size for the current channel sounding settings

Name	inline double Sounder::FindMax(const vector<double> &vec)
Receives	const vector<double> &vec // the vector to search through
Returns	double //the maximum value in the vector vec.
Purpose	To find the maximum value in the vector vec.

Name	inline vector<double>::iterator Sounder::FindMaxIterator(vector<double> & vec)
Receives	vector<double> &vec
Returns	vector<double>::iterator //the iterator pointing to the max value in the vector vec
Purpose	To find the iterator for maximum value in the vector vec.

Name	inline vector<double>& Sounder::FloorVector(vector<double> &vec)
Receives	vector<double>& vec //vector to round
Returns	vector<double>& //vector where each value is rounded down to the nearest integer.

Purpose	Takes the given vector and returns a vector with every value rounded down to the nearest integer.

Name	inline void Sounder::LowPassFilter(vector<double>& vec)
Receives	vector<double>& vec //vector to filter
Returns	nothing
Purpose	Uses a linear low pass filter on the vector vec.

Name	inline void Sounder::RemoveNoise(vector<double> &vec)
Receives	vector<double> &vec
Returns	nothing
Purpose	Removes all values in vec below the multipath component threshold

Name	inline vector<double> & Sounder::ScaleAddVector(vector<double> &vec,double scalor)
Receives	vector<double>&vec //vector to add to double scalor //scalar to be added
Returns	vector<double> & //the vector containing the sum of ever value in vec with the given scalar.
Purpose	Adds the given scalar to every element in the vector vec.

Name	inline vector<double> & Sounder::ScaleMulVector(vector<double> &vec,double factor)
Receives	vector<double> & vec// vector to perform multiplication on. double factor// scalar factor to multiply every element in vec by.
Returns	vector<double> & // a vector containing the product of every element in vec with factor
Purpose	Multiplies every element in vec by the given scalar factor.

Name	inline void Sounder::SetIndexVector(vector<double> &vec,double step,int length)
Receives	vector<double> &vec //vector to be made into an index vector double step //space between each index value int length //length of the index vector
Returns	Nothing
Purpose	To create an index vector with the given step between each index value

Name	void Sounder::SetTime()
Receives	Nothing
Returns	Nothing
Purpose	Sets the beginning time of a sounder snapshot.

Name	inline vector<double> Sounder::SignVector(const vector<double> &vec)
Receives	const vector<double> &vec //vector to generate sign mask of
Returns	vector<double> containing sign mask of vector
Purpose	Each element in the returned vector will be of the following values // 0 if the corresponding element in vec was 0 // 1 if the corresponding element in vec was > 0 // -1 if the corresponding element in vec was < 0

Name	int Sounder::SounderCalcThreadFunction(void *arglist)
Receives	void *arglist //pointer to the current sounder objet

Returns	int //exit code of the calculation thread
Purpose	This function launches thread that performs all the calculations on the collected sounder data. First it retrieves the data from the sounder control thread. It calculates the mean excess delay, RMS delay spread, and coherence bandwidth of the channel.

Name	int Sounder::SounderControlThreadFunction(void *arglist)
Receives	void *arglist //void to pointer to the sounder class
Returns	int //exit code of the calculation thread
Purpose	This is the control function for thread that launches the sounder scripts and loads the data from the scripts output file. After the thread launches the scripts it sleeps till the scripts complete. It then loads the data file into memory and signals the main application that the data is in the shared linked list.

Name	inline vector<double> Sounder::VectorCrossProduct(const vector<double> &lhs, const vector<double> &rhs)
Receives	const vector<double> &lhs //left hand side vector for cross product const vector<double> &rhs //right hand side vector for cross product
Returns	vector<double> // the cross product of lhs X rhs
Purpose	This function returns the cross product of the two given vectors *VECTORS MUST BE OF SAME LENGTH*

A.2.16.4. Public Method Members

Name	CString Sounder::GetCIRString()
Receives	Nothing
Returns	Cstring //ascii string form of CIR value collected by Sounder
Purpose	This function converts the CIR value collected by the sounder to a string to be displayed.

Name	void Sounder::SetRecieveFrequency(double value)
Receives	double value
Returns	Nothing
Purpose	This function sets the receive frequency used in the sounder calculations

Name	void Sounder::CalculateChannelState()
Receives	Nothing
Returns	Nothing
Purpose	To Perform the calculations on the channel data read in from the sounder output file

Name	void Sounder::ConfigureSettings()
Receives	nothing
Returns	nothing
Purpose	Configures sounder calculation paramters based on the following public members of this class MultiPathComponentThresholdDB, PulseWidthSec, RequestedBinTimeSize, TXPulseRepetitionRate

Name	double Sounder::GetAlpha()
Receives	nothing

Returns	double //Alpha being used for the linear filter
Purpose	Returns the alpha for the linear filter

Name	CString Sounder::GetBcHighMhz()
Receives	nothing
Returns	CString //string containing current high boundary of the coherence bandwidth
Purpose	To get a string containing the high boundary of the coherence bandwidth

Name	CString Sounder::GetBcLowMhz()
Receives	nothing
Returns	CString //string containing current low boundary of the coherence bandwidth
Purpose	to get a string containing the low boundary of the coherence bandwidth

Name	CString Sounder::GetBDWOccupiedByPulse()
Receives	nothing
Returns	CString //string holding the BDWOccupiedByPulse value in a string
Purpose	To get the Bandwidth Occupied By Pulse in a string

Name	CString Sounder::GetMeanExcessDelay()
Receives	Nothing
Returns	CString //string holding the mean excess delay for the channel
Purpose	To get the Mean Excess Delay for the channel in a string

Name	CString Sounder::GetRMSDelaySpread()
Receives	Nothing
Returns	CString //string holding RMS Delay Spread for the channel
Purpose	To get the RMS Delay spread for the channel in a string

Name	CString Sounder::GetSounderRange()
Receives	nothing
Returns	Cstring //string containing the current sounder range
Purpose	To get a string containing the current sounder range

Name	BOOL Sounder::IsOn()
Receives	nothing
Returns	BOOL //true if the sounder threads are currently active
Purpose	This function returns true if the sounder threads are currently active and the sounder is collecting data.

Name	void Sounder::KillSounderThreads(BOOL wait)
Receives	BOOL wait //TRUE if function should wait to confirm shutdown of threads.
Returns	nothing
Purpose	This function terminates the control and calculation threads of the sounder and sets SounderIsOn = FALSE. If wait == TRUE, the function waits to confirm that the threads have exited. Otherwise it immediately exits.

Name	void Sounder::LaunchSounderThreads()
Receives	nothing
Returns	nothing
Purpose	This function launches the control and calculation threads for the sounder and sets

	SounderIsOn = TRUE.

Name	bool Sounder::LoadSounderDataFile()
Receives	Nothing
Returns	bool //true, if the sounder data file was loaded correctly
Purpose	Loads the sounder data from the sounder scripts output file

Name	void Sounder::SetApp(CWinApp *App)
Receives	CWinApp *App //pointer
Returns	nothing
Purpose	

Name	void Sounder::SetFilterAlpha(double a)
Receives	double a
Returns	nothing
Purpose	Changes the alpha parameter for the filter to the given value a.

Name	void Sounder::SetMultiPathComponentThresholdDB(double value)
Receives	double value
Returns	nothing
Purpose	Sets the MultiPathCompomentThresholdDB to the given value.

Name	void Sounder::SetPulseWidthSec(double value)
Receives	double value
Returns	nothing
Purpose	Sets PulseWidthSec to the given value.

Name	void Sounder::SetRequestedBinTimeSize(double value)
Receives	double value
Returns	nothing
Purpose	Sets RequestedBinTimeSize to the given value.

Name	void Sounder::SetTXPulseRepetitionRate(double value)
Receives	double value
Returns	nothing
Purpose	Sets TXPulseRepetitionRate to the given value.

Name	Sounder::Sounder()
Receives	nothing
Returns	nothing
Purpose	Default constructor for the sounder object. It sets the sounder object to the default configuration.

Name	Sounder::~Sounder()
Receives	nothing
Returns	nothing
Purpose	Default destructor for the sounder object. It ensures that the control thread and calculation thread have exited before exiting.

Name	void Sounder::UpdateSounderStatus()
Receives	nothing
Returns	nothing
Purpose	This function uses PostMessage function to post a message to the main application to that informs the main application that sounder data is ready to be displayed.

A.2.17. SounderParametersDlg Class

This section contains an explanation of the interface for the SounderParametersDlg class. This class creates a modeless dialog box that allows the user to change the parameters for the sounder.

A.2.17.1. Private Data Members

Type and Name	Represents
Sounder * BBSounder	Pointer to sounder object currently in use by the application.

A.2.17.2 Public Data Members

Type and Name	Represents
double m_Alpha	Alpha parameter for sounders linear low pass filter
double m_BinTime	Requested collection bin time in seconds
double m_PulseRate	The transmission pulse rate for the transmitting part of the sounder.
double m_PulseWidth	Pulse width in seconds of the sounder pulse
double m_Threshold	Multi Path Component threshold in dB for the sounder
CString m_Dir	Directory containing sounder scripts
int m_NumberOfIterations	Number of entire power profile delays to collect. If this is set to –1, the sounder will collect an infinite number of power profile delays.
CButton m_LiveMode	CButton object attached to the LIVE mode check button
CButton m_TestMode	CButton object attached to the TEST mode check button

A.2.17.3. Public Method Members

Name	void SounderParametersDlg::SetSounder(Sounder *sounder)
Receives	Sounder* sounder //pointer to the sounder object currently in use by the application
Returns	nothing
Purpose	This connects the SounderParametersDlg to the sounder object in use by the program. It allows the dialog box to change the configuration of the sounder object when the "Apply", "OK", or "Set Default" buttons are pressed.

Name	SounderParametersDlg::SounderParametersDlg(CWnd* pParent /*=NULL*/)
Receives	CWnd* pParent //pointer to the window that the dialog box is attached to.
Returns	nothing
Purpose	This function is the constructor for the SounderParametersDlg box. It can be attached to a parent window if pParent is not NULL.

A.2.17.4. Protected Method Members

Name	void SounderParametersDlg::DoDataExchange(CDataExchange* pDX)
Receives	CDataExchange* pDX //pointer to data exchange object
Returns	nothing
Purpose	Performs the data exchange between the screen and members of the SounderParametersDlg box.

Name	void SounderParametersDlg::OnApply()
Receives	nothing
Returns	nothing
Purpose	This function is called when the “Apply” button is pressed. It applies the entered configuration to the sounder.

Name	void SounderParametersDlg::OnCancel()
Receives	nothing
Returns	nothing
Purpose	Closes the dialog box and does not apply the entered configuration to the sounder.

Name	BOOL SounderParametersDlg::OnInitDialog()
Receives	nothing
Returns	BOOL //always returns TRUE
Purpose	Initializes the dialog box upon startup.

Name	void SounderParametersDlg::OnLiveMode()
Receives	nothing
Returns	nothing
Purpose	Sets m_Mode to LIVE when the “Live” check box is checked on the dialog box.

Name	void SounderParametersDlg::OnTestMode()
Receives	nothing
Returns	nothing
Purpose	Sets m_Mode to TEST when the “Test” check box is checked on the dialog box.

Name	void SounderParametersDlg::OnOK()
Receives	nothing
Returns	nothing
Purpose	This function is called when the “OK” button is clicked. It applies the current configuration to the sounder.

Name	void SounderParametersDlg::OnSave()
Receives	nothing
Returns	nothing
Purpose	This function is called when the “Set as Default” button is clicked. It applies the current configuration and saves it to the configuration file.

