An Adaptive Jam-Resistant Cross-Layer Protocol for Mobile Ad-Hoc Networks In Noisy Environments

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

Mark Gregory Kuhr

Certificate of Approval:

Richard O. Chapman Associate Professor Computer Science and Software Engineering	John A. Hamilton, Jr., Chair Professor Computer Science and Software Engineering
David A. Umphress Associate Professor Computer Science and Software Engineering	Martin C. Carlisle Professor Computer Science United States Air Force Academy

George T. Flowers
Dean
Graduate School

An Adaptive Jam-Resistant Cross-Layer Protocol for Mobile Ad-Hoc Networks In Noisy Environments

Mark Gregory Kuhr

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
May 9, 2009

An Adaptive Jam-Resistant Cross-Layer Protocol for Mobile Ad-Hoc Networks In Noisy Environments

Mark Gregory Kuhr

_____

Signature of Author

_____

Date of Graduation

Dissertation Abstract

An Adaptive Jam-Resistant Cross-Layer Protocol for Mobile Ad-Hoc Networks In Noisy Environments

Mark Gregory Kuhr

Doctor of Philosophy, May 9, 2009
(Master of Software Engineering, Auburn University, AL, 2008)
(Bachelor of Science in Wireless Engineering, Auburn University, AL, 2006)

374 Typed Pages

Directed by John A. Hamilton, Jr.

This dissertation contributes a jam-resistant communications protocol for use in a mobile ad-hoc network (MANET). Specific focus is given to the network layer routing of packets within a MANET. A MANET is a self-organizing multi-hop wireless network that is comprised of many mobile hosts communicating wirelessly. As a result of their mobility and dependence on wireless communications, network layer routing is especially prone to errors. An algorithm has been developed using the concept of concurrent codes to allow for jam-resistant wireless communications without a pre-shared secret. The algorithm allows for the recovery of messages even in the presence of significant noise. Experimental results show that a cross-layer protocol designed to take advantage of this property allows for more effective communications in noisy environments such as disaster sites where current network protocols experience significant packet loss rates. Ultimately, this research presents a communications protocol

that adapts to the level of interference in the environment to which the network is deployed to allow for effective communication.

Acknowledgments

- Thanks to my wife Laura who has always had the utmost confidence in my abilities. Thank you for your continued love and support.
- Thanks to my graduate advisor and committee members for their support, advice, and encouragement.
- Thanks to the Information Assurance Center graduate students at Auburn University for your support.
- Thanks to the US Air Force Academy Professors Bill Bahn, Leemon Baird, and Martin Carlisle for their tireless support, advice, and expertise.
- Thanks to my friend, Derek Sanders, for keeping my ideas grounded in reality.
- This work would not be possible without the continued financial support of the Department of Defense (DoD) Information Assurance Scholarship Program (IASP) who has funded me through this research.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliography follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

Chapter 1

Introduction

Mobile Ad-Hoc Networks (MANETs) consist of a number of mobile devices which self-organize to form a network without any infrastructure in place at the deployment location. Each node will communicate wirelessly with other nodes within its transmission range. Due to the lack of infrastructure, each node will act as a router and will be partially responsible for routing packets throughout the network. Formally, a MANET is the union of a set of autonomous nodes that communicate via unpredictable wireless communications to form an arbitrary communication graph. This graph, otherwise known as network topology, will vary with time as mobile nodes move in and out of range with their surrounding nodes. Essentially, these networks are peer-to-peer multi-hop wireless networks that transmit packets of information in a store-and-forward manner. In a store-and-forward scheme, a receiving node may choose to store a packet temporarily if it currently does not have any neighboring nodes within its transmission range. In the presence of neighbors, the receiving node must decide whether to forward the packet or not. The ability for this type of network to function is dependent upon the network layer protocol used to route packets among nodes in the network.

MANETs have several characteristics that affect communication throughout the network.

- **Dynamic Topologies:** Since nodes are free to move in any direction and at any speed, the network topology may change rapidly. When bidirectional communication is necessary, frequent topology changes may cause significant routing protocol overhead. Furthermore, as a result of the wireless medium, some nodes may only be able to form unidirectional links. A more detailed discussion of the issues with this dynamic topology will be covered later.

- **Power Constraints:** Most nodes in a MANET rely on batteries for power. In this case, processing messages has a direct correlation to consumption of energy. For this reason, protocol efficiency is especially important. The network protocol designed in this research dynamically adjusts power consumption in response to environmental conditions such as noise.

- **Bandwidth Limitations:** Wireless links have a significantly lower transmission rate than wired infrastructures. In addition to an already constrained capacity, the effects of wireless communication further decrease bandwidth. For example, some of these effects on wireless transmissions may include noise, interference, and fading. The term *bandwidth* also has another meaning depending on the context. Another definition for bandwidth in wireless communications concerns the frequency limitations of the wireless channel. The network protocol designed in this dissertation will attempt to use the full spectrum allowed through spread spectrum techniques, while maximizing the transmission rate.

In addition, the protocol adapts its transmissions to resist interference if necessary which may result in reduced data rate.

- **Limited Physical Security:** Most MANET nodes are susceptible to physical attack since they are deployed to remote and often unattended locations. This research will not cover physical security attacks on wireless nodes.

Due to the characteristics listed above, communications in disaster and battlefield situations is especially troublesome (Figure 1.1). In these scenarios, there is no network infrastructure to rely upon, therefore MANETs are well suited for the job. Furthermore, there may be environmental conditions or intentional jamming by an adversary that affects network availability. Current MANET designs have problems with noise because it causes significant packet loss which triggers retransmissions. However, these current designs also depend upon sensing the medium to avoid packet collisions. Packet loss due to interference and collisions affects the reliability and availability of the MANET. In an emergency situation, network availability is of the utmost importance even at the sacrifice of the data rate.

The communications protocol developed in this research is a cross-layer protocol in the sense that it will mainly operate at the network layer, yet it will manipulate data encoding and decoding at the data-link layer. Furthermore, in order to provide a working prototype, the physical layer functionality is implemented on software defined radios. A form of error correcting codes, known as concurrent codes, forms the basis for this new cross-layer protocol. As a subset of concurrent codes, this protocol

will use BBC (named after creators Baird, Bahn, and Collins) codes [Baird, Bahn and Collins 2007] which are superimposed codes that can be decoded in polynomial time. These codes can help a receiver recover messages that have been affected by noise. Noise adds energy to a signal and thus changes bits from 0's to 1's, but the use of concurrent codes allows the receiver to decode the corrupted signal up to a certain threshold. This research provides a cross-layer protocol that takes advantage of the jam-resistance provided by concurrent codes to allow MANETs to operate in environments with significant spectrum interference.

According to the Open Systems Interconnection Basic Reference Model (commonly referred to as the OSI Model), the network layer (layer 3) is responsible for the source-to-destination delivery of a packet in a network. The network layer has two major functions in the OSI model:

- **Addressing:** The network layer must assign a logical address using a scheme that allows every network device to have a universally unique address on the network. For example, an Internet Protocol (IP) address uniquely identifies one connection to the wired Internet. In a MANET, pre-configurating unique network addresses is common prior to node deployment.

- **Routing:** To accomplish end-to-end delivery of a packet within a network, the network layer must decide the most effective way to route the packet. In the traditional wired Internet, routers use extensive routing tables to route packets

efficiently. In a MANET, the route is constructed "on-the-fly" as the packet traverses the ad-hoc network.

This cross-layer protocol will mainly operate at the network layer, however it will manipulate data encoding and decoding at the data link layer to provide adaptive jam-resistance. In this way, it is considered a cross-layer protocol.