A.2.18. SounderStatusDoc Class

This section contains an explanation of the interface for the SounderStatusDoc Class. The SounderStatusDoc Class is the document for the document/view MFC architecture. It is used to help control the sounder status window.

A.2.18.1. Public Method Members

Name	void SounderStatusDoc::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function calls CDocument::AssertValid(); function. The default implementation provided by MFC Visual Studio was used for this function

Name	SounderStatusDoc::~SounderStatusDoc()
Receives	nothing
Returns	nothing
Purpose	Destroys a SounderStatusDoc(). The default implementation provided by MFC Visual Studio was used for this function.

Name	void SounderStatusDoc::Dump(CDumpContext& dc) const
Receives	CDumpContext &dc //device context for the document object
Returns	nothing
Purpose	Calls CDocument::Dump(CDumpContext& dc). The default implementation provided by MFC Visual Studio was used for this function.

Name	BOOL SounderStatusDoc::OnNewDocument()
Receives	nothing
Returns	BOOL //TRUE if the object is initialized correctly
Purpose	This function initializes the SounderStatusDoc object. The default implementation provided by MFC Visual Studio was used for this function

Name	void SounderStatusDoc::Serialize(CArchive& ar)
Receives	CArchive& ar //archive object that the document object will be archived into
Returns	nothing
Purpose	This function was supplied by the MFC Visual Studio and was not used in this program.

A.2.19. SounderStatusView Class

This section contains an explanation of the interface for the SounderStatusView class. The SounderStatusView Class presents data from the sounder in the sounder status window. This class is apart of the document/view architecture of MFC.

A.2.19.1. Public Data Members

Type and Name	Represents

CEdit m_SounderStatus;	This CEdit object displays if the sounder is on or off in the sounder status window.
CEdit m_CIR	This Cedit object displays the CIR collected from the sounder in dB.
CEdit m_SounderRange	This CEdit object displays the range in meters for the sounder
CEdit m_BcLowMhz	This CEdit object displays the amount of bandwidth with a frequency correlation function above 0.5
CEdit m_BCHighMhz	This CEdit object displays the amount of bandwidth with a frequency correlation above 0.9
CEdit m_BDWOccupiedByPulse	This CEdit object displays the amount of bandwidth that is occupied by the sounder pulse.
CEdit m_RMSDelaySpread;	This CEdit object displays the RMS Delay Spread of the channel
CEdit m_MeanExcessDelay	This CEdit object displays the Mean Excess Delay of the channel

A.2.19.2. Public Method Members

Name	void SounderStatusView::AssertValid() const
Receives	nothing
Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	void SounderStatusView::DoDataExchange(CDataExchange* pDX)
Receives	CDataExchange* pDX //pointer to CData Exchange object
Returns	nothing
Purpose	Displays the values contained in the private CEdit objects of this view class on the screen.

Name	void SounderStatusView::Dump(CDumpContext& dc) const
Receives	CDumpContext & dc //device context object
Returns	nothing
Purpose	This function was created by C++ Visual Studio. The default implementation was used in this program.

Name	BOOL SounderStatusView::OnPreparePrinting(CPrintInfo* pInfo)
Receives	CPrintInfo* Info
Returns	BOOL //returns result of DoPreparePrinting. If successful, TRUE is returned.
Purpose	To prepare the view for printing. This function was created by C++ Visual Studio and was not used by this program.

Name	SounderStatusView::SounderStatusView()
Receives	nothing
Returns	nothing
Purpose	Default constructor for the SounderStatusView class. This function was created by C++ Visual Studio.

Name	SounderStatusView::~SounderStatusView()
Receives	nothing
Returns	nothing
Purpose	Default destructor for the SounderStatusView class. This function was created by C++ Visual Studio.

A.2.20. TxDialog Class

This section contains an explanation of the interface for the TxDialog class. This dialog box is a modeless dialog box that allows the user to enter commands to be sent to the modem.

A.2.20.1. Public Data Members

Type and Name	Represents
CEdit TxEdit	Edit control that is attached to the command entry Edit box.

A.2.20.2. Public Method Members

Name	TxDialog::TxDialog(CWnd* pParent /*=NULL*/)
Receives	CWnd* pParent //pointer to the parent window.
Returns	nothing
Purpose	Creates the TxDialog box. The dialog box can be attached to a parent window if pParent is not set to NULL.

A.2.20.3. Protected Method Members

Name	void TxDialog::DoDataExchange(CDataExchange* pDX)
Receives	CDataExchange *pDX // pointer to the data exchange
Returns	nothing
Purpose	Performs the data exchange between TxEdit and the screen.

Name	void TxDialog::OnTxCancel()
Receives	nothing
Returns	nothing
Purpose	This function hides the TXDialog box.

Name	void TxDialog::OnTxSend()
Receives	nothing
Returns	nothing
Purpose	This function takes the text entered in TXEdit and sends it to the modem controller.

A.2.21. Win32SerialPort Class

This section provides an explanation of the interface for the Win32SerialPort class. This class launches two threads. One performs the reading of the serial port. The other performs the writing to the port. This prevents the program from locking up while the user is using the serial port. This class also allows any serial port to be opened and use any baudrate or framing that NT supports.

A.2.21.1. Private Data Members

Type and Name	Represents
CEvent BreakRequestedEvent	CEvent object that is sent when a break on the serial port is received.

MultiThreadQueue InputBuffer	Mutex protected queue that holds incoming bytes until the application reads them.
CEvent InputThreadKilledEvent	CEvent object that is set when the input thread has been shutdown.
HANDLE InputThreadP	Handle to the input thread used to read from the serial port.
BOOL is_open	BOOL to determine if the serial port is open. If set to TRUE, the serial port is opened.
CEvent KillInputThreadEvent	CEvent object that is set to signal the input thread to shutdown.
CEvent KillOutputThreadEvent	CEvent object that is set to signal the output thread to shutdown.
DWORD m_BreakDuration	Duration of the serial port break in milliseconds.
HANDLE m_hPort	Handle to the serial port that is currently open by the serial port object.
BOOL m_InputThreadReading	BOOL used to tell if the input thread is actively reading. If TRUE, the thread is reading from the port.
DWORD m_LastError	DWORD containing the last error status word of the serial port.
MultiThreadQueue OutputBuffer	Mutex protected queue that holds bytes ready to written to the serial port.
CEvent OutputThreadKilledEvent	CEvent object that is set when the output thread has shutdown.
HANDLE OutputThreadP	Handle to the output thread of the serial port object.
string port_name	Character string containing the name of the serial port in use by the serial port object.
CEvent ReadRequestedEvent	CEvent object that is set by the main application thread to signal the input thread that the application needs to read data from the port.
DCBclass settings	DCBclass containing the settings of the serial port.
CEvent WriteRequestedEvent	CEvent object that is set by the main application thread to signal the signal the output thread that data needs to be written to the port.

A.2.21.2. Public Method Members

Name	BOOL Win32SerialPort::Adjust(DWORD baud_rate, BYTE parity, BYTE word_length, BYTE stop_bits, DWORD dtr, DWORD rts, DWORD xon_xoff, DWORD rts_cts, DWORD dtr_dsr)
Receives	All the following parameters can be set to the value SAME if you do not want them to change from the system default settings. DWORD baud_rate, baud rate of the serial port. For example, 9600 BYTE parity, Amount of parity to be used for the port. The parameter can take on the following values EVENPARITY Even MARKPARITY Mark NOPARITY No parity ODDPARITY Odd SPACEPARITY Space BYTE word_length, word length to be sent across the link, 5,7, or 8 for example BYTE stop_bits, number of stop and start bits. The following values can be used ONESTOPBIT 1 stop bit ONE5STOPBITS 1.5 stop bits TWOSTOPBITS 2 stop bits DWORD dtr, Raises or lowers the DTR line. It can have the following values SET raise the DTR line UNSET lower the DTR line

	This value must be SET for normal hardware flow control DWORD rts, Raises or lowers the RTS line. It can have the following values SET raise the DTR line UNSET lower the DTR line This value must be SET for normal hardware flow control DWORD xon_xoff, This value enables or disable XON/XOFF handshaking ENABLE handshaking on DISABLE handshaking off DWORD rts_cts, This value enables or disable RTS/CTS handshaking ENABLE handshaking on DISABLE handshaking off DWORD dtr_dsr, This value enables or disable DTR/DSR handshaking ENABLE handshaking on DISABLE handshaking off
Returns	TRUE if the port properties are changed successfully
Purpose	Changes the serial port properties to the given settings in the parameters.

Name	BOOL Win32SerialPort::Adjust(LPDCB newsettings)
Receives	LPDCB newsettings //pointer to a DCB structure
Returns	BOOL //TRUE if the settings were valid
Purpose	Changes the settings of the serial port to those given in the LPDCB structure.

Name	BOOL Win32SerialPort::Close()
Receives	Nothing
Returns	BOOL TRUE if the serial port could be closed
Purpose	To close the serial port that is attached to the serial port object

Name	string Win32SerialPort::GetPortName()
Receives	nothing
Returns	string //name of the port if it is open
Purpose	Returns the name of the port that is open. If the port isn't open, and empty string is returned;

Name	BOOL Win32SerialPort::IsOpen()
Receives	Nothing
Returns	BOOL //TRUE if the port is open
Purpose	This member function is called to check if the port is opened or closed

Name	BOOL Win32SerialPort::Open(const string &port, DWORD baud_rate, BYTE parity, BYTE word_length, BYTE stop_bits, DWORD dtr, DWORD rts, DWORD xon_xoff, DWORD rts_cts, DWORD dtr_dsr)
Receives	const string &port, serial port name, follow UNICODE naming convention, for example "\\\\.\\COM1" will open COM port 1 All the following parameters can be set to the value SAME if you do not want them to change from the system default settings.

	DWORD baud_rate, baud rate of the serial port. For example, 9600 BYTE parity, Amount of parity to be used for the port. The parameter can take on the following values EVENPARITY Even MARKPARITY Mark NOPARITY No parity ODDPARITY Odd SPACEPARITY Space BYTE word_length, word length to be sent across the link, 5,7, or 8 for example BYTE stop_bits, number of stop and start bits. The following values can be used ONESTOPBIT 1 stop bit ONE5STOPBITS 1.5 stop bits TWOSTOPBITS 2 stop bits DWORD dtr, Raises or lowers the DTR line. It can have the following values SET raise the DTR line UNSET lower the DTR line This value must be SET for normal hardware flow control DWORD rts, Raises or lowers the RTS line. It can have the following values SET raise the DTR line UNSET lower the DTR line This value must be SET for normal hardware flow control DWORD xon_xoff, This value enables or disable XON/XOFF handshaking ENABLE handshaking on DISABLE handshaking off DWORD rts_cts, This value enables or disable RTS/CTS handshaking ENABLE handshaking on DISABLE handshaking off DWORD dtr_dsr, This value enables or disable DTR/DSR handshaking ENABLE handshaking on DISABLE handshaking off
Returns	TRUE if the port is opened successfully.
Purpose	This member function opens a serial port with the given settings.

Name	BOOL Win32SerialPort::Open(const string &port, LPDCB dcb)
Receives	const string &port, serial port name, follow UNICODE naming convention, for example "\\\\.\\COM1" will open COM port 1 LPDCB newsettings //pointer to a DCB structure
Returns	TRUE if the port is opened successfully.
Purpose	This member function opens a serial port with the given port settings in LPDCB.

Name	DWORD Win32SerialPort::Read(string &buffer,int buffer_length)
Receives	string &buffer buffer to read data from serial port into int buffer_length amount of data to read from the port in bytes
Returns	DWORD number of bytes returned in the buffer
Purpose	This function is only provided so C++ strings can be used with a serial port

Name	DWORD Win32SerialPort::Read(char * buffer, int buffer_length)
Receives	char * buffer buffer to read data into int buffer_length length of the given buffer
Returns	DWORD number of bytes returned in the buffer
Purpose	Reads data received on the serial port into the given buffer

Name	BOOL Win32SerialPort::ReadByte(char &byte_buffer)

Receives	char &byte_buffer //one byte buffer to put a byte in
Returns	BOOL TRUE if a valid byte was placed in the buffer
Purpose	To read a signal byte from the serial port

Name	BOOL Win32SerialPort::SendBreak(DWORD length)
Receives	DWORD length //time length in milliseconds for the break duration
Returns	TRUE if the break was performed successfully
Purpose	Sends a break on the communication port

Name	Win32SerialPort::Win32SerialPort()
Receives	Nothing
Returns	Nothing
Purpose	Creates a Win32SerialPort object. It does not open the serial port. However, it does initialize the input and output buffers. In order to use the serial port, the Open() member function must be called.

Name	Win32SerialPort::Win32SerialPort(const string &port, DWORD baud_rate, BYTE parity, BYTE word_length, BYTE stop_bits, DWORD dtr, DWORD rts, DWORD xon_xoff, DWORD rts_cts, DWORD dtr_dsr)
Receives	const string &port, serial port name, follow UNICODE naming convention, for example "\\\\.\\COM1" will open COM port 1 All the following parameters can be set to the value SAME if you do not want them to change from the system default settings. DWORD baud_rate, baud rate of the serial port. For example, 9600 BYTE parity, Amount of parity to be used for the port. The parameter can take on the following values EVENPARITY Even MARKPARITY Mark NOPARITY No parity ODDPARITY Odd SPACEPARITY Space BYTE word_length, word length to be sent across the link, 5,7, or 8 for example BYTE stop_bits, number of stop and start bits. The following values can be used ONESTOPBIT 1 stop bit ONE5STOPBITS 1.5 stop bits TWOSTOPBITS 2 stop bits DWORD dtr, Raises or lowers the DTR line. It can have the following values SET raise the DTR line UNSET lower the DTR line This value must be SET for normal hardware flow control DWORD rts, Raises or lowers the RTS line. It can have the following values SET raise the DTR line UNSET lower the DTR line This value must be SET for normal hardware flow control DWORD xon_xoff, This value enables or disable XON/XOFF handshaking ENABLE handshaking on DISABLE handshaking off

	DWORD rts_cts, This value enables or disable RTS/CTS handshaking ENABLE handshaking on DISABLE handshaking off DWORD dtr_dsr, This value enables or disable DTR/DSR handshaking ENABLE handshaking on DISABLE handshaking off
Returns	Nothing
Purpose	This constructor creates a serial port object and opens the serial port with given settings

Name	Win32SerialPort::~Win32SerialPort()
Receives	Nothing
Returns	Nothing
Purpose	Default Desctructor for the Win32SerialPort class. It calls the Close() member function to ensure that the input and output threads are stopped and the file handle for the serial port is returned to the operating system.

Name	DWORD Win32SerialPort::Write(char *buffer,int buffer_length)
Receives	char * buffer buffer of data to be written to port int buffer_length number of bytes to be written to the port
Returns	DWORD number of bytes written to the port
Purpose	This member function writes the given buffer to the serial port

Name	DWORD Win32SerialPort::Write(string buffer, int buffer_length)
Receives	string buffer buffer of data to be written to the port int buffer_length number of bytes to write to the port
Returns	DWORD number of bytes written to the port
Purpose	This member function writes the given buffer to the serial port.

Name	BOOL Win32SerialPort::WriteByte(char byte_buffer)
Receives	char byte_buffer one byte buffer to be written to the serial port
Returns	TRUE if the byte is written to the serial port
Purpose	This function writes a one byte buffer to the serial port.

A.2.21.3. Private Member Methods

Name	void Win32SerialPort::InputControlThread(void *voidp_thisport)
Receives	void * voidp_thisport void pointer to the calling Win32SerialPort object
Returns	int EXIT_SUCCESS if the input thread closed without any errors
Purpose	This member function is the main function for the input thread. This thread reads data from the serial port using the Win32 OVERLAPPED IO method

Name	void Win32SerialPort::OutputControlThread(void *voidp_thisport)
Receives	void *voidp_thisport void pointer to the calling Win32SerialPort object
Returns	int EXIT_SUCCESS if the thread closed without any errors
Purpose	This member function is the main function for the output thread. The output thread writes data from the ouptutbuffer to the serial port using the Win32 OVERLAPPED IO method.

Name	bool Win32SerialPort::write_output_buffer()
Receives	Nothing

Returns	bool true if the ouputthread has been requested to shutdown false, otherwise
Purpose	This member function does the actuall work for the OutputControlThread member function. It performs the OVERLAPPED IO on the port. It also informs the output thread to shut down if a kill request is received while writing to the port.

A.2.21.4. Protected Member Methods

Name	virtual void NotifyOfRx(DWORD bytes_recieved)
Receives	DWORD bytes_received
Returns	nothing
Purpose	This is a virtual function provided with an empty implementation. It is to be overridden to allow applications use the windows message to notify when data has been received on the report.

Name	virtual void NotifyOfTx()
Receives	nothing
Returns	nothing
Purpose	This is a virtual function provided with an empty implementation. It is to be overridden to allow applications use the windows message to notify when data is ready to be written to the serial port.

Appendix B: User's Manual for LMDS Modem Controller Software

This appendix contains a user manual for the LMDS Modem Controller Software developed.

B.1. Installation

Copy the "LMDSMCConsole.exe" and "LMDSMC.ini" files from a previous installation. Copy the entire "BroadbandSounder" from the previous installation to the new installation.

B.2. Configuration

To configure the LMDS Modem controller software before the first startup, open the "LMDSMC.ini" using notepad. Below is an example copy of the "LMDSMC.ini" file.

```
Sounder Parameters
Filter Alpha = 0
MultiPath Component Threshold DB = -15
Requested Bin Time Size = 1.86e-009
Pulse Width In Sec = 4e-009
TX Pulse Repetition Rate = 65536
Sounder Script Directory = D:\BroadbandSounder
Mode = LIVE
Number Of Iterations = 1

Adaptive Protocol Parameters
Throughput Control Switch Point = 33
Delay Control High Switch Point = 43
Delay Control Low Switch Point = 33
Tolerance = 1.0
```

The remainder of this section explains each item in the "LMDSMC.ini" file.

B.2.1. Sounder Parameters

The sounder parameters section of the "LMDSMC.ini" file contains all items that control the broadband sounder's operation.

B.2.1.1. Filter Alpha

This Filter Alpha refers to the alpha parameter that controls the linear filter performed on the sounder data. The equation for the filter is $y[i] = y[i] + \alpha * y[i-1]$. The alpha parameter may values between 0 and 1. A value of 0 turns the filter off.

B.2.1.2. MultiPath Component Threshold DB

The MultiPath Component Threshold DB is noise threshold of the sounder in dB. Typically this value is set to –15dB.

B.2.1.3. Requested Bin Time Size

The Requested Bin Time Size is the bin time size in seconds used by the broadband sounder for sampling the incoming pulse.

B.2.1.4. Pulse Width In Sec

The Pulse Width In Sec is the width of sounder pulse in seconds.

B.2.1.5. TX Pulse Repetition Rate

The TX Pulse Repetition Rate is the transmission pulse rate in Hz of the pulse transmitted by the broadband sounder.

B.2.1.6. Sounder Script Directory

The Sounder Script Directory is the location of the broadband sounder scripts developed by Christian Rieser.

B.2.1.7. Mode

The LMDS Modem Controller software has two operation modes. The Mode can be set to LIVE or TEST. In LIVE mode, the LMDS Modem Controller software operates the sounder using the scripts in the sounder script directory. In TEST mode, the software does not gather data from the sounder. It reads in values from ”testdata.txt” every time the sounder is enabled. The “testdata.txt” file is located in the same directory where the “LMDSMCConsole.exe” is located.

B.2.1.8. Number Of Iterations

The Number Of Iterations parameter controls the number of times the sounder will take an entire power delay profile. If the value is set to –1, the software will continue to take power delay profiles from the sounder until the sounder is disabled.

B.2.2. Adaptive Protocol Parameters

Each of the values in this section control the operation of the adaptive protocol used on the LMDS network. These values contain the threshold for switching between each level of FEC coding and ARQ setting.

B.2.2.1. Throughput Control Switch Point

Single switching point used in the throughput control algorithm described in section 6.4.2. The units are dBc.

B.2.2.2. Delay Control High Switch Point

There are three states used for the end-to-end delay control algorithm. The delay control high switch point is the point where the algorithm switches between no FEC with ARQ (see section 6.4.3.). The units are dBc.

B.2.2.3. Delay Control Low Switch Point

The delay low switch point is the point where the algorithm switches between no FEC without ARQ and high FEC with ARQ. The units are dBc.

B.2.2.4. Tolerance

Tolerance control value is the ε described in section 6.4.2 and 6.4.3. The units are dB.

B.3. Connecting to the Modem Controller

To connect to the modem controller, go to File->Open Console. This brings up a dialog box requesting the number of the serial port to which the modem controller is connected. The number of the serial port can be between 1 and 99. After the number of the serial port is opened, a dialog box appears showing the current design settings for the modem controller serial port. Adjust the parameters if necessary. After making any changes to the settings, click the OK button. The modem controller console window should now appear.

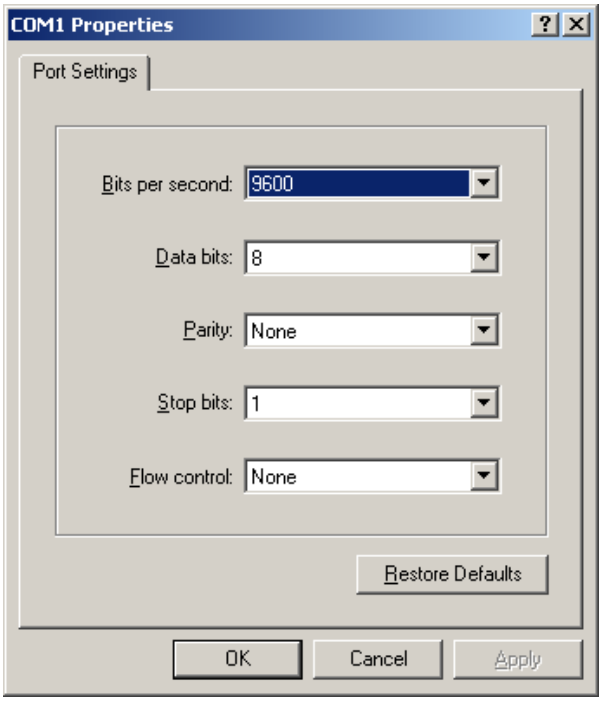

Figure B.1. Serial Port Properties Dialog for the Modem Controller Serial Port.