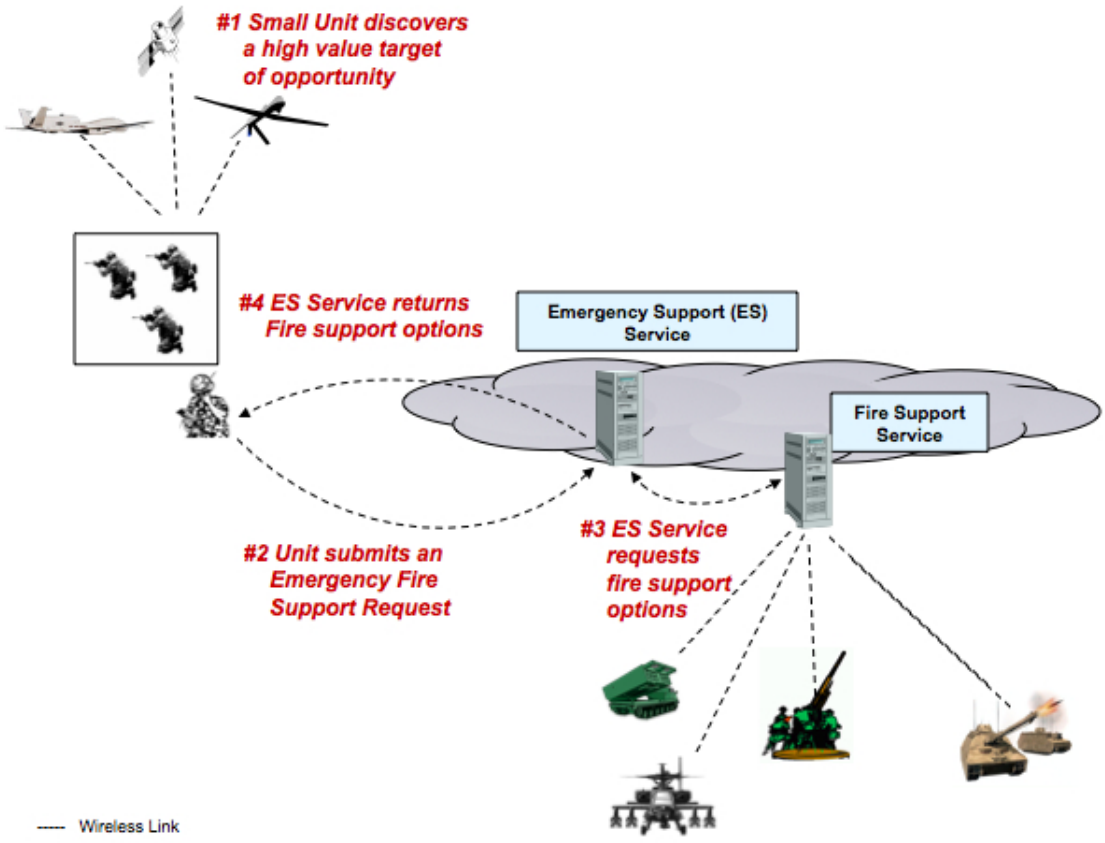

Figure 1.1: Battlefield MANET Scenario

1.1 Goals

This dissertation demonstrates the following contributions:

- A contribution to the area of address auto-configuration in MANETs. The cross-layer protocol adapts the addressing scheme to the number of nodes in the network automatically. This is validated using software defined radios and a kernel network stack.

- A contribution to the area of MANET addressing in the form of an addressing scheme that provides unique addressing in network merges. This is validated using software defined radios and a kernel network stack.

- A contribution to the area of autonomic tuning of routing in a MANET to adapt to current spectrum conditions such as the presence of noise by adjusting the routing protocol control packets to resist destruction by interference. This is validated using software defined radios.

- A contribution to the area of cross-layer cooperation in wireless communications. The network protocol will pass parameters to the data-link layer to control the encoding of packets for transmission. The cross-layer protocol will adjust signal encoding based on spectrum interference conditions to guarantee availability. This is validated using software defined radios.

- A contribution to the MANET network protocol realm in the form of a network layer that adjusts to pre-defined priorities for network performance. The

protocol should allow for the prioritization of bandwidth (data bit rate in the network) or availability which compete under the BBC encoding scheme. This is validated using software defined radios.

1.2 Challenges

The nature of MANETs make routing packets among their nodes inherently difficult. The main challenge for a network protocol is to provide a communications platform that is adaptive, dynamic, and resilient in the presence of fluctuating conditions in the wireless spectrum and node mobility. The BBC-based communications protocol presented in this dissertation advances this area of research and has met the research challenges identified below.

- The protocol must adjust the BBC message encoding in all network nodes in a top-down fashion based on feedback from the data-link and physical layers to ensure jam-resistance. Only with cross-layer support is this protocol able to adapt to changing spectrum conditions.

- The network protocol must uniquely and universally address the nodes in the network as nodes move in and out of the network. Most current auto-configuration protocols rely on periodic or reliable flooding which consumes considerable bandwidth. A BBC-enabled routing protocol is able to flood the network in a more reliable fashion due to fewer packet collisions.

- As spectrum conditions change, the protocol must optimize the network to operate under pre-set priorities. This involves a trade-off between availability and bandwidth. As availability in terms of the degree of jam-resistance increases, the bandwidth will be reduced. This tradeoff is managed dynamically and continuously throughout the entire network.

- The BBC encoding algorithm allows for jam-resistant communication, but it imposes a penalty in the form of reduced data rates. This new routing protocol limits the size of messages passed in the network to allow for continuous jam-resistant communication.

1.3 Outline

The remainder of this dissertation is organized as follows:

- Chapter 2 provides a background on Wireless Communications and the BBC Algorithm.

- Chapter 3 discusses network layer routing protocols for MANETs to include proactive, reactive, hybrid, and interference aware protocols.

- Chapter 4 discusses mobile node address auto-configuration techniques and methods for consideration.

- Chapter 5 provides the initial design for the cross-layer protocol.

- Chapter 6 describes the design and implementation of the physical, data link and network layers for the jam-resistant communications protocol.
- Chapter 7 describes how the jam-resistant protocol developed in this research solves the hidden and exposed station problems.
- Chapter 8 provides a comparison study of 802.11G to the BBC-enabled protocol.
- Chapter 9 discusses experimental results of the BBC-enabled network layer and provides a comparison to a traditional OLSR based network routing layer.
- Chapter 10 discusses the key contributions of this research to the field.
- Chapter 11 concludes with a discussion of the anticipated final contribution of this dissertation.

Chapter 2

Background on Wireless Communications

2.1 Chapter Introduction

Understanding the basics of wireless communication is essential to understanding the problems that most mobile networks face. The purpose of this section is to familiarize the reader with wireless communications techniques and the problems inherent to the wireless medium. These problems are further exacerbated by the mobility among nodes and spectrum interference. Furthermore, this chapter will discuss the BBC algorithm [Baird, Bahn and Collins 2007] in enough detail to understand its resistance to jamming and application to the cross-layer protocol described in this dissertation.

2.2 Wireless Transmission Techniques

2.2.1 Introduction

Most people are familiar with the idea of wireless communication since we use it in our everyday lives. However, few are familiar with the details of how it works and why the wireless medium is inherently lossy. In wireless systems, signals are broadcast through free space (either air or a vacuum) and are available to any receiver capable of receiving them. Consider the MANET scenario where multiple nodes are transmitting and every node within range is capable of receiving the transmission. It is for this

reason MANETs are especially prone to unintentional jamming by other nodes. The remainder of this section will provide more detail on wave propagation in free space and schemes for allowing multiple nodes to share the wireless spectrum.

First, it is important to understand a few terms common to this area of study.

Terminology

Spectrum A range of wavelengths of electromagnetic radiation.

Omnidirectional Transmitting signals in all directions.

Bandwidth A range of frequencies used for transmitting a signal or the transmission capacity of a computer network.

Interference The disturbance of received radio signals caused by unwanted signals from other sources. Also, the act of combination of two or more waves to form a resultant wave (constructive) or cancellation (destructive).

Propagation The transmission of electromagnetic waves through a particular medium.

Path Loss The average propagation loss over an area due to distance between receiver and transmitter, antenna gains, transmitter and receiver power, environmental obstacles, and carrier frequency.

2.2.2 Mobile Radio Propagation

In an ideal world, radio wave propagation in free space without obstacles is the ideal situation. However, this is only possible in a perfect vacuum. In most real-world

situations, radio waves encounter obstacles during transmission. For the purposes of this example, assume all mobile stations are using omnidirectional antennas so their transmissions radiate from the transmitter like ripples in a pond. MANETs rely on wireless communications in order to operate effectively and propagation loss represents a major reason a receiver cannot decode a transmitted message correctly. The combination of waves creates errors in the signal and as a result nodes must retransmit the message. When radio waves encounter obstacles, the following propagation effects may occur and result in path loss [Agrawal and Zeng 2006].