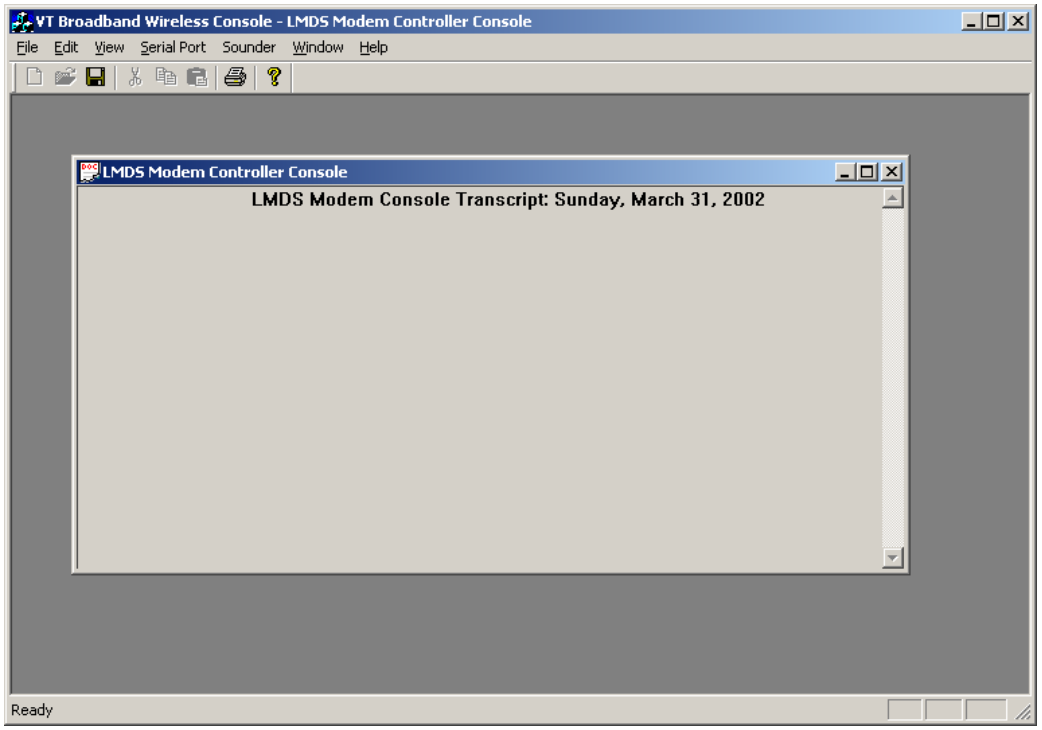

Figure B.2. LMDS Modem Console Window

B.4. The View Menu

This section explains the functionality available under the View menu.

B.4.1 Viewing Modem Status

To view the current configuration of the modem and the adaptive protocol, select Modem Status from the View menu. The Figure B.3 shows the modem status window that should appear.

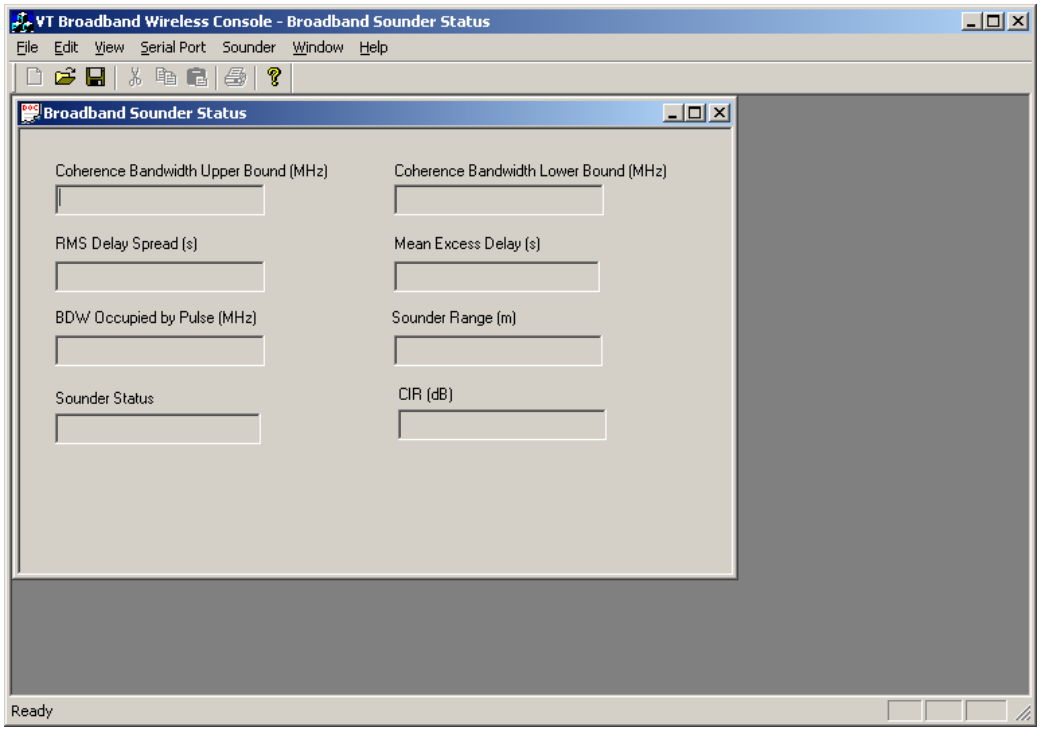

Figure B.3. LMDS Modem Controller Status Window

B.4 2.Transmitting Commands to the Modem Controller Manually

To transmit a command to the modem controller manually, select Transmit Command Dialog from the View menu. To send a command, enter a command in the dialog box (see Figure B.4.) and press the OK button. To close the dialog box, press the cancel button.

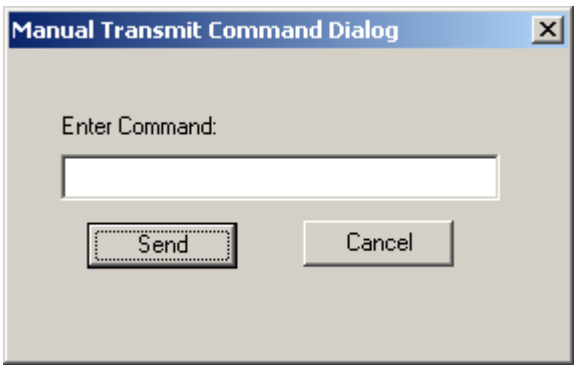

Figure B.4. Manual Transmit Command Dialog for the Modem Controller

B.4.3. Viewing Broadband Sounder Status

To view the status of the broadband sounder, select Sounder Status from the View menu. This window (see Figure B.5.) will not contain any values until the broadband sounder is enabled while this window is open.

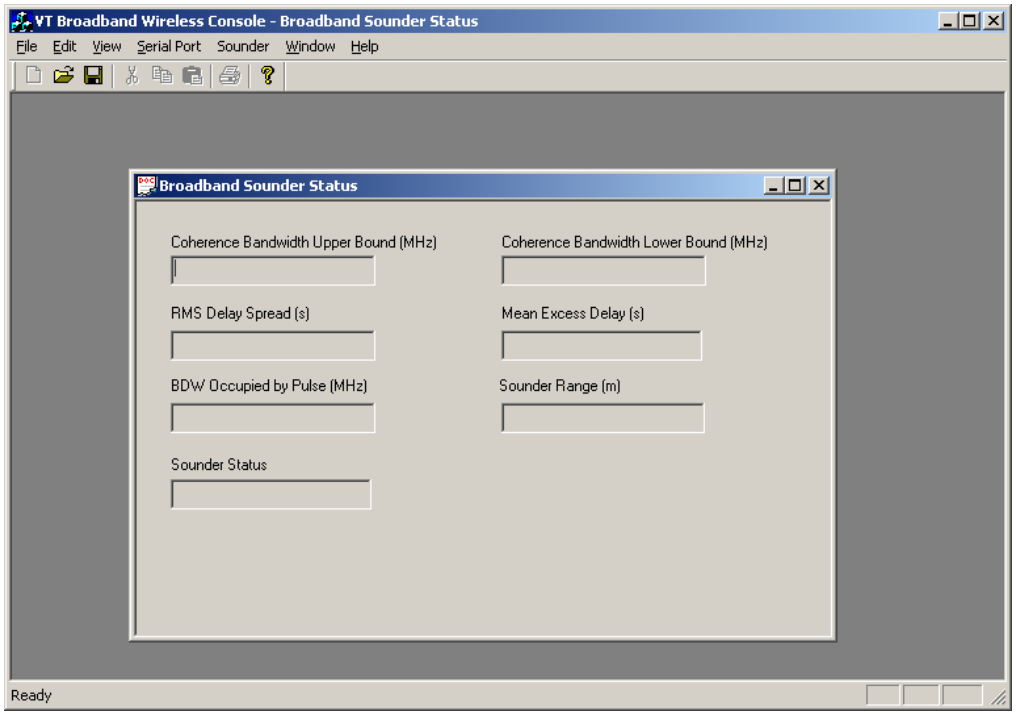

Figure B.5. Broadband Sounder Status Window

B.4.4. Viewing the Channel Power Profile

To view a graph of the channel power profile, select Channel Power Profile from the View menu. This window (see Figure B.6.) presents a graph of the power delay profile of the channel. The graph will remain empty until the broadband sounder is enabled while the window is opened.

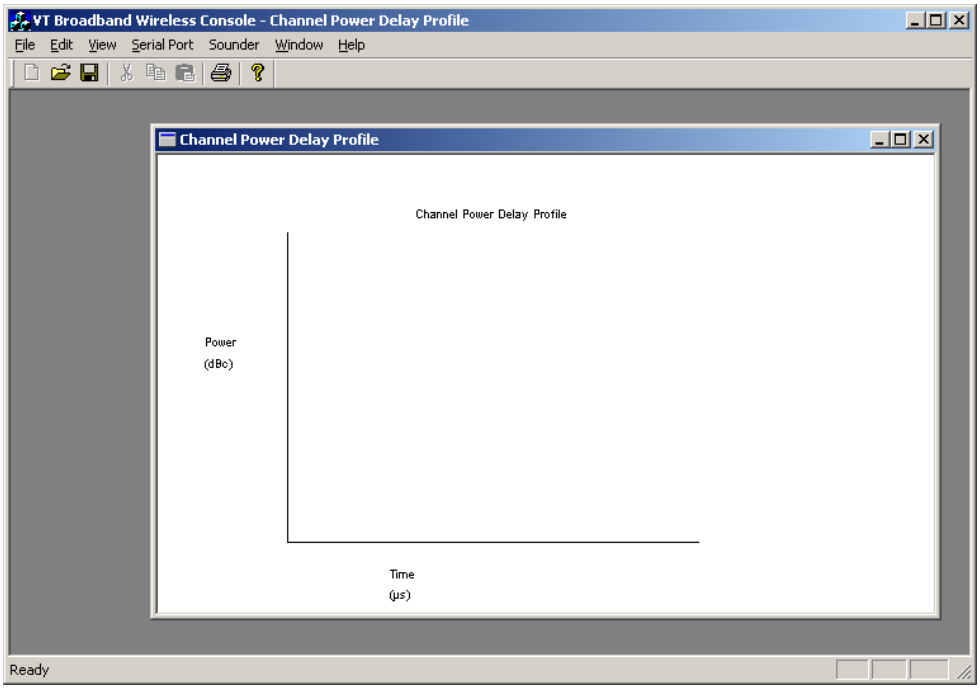

Figure B.6. Channel Power Delay Profile Window

B.5. Serial Port Menu

This section explains the functionality available under the Serial Port menu.

B.5.1. Opening the Serial Port

The first option under the Serial Port menu provides another way to open a connection to the modem controller. Selecting Open Port under the Serial Port menu starts the same process described under Section B.3.

B.5.2. Changing the Properties of the Serial Port without Closing the Port

To change the properties of the serial port connection to the modem controller without closing it, select the Properties under the Serial Port menu. The dialog box in Figure B.1 appears. Change any properties and click the OK button.

B.5.3. Closing the Serial Port

To close the serial port to the modem controller, select Close under the Serial Port menu. The serial port connected to the modem controller automatically closes when the program is closed.

B.6. The Sounder Menu

This section describes the functionality available under the Sounder menu.

B.6.1. Enabling and Disabling the Sounder

To enable the sounder, select Enable Sounder under the Sounder menu. After enabling the sounder, the sounder will begin to as many power profiles as specified in the "LMDSMC.ini" file (see Section B.2.) To disable the sounder, select Disable Sounder under Sounder menu after the sounder has been enabled.

B.6.2. Changing the Sounders Parameter

To change the parameters of the sounder without changing the "LMDSMC.ini" file and restarting the program, select Parameters under the Sounder menu. Change the parameters in the dialog box as needed (see Figure B.7.) Click OK to apply the changes and close out the dialog box. Select Apply to apply the changes without closing the dialog box. Select Cancel to close the dialog box without making any changes. Select Save As Default to write the changes made to the "LMDSMC.ini" file.

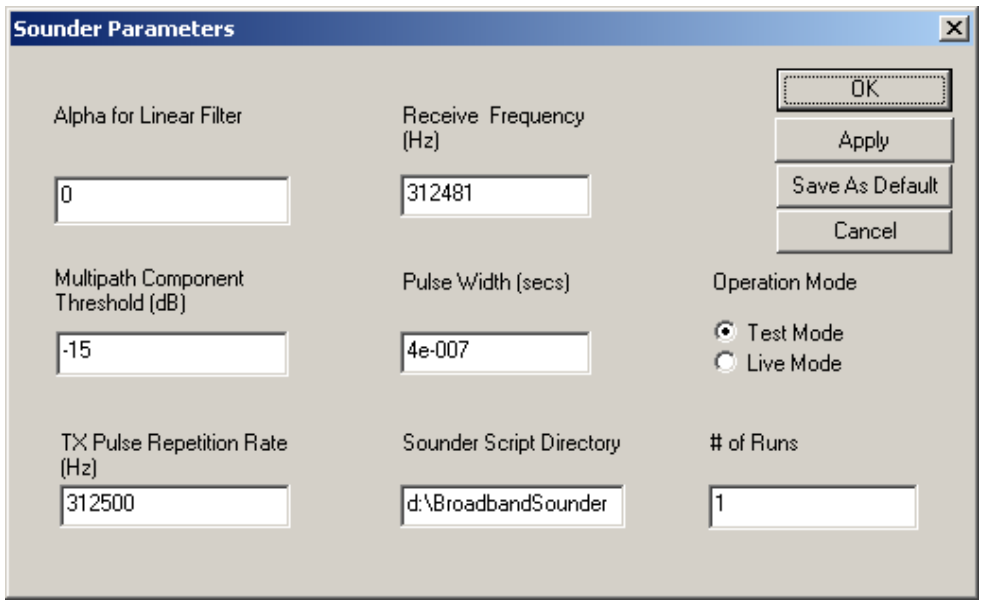

Figure B.7. Sounder Parameters Dialog Box

B.6.3. Enabling and Disabling Throughput and Delay Control Algorithms

Under the Sounder menu, there are two menu items that disable and enable the throughput and delay control algorithms. Only one algorithm can be enabled at a time. The state of the algorithms are updated every time the broadband sounder collects data of the channel.

Appendix C: Modem Controller Commands

This appendix provides an overview of the command set for the modem controller. It also describes the data communications format for the control port.

C.1. Data Communications Format

The modem commands are sent across a serial port with the properties shown below in the chart.

Table C.1. Serial Port Settings for the Modem Controller

Baud Rate	56000
Data Word Size	8 bits
Number of Start/Stop Bits	1 bit
Parity	None
Flow Control	Xon/Xoff

C.2. Command Format

The commands are sent across the serial link in frames using the following format:

Table C.2. Modem Controller Command Format

Command Code	Space	Parameters	End Command Marker
8 bit ASCII character	*space*[1]	Any integer multiple of 8 bit ASCII characters	"\r\n" Carriage return followed by a line feed[2]

Command frames are at least 24 bits long. Each command frame ends with a carriage return followed by a line feed. Each command code is a single letter. If the command has parameters, a command code is followed by a space (0x20). Each command parameter will be separated by a space (0x20).

The command structure has been designed to allow for more commands to be added to the command set. The structure also allows for commands with more than one option to be added.

[1] *space* represents an ASCII space (hex value 20).
[2] \r in this document represents a carriage return (hex value 0D). \n in this document represents a line feed (hex value 0A)..

C.3. Commands

Table C.3. Modem Controller Commands

Command	Command Code	Parameters	Description
HELLO	'H'	None	Test command used to make sure serial communication between modem controller and computer host are functioning
STATUS	'S'	None	Command used to request the current status of the modem
TDMA MODE	'M'	'0' Static TDMA Mode '1' Dynamic TDMA Mode	Command used to set the TDMA mode of the modem controller. The TDMA mode can be static or dynamic
NUMBER OF REMOTES	'N'	Number of remotes in the system in decimal	Command used to inform the modem of the number of remotes in the network. This command should only be used when Static TDMA Mode has been set. The modem controller should ignore this command, if Dynamic TDMA mode has been set.
REMOTE ID	'R'	ID of the remote in decimal	Command used to set the ID of the remote to be used for Static TDMA mode. The hub modem controller must receive an ID of 0. The remote modem controller IDs range from 1 to 99.
FEC LEVEL	'F'	Level of FEC to used for transmission '0' for none '1' for low FEC '2' for medium FEC '3' for high FEC	Command used to set the level of FEC to be used for transmission.
ARQ	'A'	ARQ On/Off '0' for OFF '1' for ON	Command used to turn the ARQ scheme on and off.
ERROR STAT REQUEST	'E'	None	Command used to request error statistics from the modem controller
TRANSMIT	'T'	'0' for OFF '1' for ON 'A' for AUTO 'L' for LOOPBACK	Command used to turn the transmitter in the modem on or off. When AUTO is set, the modem uses the current TDMA scheme to turn the transmitter on or off. In LOOPBACK mode, the modem redirects output back to its input.

C.4. Modem Responses to Commands

Except for the STATUS and ERROR STAT REQUEST commands, the modem controller should respond to all commands successfully received and executed by echoing the received command back to the host computer.

The STATUS command response should be as follows:

Table C.4. Modem Controller Responses

'S'	*space*	'M'	*space*
Current TDMA Mode being used	*space*	'N'	*space*
Current number of remotes using the system	*space*	'R'	*space*
ID of the remote	*space*	'F'	*space*
Current FEC Level being used	*space*	'A'	*space*
State of ARQ Scheme	*space*	'T'	*space*
Current Transmit mode	"\r\n"		

The response for the ERROR STAT REQUEST command has yet to be determined.

The modem should be able to respond to error in commands and unknown commands. The modem should respond to the errors by sending the following:

Table C.5. Modem Controller Error Response Format

Error Response Code	Space	Error Type Code	End Command Marker
'?'	*space*	Code of the type of error received	"\r\n"

Below is a table of the error type codes.

Table C.6. Modem Controller Error Response Types

Error Type	Code	Description
Unknown Command or syntax error	'U'	Code to be used for commands that modem does not know or could not read correctly due to syntax error
Parameter Error	'P'	Code used to specify that the command was understood, but it could not be executed because of an invalid parameter

Appendix D: Annotated ARQ State Log

This appendix presents an annotated log of the ARQ scheme operation. This log was taken from a session where Node 1 was transmitting to the Hub.

```
Transmitting packet 0
Received ack 1      /*Packet 0 successfully received by Hub*/
Received ack 1
Received ack 1
Received ack 1
Received ack 1
Received ack 1
ack distance 0
Transmitting packet 1
Received ack 1      /*Packet 1 successfully received by Hub*/
Received ack 1
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Received ack 2
ack distance 0      /*No Packet transmitted because of empty queue*/
Received ack 2
ack distance 0
Transmitting packet 2
Received ack 3      /*Packet 2 successfully received by Hub*/
Received ack 3
Received ack 3
Received ack 3
Received ack 3
Received ack 3
ack distance 0
Transmitting packet 3
Received ack 3      /*Packet 3 successfully received by Hub*/
Received ack 3
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 0
Received ack 0   /*No Packet transmitted because of empty queue*/
ack distance 0
Transmitting packet 0
Received ack 1      /*Packet 0 successfully received by Hub*/
Received ack 1
Received ack 1
Received ack 1
Received ack 1
Received ack 1
ack distance 0
Transmitting packet 1
Received ack 1      /*Packet 1 successfully received by Hub*/
Received ack 1
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Received ack 2
ack distance 0
```

```
Received ack 2      /*No Packet transmitted because of empty queue*/
ack distance 0
Transmitting packet 2
Received ack 3      /*Packet 2 successfully received by Hub*/
Received ack 3
Received ack 3
Received ack 3
Received ack 3
Received ack 3
ack distance 0
Transmitting packet 3
Received ack 3      /*Packet 3 successfully received by Hub*/
Received ack 3
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 0
Received ack 0      /*No Packet transmitted because of empty queue*/
ack distance 0
Transmitting packet 0
Received ack 0      /*Packet 0 was NOT received by Hub*/
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 1
Transmitting packet 1 /* Sending window not full, no retransmission performed*/
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 2      /*Queue empty*/
fast retransmitting 0  /*Performing fast retransmission*/
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 2      /*Queue empty*/
fast retransmitting 0 /*Performing fast retransmission*/
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 2      /*Queue Empty*/
fast retransmitting 0 /*Performing fast retransmission*/
Received ack 0
Received ack 0
```

```
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 2      /*Sending window not full*/
Transmitting packet 2  /*Transmitting packet 2*/
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 3      /*Sending window full*/
Retransmitting packet 0  /*Retransmitting Packet 0*/
Received ack 0
Received ack 0
Received ack 1
Received ack 1
Received ack 1
Received ack 1
Received ack 1
Received ack 1
Retransmitting packet 1 /*Retransmitting Packet 1*/
Received ack 1
Received ack 1
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Retransmitting packet 2 /*Retransmitting Packet 2*/
leaving retransmission window
Received ack 2
Received ack 2
Received ack 3
Received ack 3
Received ack 3
Received ack 3
Received ack 3
Received ack 3
ack distance 0
Transmitting packet 3 /*Transmitting Packet 3*/
Received ack 3
Received ack 3
Received ack 0      /*Packet 3 successfully received by Hub*/
Received ack 0
Received ack 0
Received ack 0
Received ack 0
Received ack 0
ack distance 0
Transmitting packet 0 /*Transmitting Packet 0*/
Received ack 0
Received ack 0

Received ack 1      /*Packet 0 successfully received by Hub*/
Received ack 1
Received ack 1
```

```
Received ack 1
Received ack 1
Received ack 1
ack distance 0
Transmitting packet 1      /*Transmitting Packet 1*/
Received ack 1
Received ack 1
Received ack 1             /*Packet 1 not received*/
Received ack 1
Received ack 1
Received ack 1
Received ack 1
Received ack 1
ack distance 1
fast retransmitting 1      /*Fast retransmitting Packet 1*/
Received ack 1
Received ack 1
Received ack 2             /Fast retransmission was successful*/
Received ack 2
Received ack 2
Received ack 2
Received ack 2
Received ack 2
ack distance
```

Appendix E: Figures of Results

This section contains the figures referenced in Chapter 6. These figures allow for easy comparison of simulation results.