1. **Diffraction:** Occurs when a wave bends around an object with sharp irregular surfaces. For instance, consider the case where line of sight does not exist between a transmitter and receiver, but the receiver still receives the signal. A good example of this scenario occurs when a wave bends around a sharp corner such as the edge of a building.

2. **Reflection:** Occurs when a propagating wave encounters an object that is larger compared to its wavelength. For example, when a radio wave encounters the face of a building it will reflect off the building.

3. **Scattering:** Similar to reflection, except this occurs when a wave encounters an object that is smaller than the wavelength of the propagating wave. For instance, a radio wave will scatter into several weaker signals if it hits a street sign.

2.2.3 Multiple Access Schemes

In mobile ad-hoc networks, it is necessary to allow multiple nodes to utilize limited spectrum space simultaneously. In general, there are three ways to allow for multiple communication channels within the same allocated bandwidth to include frequency, time, and code division [Agrawal and Zeng 2006, Forouzan 2007].

- Frequency Division Multiple Access (FDMA): Each user is assigned a different carrier frequency to transmit their messages. A related approach is Orthogonal Frequency-Division Multiple Access (OFDMA) where each user is a assigned a subset of orthogonal carrier frequencies

- Time Division Multiple Access (TDMA): Each user is assigned a different time slot to transmit their messages.

- Code Division Multiple Access (CDMA): Each user is assigned a different code (also called a "chip sequence") to use to transmit their messages. Under this scheme, the assigned code spreads the signal over the entire bandwidth range (known as spread spectrum). Jam-resistance is achieved through spread-spectrum techniques since the modulated signal is spread over the entire allocated bandwidth. In order to jam a spread spectrum signal, the jammer must add power to the entire channel (requires significant resources) or must possess the spreading code [Agrawal and Zeng 2006, Forouzan 2007] to focus their efforts. Figure 2.1 demonstrates the process used in spread spectrum techniques to spread the signal.

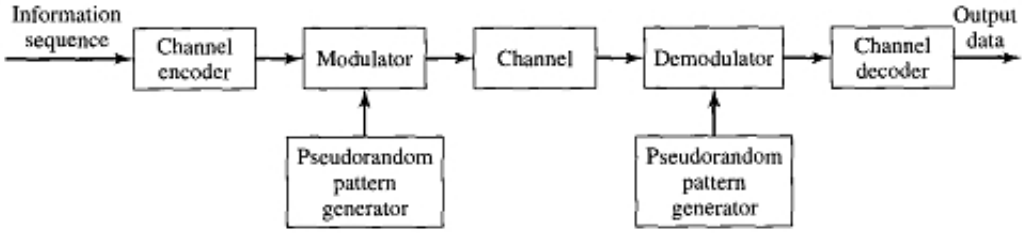

Figure 2.1: Spread Spectrum Process

In practice, there are two main types of spread spectrum techniques:

1. *Frequency Hopping Spread Spectrum (FHSS):* A pseudorandom sequence (or code) is used to spread a frequency across a frequency band. The transmitter will hop from frequency to frequency according to the assigned code. Multiple users may use this technique as long as no two users occupy the same frequency at the same instant in time, but collisions may occur if the hop sequences are not chosen carefully.

2. *Direct Sequence Spread Spectrum (DSSS):* A pseudorandom sequence (or code) directly phase modulates a carrier frequency as shown in Figure 2.2. Each data bit of the original signal is replaced with a code of n bits called chips and where the chip rate is n times that of the data bit. The resulting signal resembles noise to any receiver within range. Only the receiver that possesses the pre-shared spreading code will be able to recover the signal. Privacy and jam-resistance is only guaranteed while the chip sequence remains private. If the sequence is known, an attacker may broadcast a strong signal using that sequence to interfere with a legitimate user. For

the purposes of this research, the jam-resistant network layer protocol will assume all mobile nodes use DSSS for their transmissions as this is common in current 802.11 wireless systems [Forouzan 2007].

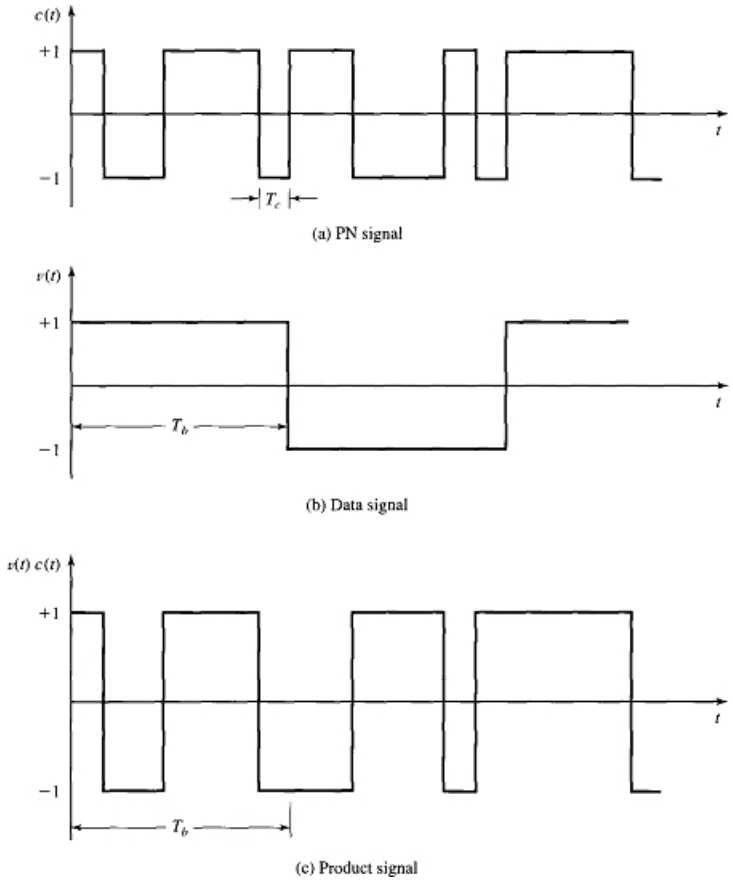

Figure 2.2: Direct Sequence Spread Spectrum (DSSS) Example

2.2.4 Signal Jamming

Mobile ad-hoc networks operate in a wireless spectrum that is highly susceptible to signal jamming. In general, this jamming is either due to interference with other nodes in the network or purposeful jamming by an adversary. In a crowded and

unregulated spectrum, the jamming can be a result of other signals propagating in the spectrum. For example, consider the unintentional jamming caused in the 2.4 GHz frequency band by kitchen microwave ovens.

- **Friendly (Unintentional) Jamming:**

  Jamming from adjacent nodes is a common problem when many nodes share the same channel. In this scenario, a mobile node receives data from more than one station at a time which results in a collision. A collision of this nature can be in the form of constructive or destructive interference that prevents the receiver from decoding any signals. It is anticipated that the protocol developed in this dissertation will compensate for these problems by allowing for concurrent messages to be decoded properly even when interference between stations exists. Two common friendly station interference scenarios are described below [Forouzan 2007].

  1. *Hidden Station Problem:* As shown below in Figure 2.3, nodes B and C are hidden from each other with respect to A. To understand this situation, consider the case where node B is sending data to node A. While this transmission is occurring, node C also sends data to node A since it does not know node B is already transmitting to A. The result is a collision at node A since it is receiving data from both B and C simultaneously.
  2. *Exposed Station Problem:* As shown in Figure 2.4, transmission ranges for each node overlap. As a result, node C is exposed to transmissions from

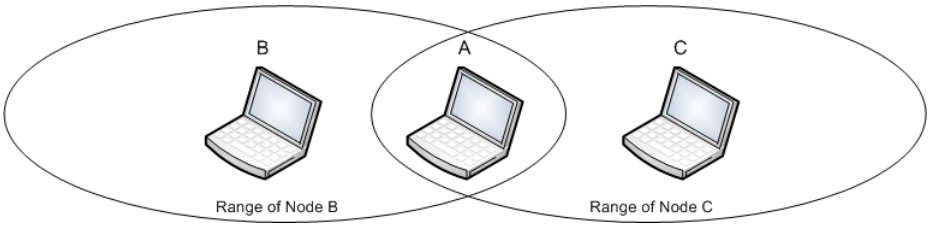

Figure 2.3: Hidden Station Problem

node A to node B. Consider the case where node A is transmitting data to node B, and node C would like to transmit data to node D. This is not a problem, except that node C is exposed to the traffic from node A and does not transmit because it senses the medium is busy. Therefore, node C wastes channel capacity and time by waiting for node A to complete its transmission to node B.

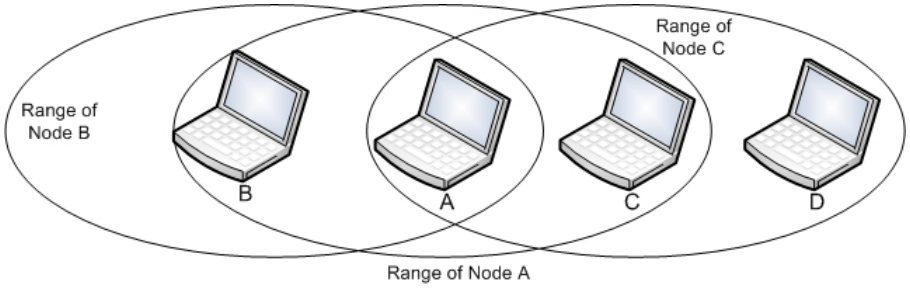

Figure 2.4: Exposed Station Problem

- **Adversarial (Intentional) Jamming:**

  Jamming from an adversary is prevalent in military communications scenarios. In addition, as hardware for jammers is fairly cheap and readily available, networks are susceptible to malicious denial-of-service attacks. For simple wireless signals, signal jamming can increase the power of a signal (flipping bits from 0 to 1) or decrease power in the signal by propagating a wave that is 180-degrees out of phase (flipping bits from 1 to 0). To combat this simple attack, spread spectrum techniques discussed in the previous section are used. However, the jam resistance of traditional spread spectrum systems depends on the use of a secret key to control the sequence of frequencies for example in FHSS. Therefore, in order to jam a spread spectrum system, an adversary possessing the secret may add power to the channel to corrupt signals.

2.3 BBC Codes

The basis for this new network protocol lies in the jam-resistance properties of the BBC algorithm[Baird, Bahn and Collins 2007]. For this reason, it is important to understand how this algorithm is able to decode signals in the presence of significant interference.

2.3.1 Introduction

Spread spectrum techniques provide some jam-resistance but the transmitter and receiver must each possess the pseudorandom code sequence. Sharing this secret

prior to communications is analogous to the use of symmetric keys in cryptography. Symmetric keys do not scale well to large systems because the compromise of one key exposes the entire system to attack. Consider the case where a single spreading code is used by every radio in a military unit. The compromise of one radio handset would then allow the enemy to jam communications for the entire unit. The creators of BBC developed the algorithm in response to this situation. The BBC algorithm allows two parties to communicate without a pre-shared secret in a jam-resistant manner. The algorithm works by defining codewords that can be combined with a bitwise OR, transmitted, and then efficiently decoded by the receiver. The remainder of this chapter will discuss details of how the algorithm operates and its resistance to jamming.

Terminology

The following lexicon is used within the BBC algorithm description and will be used in this dissertation when referring to the algorithm.

Location The position of a single bit in a bit string.

Indelible Mark A change of a bit from 0 to 1 purposefully by the encoder. An attacker cannot practically remove the mark from a radio signal and change the bit from 1 to 0.

Data The bits that form the payload of a message to be transmitted.

Message The bits of the data plus the header, padding, and checksum bits.

Codeword The encoder takes the message and produces a codeword.

Packet The combination of one or more codewords with a bitwise OR. The packet is the same length as a codeword.

Expansion Factor The ratio of the codeword length to the data length.

The algorithm functions in two modes: encoding and decoding. During the encoding stage, the algorithm takes in a binary string to be encoded and maps it to the proper transmission buckets. The bucket is used for conceptualization, and could be represented as specific frequencies, timeslots, or any other distinguishable radio transmission. The following sections describe the jam-resistant properties of BBC, and then shows a simple pulse-based broadcast using BBC to encode then decode a message.

2.3.2 Theoretical Background

In the realm of coding theory, there are currently two main types of codes including *error detecting codes* and *error correcting codes* [Forouzan 2007, Ayanoglu et al. 1993]. The authors of [Baird, Bahn and Collins 2007] propose a new family of codes known as *concurrent codes* to which BBC belongs. BBC codes belong to a larger family codes known as *superimposed codes*. In short, a concurrent code is a superimposed code that can be decoded in polynomial time. Furthermore, a concurrent code translates each message into a binary codeword. The idea of concurrency comes from the ability to combine several codewords with a bitwise OR to form a single

combined string. Then, the receiver can analyze the combined string and recover all of the original codewords and messages.

As shown in [Baird, Bahn and Collins 2007], the equations below describe the relationships among the maximum number of simultaneous messages, the bit rate, and the number of expected hallucinations. The network layer protocol described in this dissertation will make use of these relationships to adapt the MANET to changing spectrum conditions to optimize the balance between bit rate and degree of jam-resistance.

Notation

M_s = number of messages intentionally put into a packet by senders.

$R = \frac{M_S}{e}$ = the bit rate ratio of the maximum message bits per second for simultaneous broadcast divided by the maximum bits per second for ordinary sequential broadcast.

$e = \frac{c}{m}$ = codeword expansion.

$h = M_R - M_S$ = hallucination rate (expected number of hallucinations per packet). A hallucination is an unintended message that the receiver accidentally decodes but it is not authentic since it was not sent by any sender.

k = the number of 0 bits appended to the message as a checksum.

m = bit length of a message.

u_p = the probability that a packet bit is 1 (mark density)

$$M_S = \frac{-1}{lg(1 - \frac{m - lg(h)}{me})} \tag{2.1}$$

$$R = \frac{M_S}{e} \tag{2.2}$$

$$k = em(1 - (1 - u_p)^{\frac{1}{M_S}}) - m \tag{2.3}$$

2.3.3 BBC Encoding

During encoding step, the input binary string is appended with k checksum bits that are actually just 0's. The number of checksum bits, k, is determined in advance. Each bit string prefix is then sent through a hash function that maps a bit string to a bucket. The buckets that are identified by the hash function are considered marked and will be transmitted as a 1. In this example, the buckets identified with marks will be transmitted as a pulse. Using this example described in [Baird, Bahn and Collins 2007] which uses 25 buckets and 2 checksum bits, the encoding of the message M where M = 1011 proceeds as follows:

1. Append two checksum zeros to M: 1011**00**.

2. Encode each prefix string, s, using the hash function, H as shown in Table 2.1.

s	**H(s)**
1	21
10	9
101	24
1011	2
10110	14
101100	12

Table 2.1: Prefix Hash Table

3. Transmit a "1" at the corresponding bucket locations for the *H(s)* results as demonstrated in Table 2.2 below.

Bucket	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1011	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0

Table 2.2: Transmission Buckets

2.3.4 BBC Decoding

During decoding, the receiver is required to do more work than the encoder. The receiver will begin with a message that looks like Table 2.2 and needs to decode the message based on the specific buckets that received a 1 during transmission. Since it is unknown whether the message began with a 0 or a 1, the receiver must hash both and determine whether a 1 was transmitted in that bucket. The decoding phase proceeds as follows:

1. Determine whether a 0 or 1 was transmitted. H(0) = 4 and H(1) = 21, and so the receiver will listen at buckets 4 and 21. Given the message from Table 2.2, it is seen that no transmission occurs at bucket 4; however, one occurs at bucket 21 indicating the message begins with a 1.

2. Next, the current string prefix is appended with a 0 and 1 account for each possible prefix. H(1**0**) = 9 and H(1**1**) = 21 in which case both of these have a message transmission, indicating there is the possibility for two encoded messages.