E.1. Figures for Connections Not using Link Layer ARQ

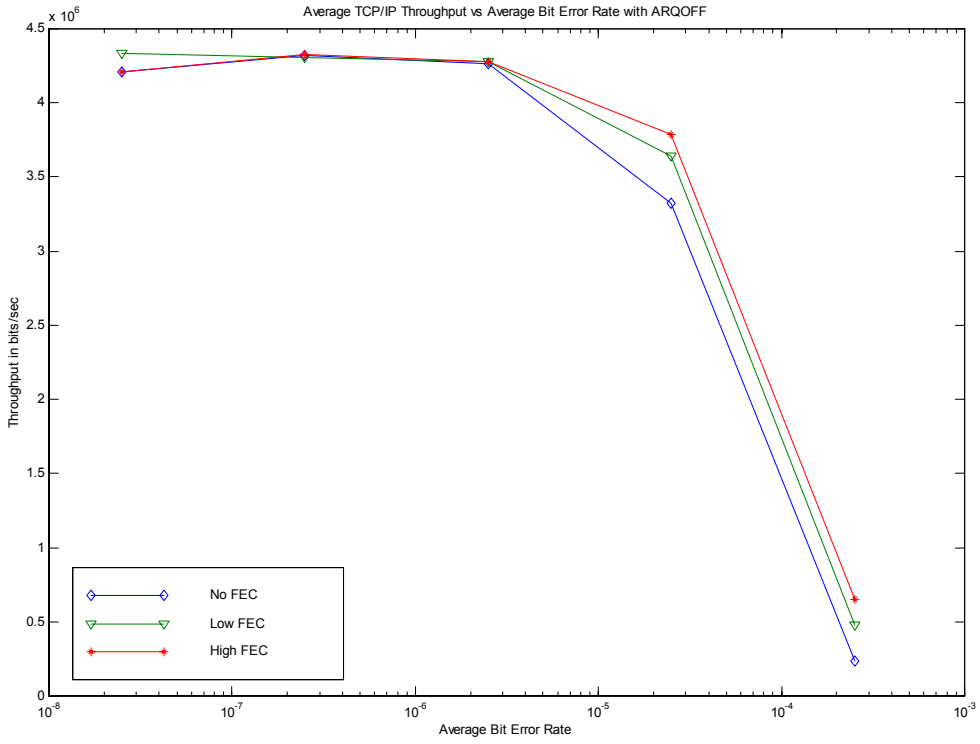

Figure E.1. Average TCP/IP throughput versus average bit error rate for varying FEC.

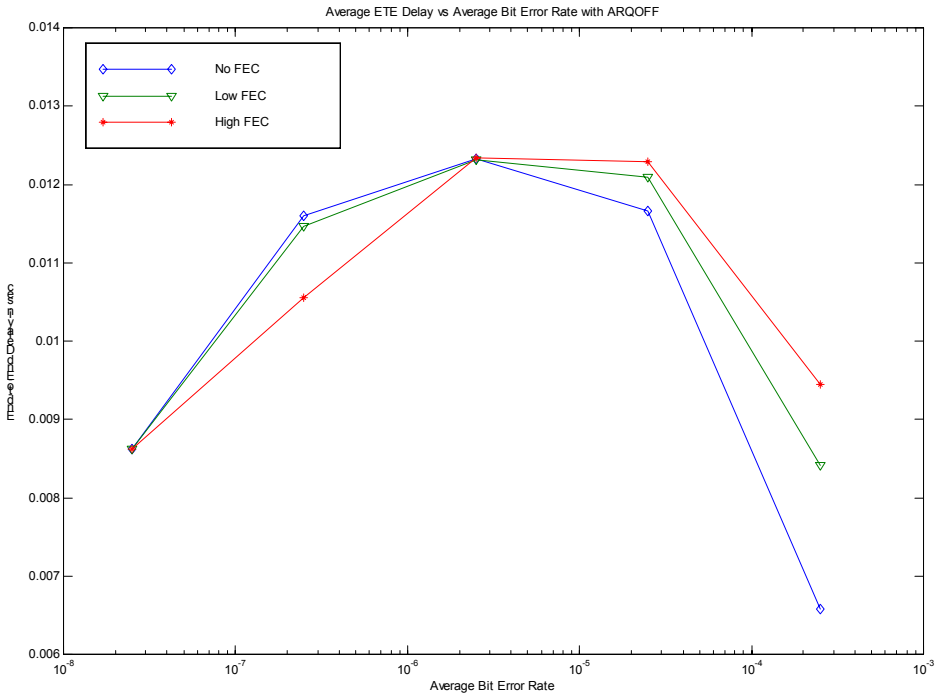

Figure E.2. Average end-to-end delay versus average bit error rate for varying FEC.

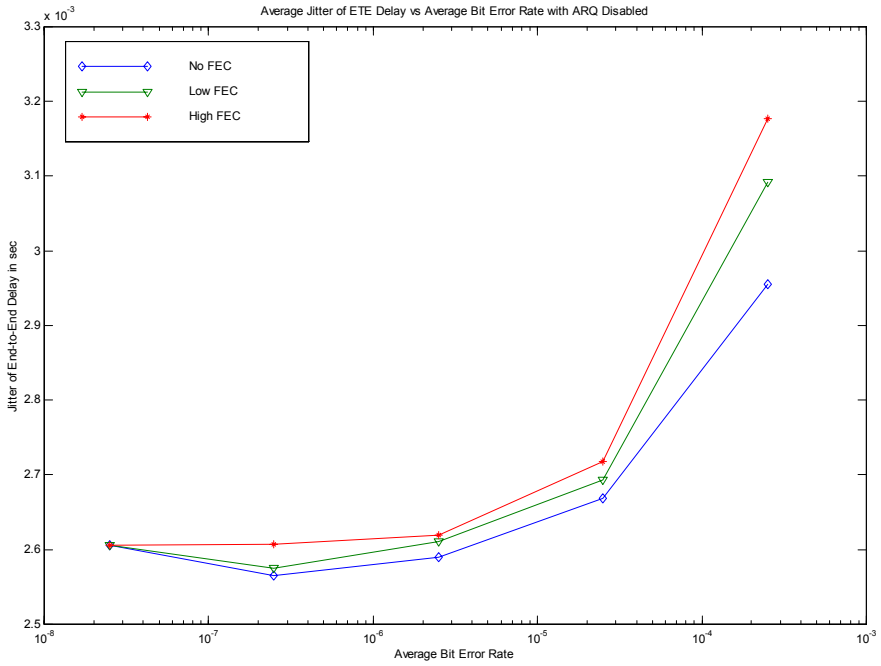

Figure E.3. Average jitter in end-to-end delay versus average bit error rate for connections not using link layer ARQ.

E.2. Figures for Connections using Link Layer ARQ

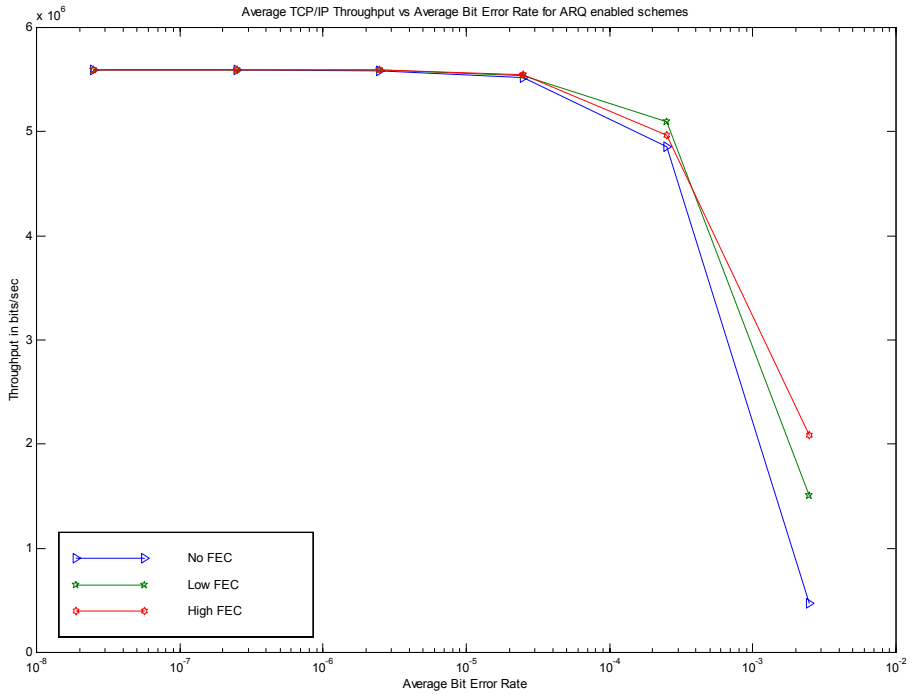

Figure E.4. Average TCP/IP throughput versus average bit error rate.

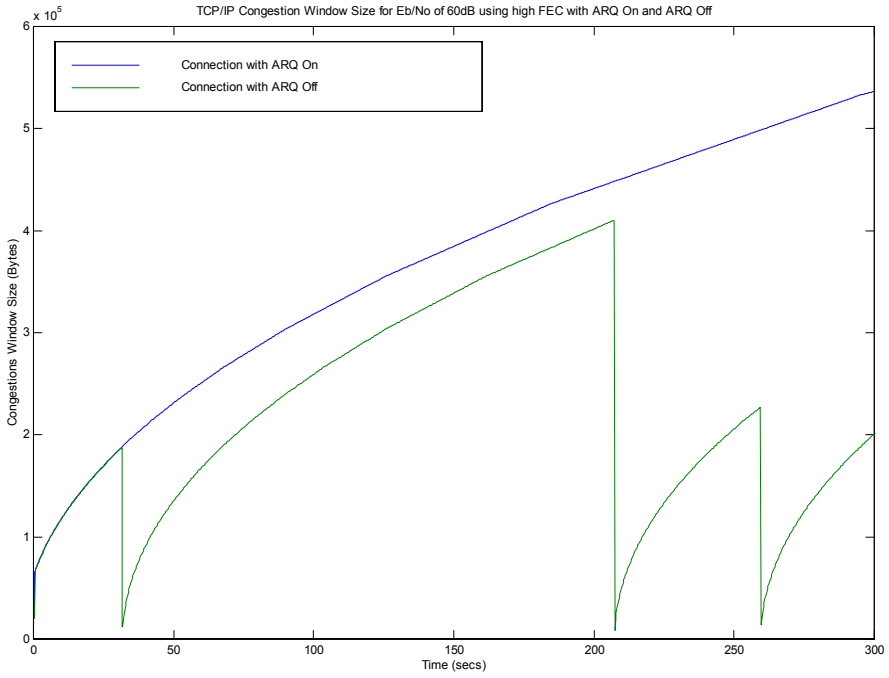

Figure E.5. Comparison of congestion window size for a TCP/IP connection with link layer ARQ and a TCP/IP connection without link layer ARQ.

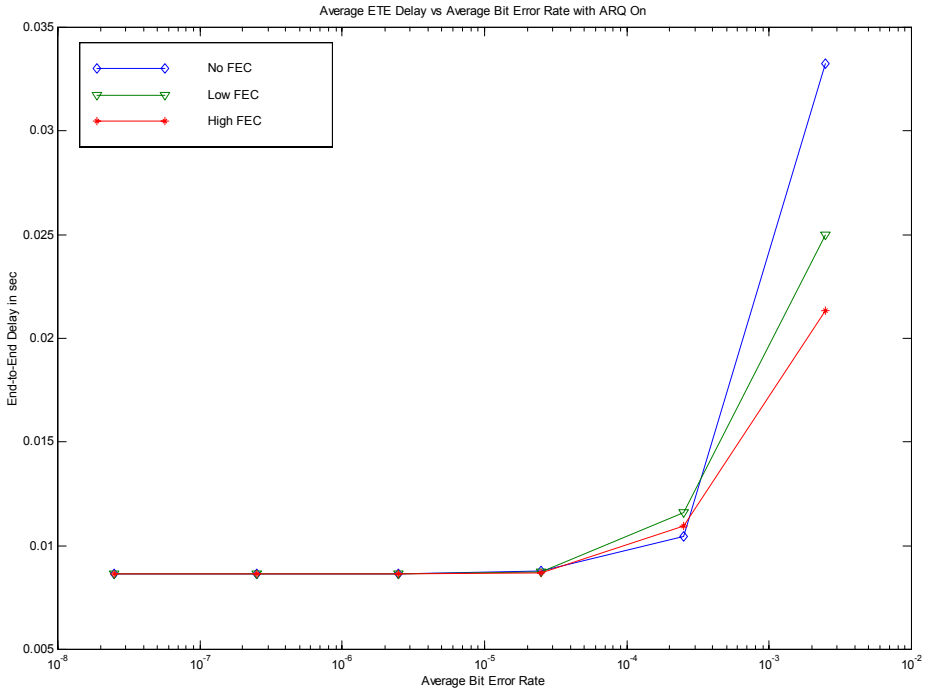

Figure E.6. Average ETE delay for TCP/IP connections as the BER increases.