3. Again, a 1 and 0 is appended to the string prefixes that are still alive. H(10**0**) = 20, H(10**1**) = 24, H(11**0**) = 16, and H(11**1**) = 2. Of these buckets, only buckets 24 and 2 have transmissions and will continue to be decoded.

4. Append a 1 and 0 to the string prefixes that are still alive. H(101**0**) = 15, H(101**1**) = 2, H(111**0**) = 14, and H(111**1**) = 23. Only buckets 2 and 14 have transmissions and continue to be decoded.

5. At this point the length of the transmitted message (4-bits) has been reached. Thus, the surviving prefixes will be appended with k-bits (only 0-bits) for at most two times since k = 2. H(1011**0**) = 14 and H(1110**0**) = 13. Only bucket 14 has a transmission and continues to be decoded.

6. Finally, the last stage in the decoding process is reached as we append the last k-bit to the message. H(10110**0**) = 12, and bucket 12 does indeed have a transmission. Stripping the two checksum k-bits from the received message yield $M' = 1011$ which is the same as the message that the transmitter sent.

Notice that during step 5 of the decoding process, the original message can be seen but there is a second surviving message. This is considered a *hallucination* as termed by [Baird, Bahn and Collins 2007]. The k checksum bits help eliminate hallucinations in the received messages and as a result play a vital role in the decoding process. Without the checksum bits, the algorithm would have determined that multiple messages had been sent instead of the fact that the transmitter only sent one message. This example demonstrated the basic functionality of the BBC algorithm. If noise had been present in the system, more bits would have been received as 1-bits and the decoding process would have eliminated more hallucinations.

2.4 Chapter Conclusion

This chapter covered the basics of wireless communication technology and provided an overview of the BBC algorithm. In addition, this chapter discussed several major problems that arise with wireless nodes in a MANET such as the hidden station and exposed station problems. A BBC-based cross-layer protocol developed in this research allows inter-node communication in environments with significant noise and is less susceptible to the aforementioned problems.

Chapter 3

Mobile Ad Hoc Network Routing

3.1 Chapter Introduction

As discussed in Chapter 1, the routing of data through a MANET is a challenging task due to several factors that include the lack of fixed infrastructure, the dynamic topology caused by node mobility, the power constraints, and the limited bandwidth resulting from wireless communications. Consequently, the choice of routing protocol at the network layer is especially important to maximize the use of limited resources such as power and bandwidth. For the purposes of this research, all nodes in this MANET will operate using the same routing protocol and thus will use the same metrics to make routing decisions. With each node acting as a router, route computation in MANETs is distributed and made on a hop-to-hop basis. A major challenge to this approach is that in order to route a packet, a node must know which nodes are within transmission range (called *neighbor nodes*). Moreover, in a MANET with high mobility, these neighbor nodes will change rapidly [Jubin and Truong 1987]. Further complicating the issue is the number of nodes in the network. As the number of nodes in the network increases, the number of potential destination also increases proportionally. With vast numbers of mobile nodes, the amount of data that must be exchanged also increases since nodes will need to exchange route information including routes, route updates, or routing tables (*scalability*). With limited bandwidth

and power, a routing protocol should strive to minimize the number of transmissions. In addition, increasing the number of transmissions further increases packet collisions and interference among nodes which will degrade network performance. Many routing protocols rely on broadcasting to discover routes and this usually results in what is known as the *broadcast storm problem* [Tseng, Ni, Chen and Shieu 2002] since this causes contention, collision and redundancy. The BBC-based network routing layer described in this research will not suffer from interference issues (to a certain upper bound) and will operate in highly noisy environments where current routing protocols fail.

Before embarking on the development of a new adaptive cross-layer routing protocol, it is important to understand the many routing protocols that already exist. This section will provide an overview of several of the most popular protocols and demonstrate the originality of the adaptive cross-layer protocol developed in this research. There are literally hundreds of network routing protocols for MANETs, but they fall into several major categories [Royer and Toh 1999]. First, this chapter will discuss proactive protocols where nodes continuously maintain the status of routes in the network. Secondly, this chapter will provide an overview of reactive protocols which only discover routes on-demand when data needs to be sent. Next, the chapter will cover hybrid protocols that attempt to combine the best attributes of both proactive and reactive protocols. Lastly, this chapter will introduce a newer category called adaptive protocols that adapt to changing conditions in the network. The protocol

developed in this research fits into the category of adaptive protocols since it adapts to spectrum interference.

3.2 Proactive Routing Protocols

Proactive protocols are also commonly referred to as table-driven protocols since they store some amount of routing information in tables. It is the goal of these types of protocols to propagate network topology updates to the entire network so each node has the same view of the network. The following section will discuss various proactive routing protocols and highlight their key characteristics.

- **Optimized Link State Routing (OLSR):**

  OLSR version 2 (OLSRv2) [Clausen, Dearlove and Jacquet 2008] represents an update to OLSR version 1 as of July 2008. The main idea behind this protocol is to reduce the number of transmissions required for control traffic by efficiently flooding selected nodes. Each node selects a set of its neighbor nodes to be "MultiPoint Relays" (MPRs). As control traffic floods the network, a node will only forward traffic directly received from a node that selected it as an MPR. For example, if node A selected node B as an MPR, then node B will only forward control traffic directly received from A. "MPR flooding" decreases the load of control traffic on the MANET and thus improves performance. Furthermore, a node selects MPRs from a one-hop neighbor that is connected via a bidirectional

link. As a result, all routes through MPRs are bidirectional routes instead of unidirectional.

- **Standard Link State Routing (SLS):**

  In SLS [Santivanez, Mcdonald, Stavrakakis and Ramanathan 2002], each node simply sends a Link State Update (LSU) packet whenever a change in link state occurs. In the absence of changes, the node will send an LSU on a predetermined timer. This algorithm has been almost completely superseded by the optimized link state routing (OLSR) algorithm.

- **Destination Sequenced Distance Vector (DSDV):**

  DSDV [Perkins and Bhagwat 1994] is a table-driven routing protocol based on the Bellman-Ford [Cormen, Leiserson, Rivest and Stein 2001] routing algorithm which is designed to find the shortest path between two nodes in a communications graph. In this protocol, each node maintains a full routing table with a route to every possible destination in the known network. In the table, routes are marked by sequence numbers which are assigned by the destination node. The sequence numbers server two main purposes. First, a sequence number will help expose stale routes when a higher sequence number supersedes it. Second, the sequence number will help prevent the formation of routing loops in the table.

  Routing table consistency in the presence of mobility is maintained through the use of periodic network broadcasts containing the table updates. To prevent

mobility and route updates from overwhelming the network, the nodes keep track of a "settling" time during which a better route may be discovered. The nodes then delay their update broadcast by the settling time in the hope of receiving a more stable route. In addition, when an update does occur, the protocol determines which type of update packet to send: a full dump packet or an incremental update packet. To prevent unnecessary load on the network, incremental update packets are most often used if possible since they are smaller in size.

- **Cluster Head Gateway Switch Routing (CGSR):**

  First, one must understand that CGSR [Chiang, Wu, Liu and Gerla 1997] uses DSDV as an underlying protocol. CGSR forms a hierarchical network topology (DSDV forms a flat topology) where Cluster Heads (CHs) control a group of nodes. A node is classified as either an internal node, a cluster head, or a gateway node. The assignment to clusters and designation of the cluster head is completed through a distributed algorithm. The distributed algorithm adds overhead to the network and operates poorly in high mobility situations. CGSR only allows cluster heads to change when two cluster heads come into the same cluster or a node moves out of range of any cluster head. This algorithm is coined the Least Cluster Change (LCC) algorithm, but it still results in frequent cluster head changes in the presence of high mobility. CGSR does have some interesting properties in that a cluster head is able to control dynamic channel scheduling

within nodes in the cluster. For example, the cluster head can assigned different CDMA codes to each node to reduce interference.

Cluster membership is maintained through a cluster member table. Along with this table, each node maintains a DV-routing table which contains next-hop routing information. To distinguish stale routes and stale member tables, CGSR uses sequence numbers as described in DSDV. Routing is conducted in the same way as DSDV, except route selection is based on the nearest cluster head to a destination node. Then, the routing proceeds through gateway and cluster head nodes. If an internal node wishes to send a packet to a destination node in another cluster, the packet is sent to the internal node's cluster head first. Then, this cluster head selects a gateway to send the packet through to the next best cluster head. This process repeats until the destination node's cluster head receives the packet and delivers it to the final destination.

- **Wireless Routing Protocol (WRP):**

  In WRP [Murthy and Garcia-Luna-Aceves 1996], each node must maintain four tables that include the routing table, the distance table, the link-cost table, and the message retransmission list (MRL) table. Update messages are used to update neighboring nodes of link changes. The MRL table keeps track of which updates should be retransmitted and which nodes must acknowledge the transmission. With the four tables at its disposal, the node must determine routing information. Since nodes must acknowledge receipt of update messages,

WRP provides faster route convergence than SLS, Bellman-Ford, and DSDV. However, maintaining four state tables is costly and the algorithm requires significant amounts of communication to operate.

- **Topology Dissemination based on Reverse-Path Forwarding (TBRPF):**

  TBRPF [Ogier, Templin and Lewis 2004] is composed of two independent modules: the neighbor discovery module (TND) and the routing module. The routing module performs topology discovery and route computation, while the TND discovers neighbor nodes. However, the protocol as a whole aims to provide the shortest path to each destination through a modification of Dijkstra's algorithm. Each node computes a source tree based on partial topology information that provides paths to all reachable nodes. To reduce overhead, each node reports only a portion of its source tree to reachable neighbors. However, in cases of high mobility, nodes have the option to report more of their source tree to neighboring nodes. A major feature of TBRPF is that each module uses "differential" update messages which only report changes in the status of neighbors. As a result, the message size is on average much smaller than other protocols for status update or routing update messages. In addition, since the modules are independent, developers are free to use a different neighbor discovery protocol or routing protocol.

3.3 Reactive Routing Protocols

Reactive routing protocols are commonly referred to as "on-demand" protocols since they start route determination procedures when there is a need to transmit a message. It is the goal of these protocols to save energy and bandwidth by not transmitting control traffic periodically as proactive protocols do. Unfortunately, the global search initiated by a route discovery protocol usually requires significant control traffic. In addition, messages will be delayed until the route discovery process is complete.

- **Plain Flooding (PF):**

  In the classical PF algorithm, each packet is forwarded by every node in the network except for the final destination. This approach requires significant bandwidth and will result in many messages traversing the network before the final destination is reached. With every node retransmitting to all neighbors, this approach requires $N-1$ transmissions for each packet and can lead to the broadcast storm problem [Tseng, Ni, Chen and Shieu 2002].

- **Ad Hoc On-Demand Distance Vector (AODV):**

  The AODV protocol [Castenada and Das 1999, Perkins, Belding-Royer and Das 2002] is a distance vector protocol based on the DSDV algorithm described above. The main difference is that AODV uses on-demand route acquisition and a more efficient way to communicate route information. It still selects routes based on the distance to a specified destination. For example, only nodes

along a particular route will participate in the routing table changes. Moreover, each node will only store information about the routes in which they actively participate. In comparison to DSDV, AODV uses fewer broadcasts and does not require every node to store the complete routing table.

The route discovery aspect of AODV relies on the use of route request (RREQ) packets. First, the source will broadcast a RREQ to all of its neighbors, who will forward it to all of their neighbors and so on. As this RREQ traverses the network, each intermediate node records the address from which they first received a copy of the broadcast packet. In this way, a reverse path is formed from intermediate nodes back to the source. The packet will propagate the network until the final destination is reached or an intermediate node is found which has a fresh route to the destination node. Since each intermediate node recorded a reverse path, the route reply (RREP) packet follows the reverse path to the source from the destination. Due to its use of the reverse paths, AODV only supports symmetric bi-directional links. Stale routes are eliminated through the use of sequence numbers and a route timer. In this protocol, every node has a sequence number and a higher sequence number identifies a more current route. Furthermore, link failures are propagated using RREP packets. Overall, AODV avoids the Bellman-Ford "count to infinity" problem [Cormen et al. 2001] and offers quick convergence on new routes [Perkins, Belding-Royer and Das 2002]. A drawback to this protocol is that it does not scale well to large numbers of nodes [Aron and Gupta 2001].

- **Dynamic Source Routing (DSR):**

  DSR [Johnson and Maltz 1996] is based on source routing where every node in the network maintains a cache of all source routes they are aware of in the network. The protocol operates in two phases: route discovery and route maintenance. When a source does not have a reliable route to a destination, the route discovery procedure is started and a route request (RREQ) packet is broadcast to all immediate neighbors. In DSR, the route request packet contains a source address, destination address, and a unique identification number. At each hop, nodes determine whether or not they have a route to the destination address. If not, they add their own address to the route record of the route request packet and rebroadcast it to their immediate neighbors. The main purpose of the route record is to prevent nodes from retransmitting route request packets they have already analyzed, thus reducing control traffic overhead. Finally, once the RREQ packet reaches the destination, the destination sends a reply packet with the route record which allows the packet to follow a step-by-step breadcrumb trail back to the source. Prior to reaching the destination node, If an intermediate node has a route to the destination, this node simply copies its cached record into the route reply packet for the same breadcrumb trail effect. In the example just described, symmetric links are supported. However, if symmetric links were not supported by the network, the destination would have to initiate a route discovery packet to find the source and include the route record in the

new RREQ packet. The second phase of DSR is the route maintenance operation. The protocol uses route-error packets and acknowledgments to clean up routing cache entries which results in quick route convergence[Boukerche 2001]. Upon notification from the data link layer that a link is no longer available, the protocol will immediately transmit route-error packets to the route initiator. DSR has been proven to not scale efficiently due to its use of blind broadcasts in the route discovery process [Aron and Gupta 2001, Santivanez, Mcdonald, Stavrakakis and Ramanathan 2002].

- **Dynamic MANET On-demand (DYMO):**

  The DYMO routing protocol is a reactive multihop protocol for unicast routing [Chakeres and Perkins 2008] and its a direct descendant from DSR and AODV. By design, it is built to handle frequent topology changes through the use of sequence numbers to identify stale routes. The protocol consists of two basic operations that include route discovery and route maintenance. During route discovery, the protocol uses Route Request (RREQ) and Route Reply (RREP) packets to establish bi-directional links. To reduce congestion, the route discovery mechanism uses a binary exponential backoff time in cases where a RREP is not received within a specified time limit. Through the use of sequence numbers, a node decides whether to forward the packet or update the route cache by analyzing the sequence number in the packet (RREQ or RREP). The sequence number in the cache is used to rate the routes as stale,

loop-possible, inferior, or superior. Furthermore, the sequence numbers prevent routing loops [Perkins and Belding-Royer 1999]. These classifications are the basis for this protocol's ability to handle frequent topology changes. In addition, the route maintenance mechanism uses a route error (RERR) packet to identify missing or invalid routes.

- **Associativity-Based Routing (ABR):**

  ABR [Toh 1997] strives to discover longer-lived routes through the use of a metric called the degree of association stability. The degree of association stability is the idea of measuring the connection stability between two nodes over time and space. As a result, nodes with high mobility will have a low degree of association stability. On the other hand, nodes with low mobility will have a high degree of association stability. New routes are selected based on this metric to prefer more stable routes. Each node in this protocol broadcasts a beacon packet to let neighbors know it is in range. When a node receives a beacon packet, it increases the associativity tick (incremental counter) for the specified node. With this metric, the protocol consists of three phases that include route discovery, route reconstruction (RRC), and route deletion. The route discovery phase consists of a broadcast query (BQ) packet that traverses the network in a similar way to the Route Request (RREQ) packets previously mentioned. However, at each node relaying the BQ, the node appends their address and associativity ticks with neighbors to the BQ message. At the destination, the

best route is selected by examining the associativity ticks of all nodes along the path. Using this best route, the destination sends a reply packet to the source. When a node along a path moves, the RRC process begins which just causes the BQ messages to be sent to find new routes if the source node has moved. In the case a downstream (closer to the sink) node has moved, then a route notification (RN) message is used to erase stale route entries. Lastly, the route deletion phase is used to eliminate routes from the table that a source node no longer needs. The route deletion is a full broadcast to all nodes since the source may not be aware of intermediate nodes that do not exist in the path after the RRC phase. It is the stated goal of this protocol to improve network throughput by using more stable links for transmission, however this protocol does not account for the breakage of a link after the routing path decision has been made [Chin, Judge, Williams and Kermode 2002]. In effect, it would have to start from the beginning in order to compensate for node mobility that moved nodes out of range faster than the transmission time.