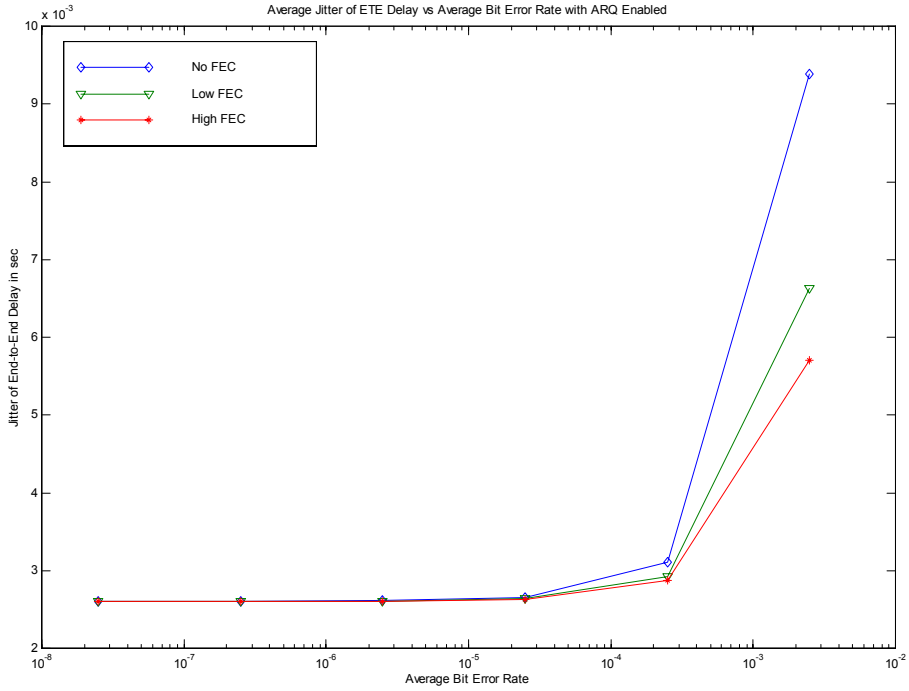

Figure E.7. Average jitter in end-to-end Delay vs. average bit error rate for connections using link layer ARQ.

E.3. Figures for CIR Algorithms

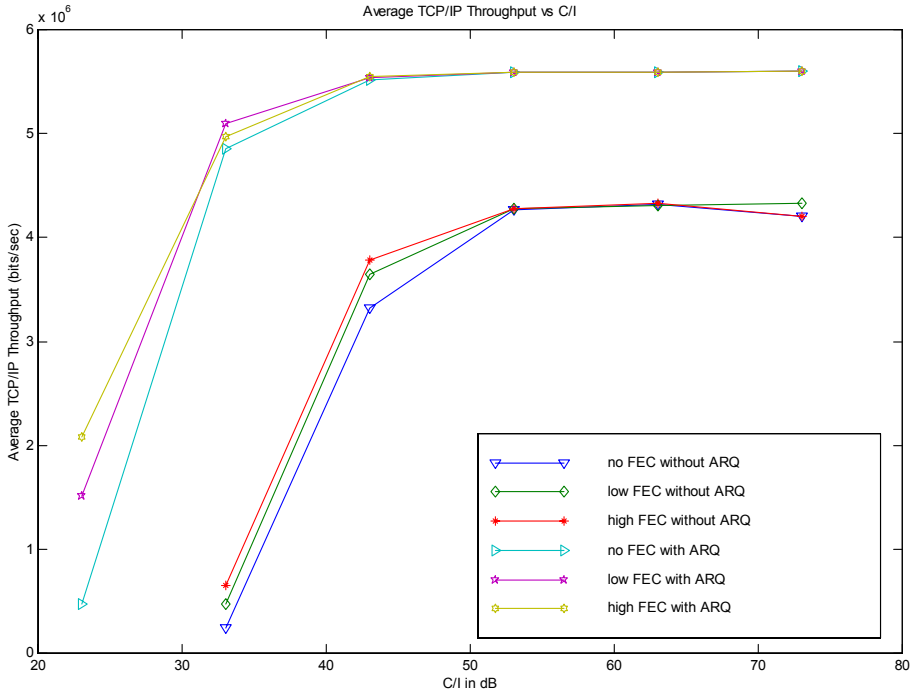

Figure E.8. Average TCP/IP throughput versus *CIR*.

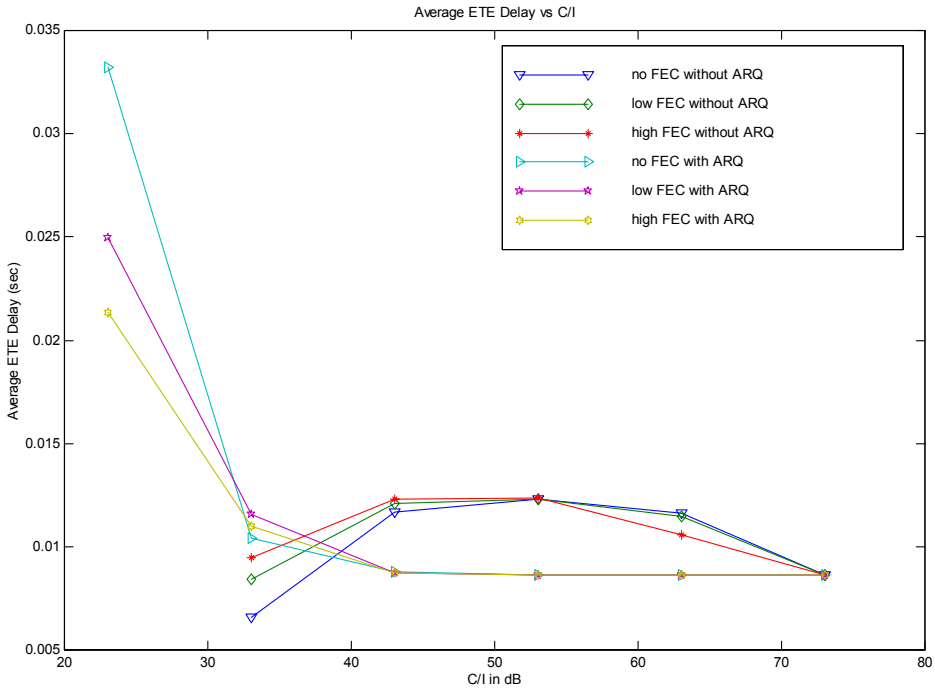

Figure E.9. Average TCP/IP ETE delay versus *CIR*

Appendix F: Tables of Results

This section presents the tables of each statistic collected from the simulation model. Each table includes the expected value, standard deviation of each run, and 95% confidence intervals for each statistic.

F.1. Average End-to-End Delay Results

Table F.1. Average TCP/IP End-to-End Delays with No FEC

Eb/No (BER)	Average End-to-End delay (sec)	Average Standard Deviation (sec)	95% Confidence Interval (sec)	ARQ scheme
20dB(2.48e-3)	N/A[3]	N/A	N/A	Disabled
20dB(2.48e-3)	3.322866e-2	1.778456e-2	±8.424913e-5	Enabled
30dB(2.49e-4)	6.575442e-3	4.323021e-3	±9.682275e-5	Disabled
30dB(2.49e-4)	1.042573e-2	4.094172e-3	±2.844562e-4	Enabled
40dB(2.50e-5)	1.166265e-2	3.667743e-3	±5.072213e-5	Disabled
40dB(2.50e-5)	8.779683e-3	2.811925e-3	±3.897845e-6	Enabled
50dB(2.50e-6)	1.232060e-2	3.221840e-3	±2.971684e-4	Disabled
50dB(2.50e-6)	8.639044e-3	2.677079e-3	±1.304135e-6	Enabled
60dB(2.50e-7)	1.159805e-2	3.314915e-3	±9.896238e-4	Disabled
60dB(2.50e-7)	8.626214e-3	2.665554e-3	±6.149250e-7	Enabled
70dB(2.50e-8)	8.624885e-3	2.663670e-3	±0	Disabled
70dB(2.50e-8)	8.625069e-3	2.663848e-3	±1.853921e-7	Enabled

Table F.2. Average TCP/IP End-to-End Delays with Low FEC

Eb/No (BER)	Average End-to-End delay (sec)	Average Standard Deviation (sec)	95% Confidence Interval (sec)	ARQ scheme
20dB(2.48e-3)	N/A	N/A	N/A	Disabled
20dB(2.48e-3)	2.496920e-2	1.210341e-2	±4.082058e-4	Enabled
30dB(2.49e-4)	8.413094e-3	4.820309e-3	±1.679297e-4	Disabled
30dB(2.49e-4)	1.157762e-2	3.985003e-3	±2.057776e-3	Enabled
40dB(2.50e-5)	1.209717e-2	3.455581e-3	±2.088957e-5	Disabled
40dB(2.50e-5)	8.727469e-3	2.758229e-3	±2.899100e-6	Enabled
50dB(2.50e-6)	1.231131e-2	3.189350e-3	±3.972292e-4	Disabled
50dB(2.50e-6)	8.633095e-3	2.671828e-3	±1.035181e-6	Enabled
60dB(2.50e-7)	1.146609e-2	3.391546e-3	±8.323243e-4	Disabled
60dB(2.50e-7)	8.625794e-3	2.664502e-3	±1.448440e-7	Enabled
70dB(2.50e-8)	8.624885e-3	2.663670e-3	±0	Disabled
70dB(2.50e-8)	8.624953e-3	2.663755e-3	±1.853921e-7	Enabled

[3] For simulations not using FEC encoding and not using link layer ARQ, simulations could not be recorded because TCP/IP Reno terminated the connection before the average value stabilized.