- **Signal Stability-Based Routing (SSR):**

  Similar to ABR, the SSR protocol [Dube 1997] selects routes based on the signal strength between the nodes and the mobility of the node. The protocol selects routes that operate over strong channels with sufficiently strong signal strength [Chlamtac and Lerner 1986]. A good way to look at the SSR protocol is to look at its subcomponents which consist of the dynamic routing protocol (DRP) and

the static routing protocol (SRP). First, the DRP protocol maintains a table for signal stability (SST) and a routing table (RT). The SST keeps track of signal strengths of the links to neighboring nodes. Depending on implementation, the channel may be recorded with a quantitative or qualitative metric. For this explanation, consider a qualitative metric is used to classify a channel as either "strong" or "weak" about some threshold value. The DRP manages all transmissions since it has the signal strength table, and it will update the routing table as well. The SRP will process any received packets that DRP passes it. The main role of SRP is to process data packets and determine whether to forward or deliver the packet to the next layer in the stack based on the routing table. In this protocol, packets are only transmitted over strong channels and therefore link reliability increases. However, the protocol will experience delays in the presence of high mobility since it must still flood the network to find routes and it is very selective about these routes based on signal strength parameters.

- **On-Demand Multipath Routing:**

  On-demand multipath routing [Nasipuri and Das 1999] is an improvement upon DSR (sometimes called MDSR) to allow for multiple paths from source to destination. More improvements on DSR include the reduction in the query flooding frequency used to discover new routes. In addition, it improves performance by giving intermediate nodes alternative routes as well. This protocol begins in the

same way as DSR by flooding the network with route discovery query messages. As the message traverses the network, each hop is added to its header. When the packet finally arrives at the destination, the destination simply copies the path from the query packet and sends the message back. As with DSR, each node maintains a route cache which stores complete routes learned through the route reply packets. To provide nodes with multiple paths, the destination node replies to several received query packets from a source route that is link-wise disjoint from the primary source route. Essentially, this provides intermediate nodes with an alternative path to the destination.

- **Ad Hoc On-Demand Vector-Backing Routing (AODV-BR):**

  AODV-BR is an offshoot of AODV but provides multiple paths for redundancy as opposed to a single path [Lee and Gerla 2000]. As with AODV, the protocol consists of two phases including Route Construction and Route Maintenance. The main difference is that during the route construction phase each node prepares an alternate route table. To form the alternate route table, a node will listen for route reply packets not directed toward it, but sent by one of its neighbors on the primary route. Now when the primary route is not available, the packets will have an alternate route to send data through. The combined primary and alternate routing tables form a mesh structure within the network and allow for highly redundant communications paths so data can be routed around broken links.

- **Split Multipath Routing (SMR):**

  The SMR protocol [Lee and Gerla 2001] strives to construct maximally disjoint routes between a source and destination node. Once these disjoint routes are established, the protocol will split traffic among the various routes to ease congestion and use the limited resources efficiently. Of course, these routes may not be the same length, but they do build in some redundancy. As with other protocols, SMR consists of route discovery and route maintenance phases. Route discovery proceeds in the same way as DSR and AODV, and the destination node will respond to the source using the path recorded in the request packet. However, in SMR, the destination node sends the response packet and then will wait to receive more route request packets for a certain amount of time. After this time, it will select another route that is maximally disjoint to the source it just responded to and sends a route response packet to this maximally disjoint source. Given a choice of many disjoint routes, the destination node chooses the one with the shortest path length by default. The route maintenance phase is quite similar to previous protocols as well, except the alternate route will be used prior to starting the route discovery process again. When both the primary and alternate routes have been disconnected, the route discovery process will start again. A major benefit to this protocol is the ability to send traffic on multiple routes to ease congestion.

- **Caching and Multipath Routing Protocol (CHAMP):**

  Another multicast protocol is the CHAMP protocol [Valera, Seah and Rao 2003] which aims to reduce packet loss due to link disruption using a temporal based scheme. Every node keeps a cache of data packets that pass through the node. Therefore, when a downstream node encounters an error, an upstream node is still capable of retransmitting the packet when it receives the error notifications. In addition, each node must maintain a route cache that contains forwarding information to every active destination and a route request cache that stores recent route requests that were processed. The route cache contains the data elements such as the destination identifier, the distance to the destination, the set of next hop nodes for the destination, the time each successor node was last used and the number of times each successor node is used. The caches play a major role in the two phases of the protocol.

  CHAMP has two phases that include the route discovery and the data forwarding. During route discovery, the protocol builds a directed acyclic graph (DAG) rooted at the source node in a similar fashion to the Temporally-Ordered Routing Algorithm (TORA) protocol [Park and Corson 2001]. As is common in these types of protocols, the RREQ packet traverses the network through intermediate nodes. Using CHAMP though, each intermediate node increments a *forward count* field in the message. This forward count allows the intermediate nodes to determine their hop count from the source. When a destination sends a route reply packet, it selects which nodes can accept the reply packet

to become part of the route based on the path the request packet took and it resets the forward count (hop count). A node listed in the route reply packet will accept the route if it is the shortest route to the destination or an existing route is too old as determined by a timer threshold. After a node accepts the route, it will forward the packet to upstream nodes with an updated forward count. This process will repeat until the route reply packet reaches the source node.

In the second phase of CHAMP, data packets are forwarded through the established routes. The data packets are stored in the data cache and forwarded on a hop-by-hop basis. If a node has multiple routes to the destination, it will send data on the routes in a *round robin* fashion. As previously mentioned, a major benefit to this protocol is its ability to cache data packets. In the event of errors, upstream nodes can use the cached data packet to transmit the packet through a different route. The authors claim that CHAMP outperforms AODV and DSR using a five packet data cache [Valera, Seah and Rao 2003] due to this redundancy and ability to use multiple routes in the event of link failures instead of starting route discovery again. This protocol proves that the ability to cache packets is a suitable approach to countering link failures.

- **Neighbor-Table-Based Multipath Routing (NTBMR):**

  The NTBMR protocol [Yao, Ma and Cao 2003] is designed to compensate for frequent topology changes due to mobility by using multipath routing. Unlike

SMR, the routes do not need to be disjoint routes. In this protocol, each node maintains a neighbor table and a route cache. In addition to the formation of the neighbor table and route cache, NTBMR also requires route discovery and route maintenance phases. To keep a current neighbor table, nodes periodically broadcast beacon packets which contain a time-to-live (TTL) field. The beacon packets are only transmitted to two-hop neighbors. The protocol allows for two approaches to manage the neighbor table that include a time driven and data driven approach. In the time driven approach, when a node receives a beacon packet along a specific route, it assumes the route is active and adds the beacon packet's information to its neighbor table. However, if the node does not receive a beacon packet along the route before the timer expires, the route is considered dormant and deleted. The timer driven approach has obvious limitations since the nodes cannot learn about changes that are two-hops away without a transmission. Instead, it would be better to use a data-driven approach which notifies neighbors of the unreachable station using a beacon packet that tells neighbors which station to remove explicitly.

In the route discovery phase, the source first consults its route cache to find a route to the destination. The route cache contains the list of routes that the station is familiar with as obtained from routing control packets. In the case where there are multiple routes available, the source picks a route based on several parameters that include route setup time, route distance, and route extraction reason. The route *extraction reason* represents the reason the route

cache has stored the route. In order of priority, the route cache prefers to obtain routes from the reply packets rather than data packets. In the event there is no suitable route, the route discovery process proceeds in much the same way as DSR. During the route maintenance phase, alternate routes are preferred over starting a new route discovery process. This protocol has a better packet delivery ratio and decreased end-to-end delay as compared to DSR [Yao, Jiang, Fan, Cao and Li 2003]. However, this protocol does use beacon packets which cause congestion and add overhead.