Table F.3. Average TCP/IP End-to-End Delays with High FEC

Eb/No (BER)	Average End-to-End delay (sec)	Average Standard Deviation (sec)	95% Confidence Interval (sec)	ARQ scheme
20dB(2.48e-3)	N/A	N/A	N/A	Disabled
20dB(2.48e-3)	2.133359e-2	1.018380e-2	±2.214097e-4	Enabled
30dB(2.49e-4)	9.441344e-3	4.947844e-3	±1.799047e-4	Disabled
30dB(2.49e-4)	1.097427e-2	3.588873e-3	±2.398355e-3	Enabled
40dB(2.50e-5)	1.229502e-2	3.432181e-3	±1.369154e-5	Disabled
40dB(2.50e-5)	8.710677e-3	2.740785e-3	±3.190589e-6	Enabled
50dB(2.50e-6)	1.233245e-2	3.226958e-3	±3.028765e-4	Disabled
50dB(2.50e-6)	8.631477e-3	2.669356e-3	±8.270014e-7	Enabled
60dB(2.50e-7)	1.055485e-2	3.376384e-3	±1.189999e-3	Disabled
60dB(2.50e-7)	8.625838e-3	2.664506e-3	±1.461757e-7	Enabled
70dB(2.50e-8)	8.624885e-3	2.663670e-3	±0	Disabled
70dB(2.50e-8)	8.624953e-3	2.663755e-3	±1.853921e-7	Enabled

F.2. Average Jitter in End-to-End Delay Results

Table F.4. Average TCP/IP Jitter in End-to-End Delay with No FEC

Eb/No (BER)	Average Jitter (sec)	Average Standard Deviation (sec)	95% Confidence Interval (sec)	ARQ scheme
20dB(2.48e-3)	N/A	N/A	N/A	Disabled
20dB(2.48e-3)	9.386218e-3	8.739116e-3	±1.002775e-4	Enabled
30dB(2.49e-4)	2.955534e-3	2.211051e-3	±1.217321e-5	Disabled
30dB(2.49e-4)	3.109532e-3	2.198131e-3	±8.179255e-6	Enabled
40dB(2.50e-5)	2.668448e-3	1.652198e-3	±1.226955e-5	Disabled
40dB(2.50e-5)	2.653893e-3	1.544440e-3	±8.999633e-7	Enabled
50dB(2.50e-6)	2.590156e-3	1.488255e-3	±3.139072e-5	Disabled
50dB(2.50e-6)	2.611123e-3	1.470744e-3	±3.181199e-7	Enabled
60dB(2.50e-7)	2.565364e-3	1.475647e-3	±3.868567e-5	Disabled
60dB(2.50e-7)	2.606756e-3	1.464099e-3	±1.990329e-7	Enabled
70dB(2.50e-8)	2.606331e-3	1.462869e-3	±5.704351e-11	Disabled
70dB(2.50e-8)	2.606385e-3	1.462959e-3	±5.394973e-8	Enabled

Table F.5. Average TCP/IP Jitter in End-to-End Delay with Low FEC

Eb/No (BER)	Average Jitter (sec)	Average Standard Deviation (sec)	95% Confidence Interval (sec)	ARQ scheme
20dB(2.48e-3)	N/A	N/A	N/A	Disabled
20dB(2.48e-3)	6.627828e-3	5.793954e-3	±2.567508e-5	Enabled
30dB(2.49e-4)	3.091655e-3	2.246574e-3	±1.645817e-5	Disabled
30dB(2.49e-4)	2.926591e-3	1.957347e-3	±3.136610e-6	Enabled
40dB(2.50e-5)	2.693498e-3	1.618262e-3	±4.417355e-6	Disabled
40dB(2.50e-5)	2.637768e-3	1.516153e-3	±7.905080e-7	Enabled
50dB(2.50e-6)	2.611220e-3	1.482484e-3	±6.685072e-6	Disabled
50dB(2.50e-6)	2.609109e-3	1.467850e-3	±4.818937e-7	Enabled
60dB(2.50e-7)	2.574576e-3	1.475554e-3	±5.400868e-5	Disabled
60dB(2.50e-7)	2.606671e-3	1.463413e-3	±9.918475e-8	Enabled
70dB(2.50e-8)	2.606331e-3	1.462869e-3	±5.704351e-11	Disabled
70dB(2.50e-8)	2.606346e-3	1.462902e-3	±2.454347e-8	Enabled

Table F.6 Average TCP/IP Jitter in End-to-End Delay with High FEC

Eb/No (BER)	Average Jitter (sec)	Average Standard Deviation (sec)	95% Confidence Interval (sec)	ARQ scheme
20dB(2.48e-3)	N/A	N/A	N/A	Disabled
20dB(2.48e-3)	5.707479e-3	4.859527e-3	±2.247656e-5	Enabled
30dB(2.49e-4)	3.176677e-3	2.195924e-3	±1.245975e-5	Disabled
30dB(2.49e-4)	2.868317e-3	1.873151e-3	±5.165209e-6	Enabled
40dB(2.50e-5)	2.717625e-3	1.602320e-3	±3.565334e-6	Disabled
40dB(2.50e-5)	2.632908e-3	1.507884e-3	±1.327023e-6	Enabled
50dB(2.50e-6)	2.618824e-3	1.477454e-3	±2.159194e-6	Disabled
50dB(2.50e-6)	2.608448e-3	1.466367e-3	±1.928643e-7	Enabled
60dB(2.50e-7)	2.607133e-3	1.463902e-3	±1.921683e-7	Disabled
60dB(2.50e-7)	2.606671e-3	1.463387e-3	±9.918477e-8	Enabled
70dB(2.50e-8)	2.606331e-3	1.462869e-3	±5.704351e-11	Disabled
70dB(2.50e-8)	2.606346e-3	1.462902e-3	±2.454347e-8	Enabled

F.3. Average TCP/IP Throughput Results

Table F.7. Average TCP/IP Throughput with No FEC

Eb/No (BER)	Average Throughput (bps)	Average Standard Deviation(bps)	95% Confidence Interval (bps)	ARQ scheme
20dB(2.48e-3)	N/A	N/A	N/A	Disabled
20dB(2.48e-3)	4.752859e+5	5.739428e+5	±1.637841e+3	Enabled
30dB(2.49e-4)	2.380530e+5	3.358418e+5	±1.366464e+3	Disabled
30dB(2.49e-4)	4.851705e+6	1.723051e+5	±1.334425e+3	Enabled
40dB(2.50e-5)	3.322885e+6	1.362489e+6	±5.362156e+4	Disabled
40dB(2.50e-5)	5.515715e+6	5.893212e+5	±9.472275e+2	Enabled
50dB(2.50e-6)	4.264533e+6	4.277659e+5	±6.218622e+3	Disabled
50dB(2.50e-6)	5.587764e+6	4.113692e+5	±3.012109e+2	Enabled
60dB(2.50e-7)	4.312450e+6	3.215146e+5	±6.152259e+3	Disabled
60dB(2.50e-7)	5.594879e+6	5.430655e+5	±1.138268e+2	Enabled
70dB(2.50e-8)	4.206235e+6	1.407010e+5	±1.193960e+3	Disabled
70dB(2.50e-8)	5.595590e+6	5.443507e+5	±0	Enabled

Table F.8. Average TCP/IP Throughput with Low FEC

Eb/No (BER)	Average Throughput (bps)	Average Standard Deviation(bps)	95% Confidence Interval (bps)	ARQ scheme
20dB(2.48e-3)	N /A	N/A	N/A	Disabled
20dB(2.48e-3)	1.509151e+6	9.836778e+5	±7.172043e+3	Enabled
30dB(2.49e-4)	4.763192e+5	5.930406e+5	±1.197736e+4	Disabled
30dB(2.49e-4)	5.092928e+6	1.520916e+5	±3.703752e+3	Enabled
40dB(2.50e-5)	3.648130e+6	1.113364e+6	±2.308677e+4	Disabled
40dB(2.50e-5)	5.542814e+6	5.730105e+5	±1.395249e+3	Enabled
50dB(2.50e-6)	4.275815e+6	4.061944e+5	±2.210106e+4	Disabled
50dB(2.50e-6)	5.591001e+6	4.112215e+5	±1.138470e+2	Enabled
60dB(2.50e-7)	4.324128e+6	3.213283e+5	±2.151890e+4	Disabled
60dB(2.50e-7)	5.595092e+6	5.424366e+5	±5.691326e+1	Enabled
70dB(2.50e-8)	4.331752e+6	1.405856e+5	±1.996438e+5	Disabled
70dB(2.50e-8)	5.595661e+6	5.436820e+5	±5.691348e+1	Enabled

Table F.9. Average TCP/IP Throughput with High FEC

Eb/No (BER)	Average Throughput (bps)	Average Standard Deviation(bps)	95% Confidence Interval (bps)	ARQ scheme
20dB(2.48e-3)	N /A	N/A	N/A	Disabled
20dB(2.48e-3)	2.085180e+6	1.003785e+6	±1.534017e+4	Enabled
30dB(2.49e-4)	6.507993e+5	7.724560e+5	±1.050059e+4	Disabled
30dB(2.49e-4)	4.966198e+6	1.362795e+5	±3.504218e+5	Enabled
40dB(2.50e-5)	3.794826e+6	8.780685e+5	±2.608512e+3	Disabled
40dB(2.50e-5)	5.551883e+6	5.724655e+5	±1.528207e+3	Enabled
50dB(2.50e-6)	4.269950e+6	3.519773e+5	±2.635951e+4	Disabled
50dB(2.50e-6)	5.592352e+6	4.118028e+5	±2.276940e+2	Enabled
60dB(2.50e-7)	4.321583e+6	3.218221e+5	±5.851746e+4	Disabled
60dB(2.50e-7)	5.595163e+6	5.427031e+5	±9.857681e+1	Enabled
70dB(2.50e-8)	4.206744e+6	1.257875e+5	±3.611640e+2	Disabled
70dB(2.50e-8)	5.595626e+6	5.435770e+5	±5.691348e+1	Enabled

F.4. Average Packet Error Rate Results

Table F.10. Average Packet Error Rate with No FEC

Eb/No (BER)	Average Packet Error Rate	Average Standard Deviation	95% Confidence Interval
20dB(2.48e-3)	7.944908e-003	9.967551e+001	±2.010236e-005
30dB(2.49e-4)	2.403867e-002	1.991149e+001	±3.142399e-005
40dB(2.50e-5)	2.398794e-003	5.717519e+001	±2.954480e-005
50dB(2.50e-6)	2.495313e-004	1.381715e+001	±9.259811e-006
60dB(2.50e-7)	2.305579e-005	4.892981e-004	±3.391411e-006
70dB(2.50e-8)	1.882384e-006	1.366961e-004	±6.034074e-007

Table F.11. Average Packet Error Rate with Low FEC

Eb/No (BER)	Average Packet Error Rate	Average Standard Deviation	95% Confidence Interval
20dB(2.48e-3)	1.147902e-002	8.062536e+001	5.089338e-005
30dB(2.49e-4)	1.577644e-002	4.239498e-003	8.751138e-005
40dB(2.50e-5)	1.582538e-003	4.660086e+001	3.954725e-005
50dB(2.50e-6)	1.534385e-004	1.082113e+001	3.872416e-006
60dB(2.50e-7)	1.707825e-005	4.241943e-004	2.196593e-006
70dB(2.50e-8)	7.531452e-007	5.051133e-005	1.205283e-006

Table F.12. Average packet error rate with high FEC

Eb/No (BER)	Average Packet Error Rate	Average Standard Deviation	95% Confidence Interval
20dB(2.48e-3)	1.142976e-002	5.171769e-003	7.156960e-005
30dB(2.49e-4)	1.284927e-002	4.367564e+001	2.063888e-005
40dB(2.50e-5)	1.307482e-003	4.235166e+001	4.410528e-005
50dB(2.50e-6)	1.123666e-004	9.241597e+000	6.293153e-006
60dB(2.50e-7)	1.558038e-005	4.057436e-004	2.534182e-006
70dB(2.50e-8)	1.129239e-006	8.618480e-005	1.043806e-006

Vita

Todd Eshler was born and raised in Richmond, Virginia. By the age of eight, he had started to learn how to program computers, and with the help of his father, he learned how repair and install hardware in PCs. In 1996, he graduated from Mills E. Godwin High School.

In September of 1996, Todd began attending Virginia Tech and working towards his Bachelor of Science in Computer Engineering. During the summers, he worked for Circuit City Stores, Inc. as a software tester and programmer. In 1998, he completed his minor in German and began a yearlong study abroad at the Technische Universität Dresden in Dresden, Germany. After graduating in June of 2000, he worked for Litton Poly Scientific as a software developer.

In September of 2000, Todd began working towards his graduate degree at Virginia Tech. He worked with other members of the Center for Wireless Telecommunications on the LMDS network for the NSF Digital Government Project.

After completing the requirements for a Master of Science in Computer Engineering, Todd will work for Digital Receiver Technology, Inc. in Germantown, Maryland as a software engineer.