3.4 Hybrid Routing Protocols

Hybrid routing protocols combine aspects of both reactive and proactive protocols. In most cases, hybrid protocols initiate route discovery in an on-demand fashion but try to limit the cost of the path search through the network.

- **Zone Routing Protocol (ZRP):**

  In the ZRP protocol [Haas and Pearlman 1998, Haas, Pearlman and Samar 2002*c*], the network is divided up into routing zones. More specifically, a node's routing zone is defined as the set of nodes whose minimum distance in hops is no greater than the zone radius. The protocol strives to limit the proactive route discovery to a node's local area or zone. However, it performs reactive route discovery for routing data between zones. First, it is important to understand that the ZRP is highly adaptable to various network scenarios. The main

parameter to configure in ZRP is the routing zone radius. Large routing zone work best when mobility is low and there are many nodes communicating since these nodes would be proactively managed. On the other hand, smaller routing zones work best for high mobility situations involving few nodes. In extreme circumstances of either case, the protocol will adjust to being entirely proactive (large fixed topology) or entirely reactive (very high mobility). However, in most common network scenarios, ZRP performs better than a purely reactive or purely proactive protocol [Haas, Pearlman and Samar 2002*c*].

A node identifies a routing zone by identifying its one-hop neighbors. In ZRP, nodes use the neighbor discovery protocol (NDP) [Clausen, Dearlove and Dean 2008] to discover neighbors. Then, to maintain routes, ZRP uses two proactive protocols. First, the intra-zone routing protocol (IARP) [Haas, Pearlman and Samar 2002*b*] is used to maintain routes within a zone. For reference, IARP is a distance vector protocol customize for use in ZRP. The other proactive protocol used is the inter-zone routing protocol (IERP) [Haas, Pearlman and Samar 2002*a*] which is used for routing to nodes beyond the current zone. The inter-zone communication is handled in a reactive manner through a normal query-response scheme as discussed previously. For example, when a source needs to contact a destination that is in a different zone, the source initiates a route query packet. This route query packet is sent to the neighbor nodes first. If the route is still unknown, these nodes add their unique ID to the query packet and then forward the packet to a border node. The border node will

reactively search for the destination until it is found. Finally, the destination will observe the reverse path of nodes added to the packet and respond with a reply message. The source may receive multiple routes and will choose the best route for a given situation based on pre-determined parameters such as hop count or traffic metrics.

- **Temporally-Ordered Routing Algorithm (TORA):**

  TORA [Park and Corson 1997; 2001] is a complex distributed routing protocol based on the concept of link reversal, rather than being distance-vector or link-state based routing. The main idea behind TORA is to decouple the generation of control messages that propagate the network from the dynamics of the network topology. In addition, the algorithm allows for multipath routing. This algorithm also contains a blend between proactive and reactive routing, so it is a hybrid approach to routing. TORA is proactive in the sense that multiple routing options are available when a link fails. It also proactively optimizes the routes. However, it can be considered reactive since route creation is done in an on-demand fashion. The following paragraphs will highlight the major features of this algorithm.

  First, it is important to understand how the multipath operation works within TORA at a high-level. Its design has been proven to decrease the amount of control packets necessary in the presence of high mobility nodes [Park and Corson 2001]. To understand TORA, consider the concept of a pin ball machine

on an incline. From the ball drop location (source), there are multiple paths to the destination (sink), hence multiple paths. When TORA starts up, it assigns a height to every node in range. As with the pin ball machine, a lower height value is closer to the sink since the balls (representing data flow) flow down hill. When a path becomes jammed in the game (due to user control), the ball will flow back uphill and come down a different path. In a similar way, TORA allows for link reversal to route data around broken or jammed links to the sink. Unfortunately, the path selected may not be the shortest path and most often it is longer since it routes data around failed nodes. In addition, when nodes use the protocol, they run a separate copy of TORA for each possible destination. As a result, nodes running TORA may have different heights and directions for different destinations. In other words, each node maintains a separate directed acyclic graph (DAG) to every destination [Park and Corson 2001]. In order to simplify this explanation, consider a specific instance of TORA running on a specific node for a specific destination. This will all become clear in the following paragraphs which go into more detail.

As mentioned earlier, TORA assigns a height to their neighboring routers. In addition, it also assigns a direction to all links as a result of this height. The direction is either upstream or downstream. A link is an upstream link for the "lower" neighboring router, but it may also be a downstream link for a "higher" neighboring router due to its assigned height metric. To add meaning to this designation, an upstream link for a node would imply that data flows to the

corresponding destination node can only come to this node from the upstream node. Downstream flows work in a similar way but in the reverse direction.

At its core, TORA has four main phases that include route creation, route maintenance, route erasure, and route optimization [Park and Corson 2001]. In response to the need for a route to a specific destination, the source will broadcast a query (QRY) packet containing the destination address as part of the route creation. The route query will propagate the network until the final destination is reached or an intermediate node is reached which has a route to the destination. Once this node is found, it will respond with an update (UPD) packet which contains its height with respect to the destination. This height value is based on path length it has to the destination. For example, in the case the query (QRY) packet reaches the destination, it will respond with a "0" height to destination. As the UPD packet proceeds back through the network, each node that receives it sets its own height to a value greater than that of its neighbor who sent it the update packet. So, once the update packet reaches the initiator, there is a series of directed links from the node that sent the QRY packet to the node that sent the UPD packet. In route maintenance, when a node finds a route to a destination that is no longer valid, the node adjusts its height to be higher than its neighbors so that it may find another route to the destination. When this height adjustment occurs, an update packet is broadcast to all neighbors notifying them of the change. If none of the neighbors possesses a route to the destination, TORA restarts the route creation procedure. In the

next aspect of this protocol, the route erasure procedure simply sets a node's height to *null* and sets links to undirected as a result of network partitions. A clear (CLR) packet is used to broadcast route erasures which erase invalid routes from the network. Finally, the route optimization aspect of TORA is a proactive route discovery algorithm which uses optimization (OPT) packets to create shorter path length routes to specific destinations. The OPT packet traverses the network in a similar fashion to the UPD packets, and updates the height of other nodes with respect to the specific destination node [Park and Corson 2001].

TORA presents some interesting features which make it suitable for lossy environments, however it has some critical drawbacks. The protocol works on top of the Internet MANET encapsulation protocol (IMEP) which provides reliable, in-order delivery of routing messages from a node to some set of neighbor nodes [Corson and Park 1997]. IMEP attempts to reduce overhead by grouping several TORA and IMEP control messages (referred to as objects) together into a a single message (referred to as an object block) prior to transmission. Each of the blocks are identified with a unique sequence number and the blocks are transmitted on a periodic basis until a specified limit is reached or all necessary acknowledgments have been received. The idea behind this is to reduce the overall message volume by combining multiple messages into one, however it still faces the same problems with message collision and interference as other protocols. Another issue with TORA is that the height metric is dependent

on the logical time of the links failure. The algorithm makes the assumption that all nodes are time-synchronized with each other [Park and Corson 2001]. Unfortunately, this really is not possible without a central source of timing information such as a network time server or a global positioning system (GPS).

- **Hazy Sighted Link State (HSLS):**

  The HSLS protocol [Santivanez and Ramanathan 2003, Santivanez, Ramanathan and Stavrakakis 2001] strives to blend the best features of proactive and reactive routing schemes to limit link state updates in time and space. HSLS reduces overhead by limiting the nodes the link state updates are transmitted to and by limiting the time between successive link status information updates. The basis for HSLS comes from the family of Fuzzy Sighted Link State (FSLS) protocols where the frequency of link state updates propagated to distant nodes is reduced based on the concept that distant nodes do not have much immediate impact in the nearby hop-by-hop routing decision. Therefore, FSLS protocols often queue link state updates and transmit them at the same time on a periodic interval. HSLS is an optimized version of FSLS so that total overhead is globally minimized. Furthermore, HSLS strikes a balance between refresh periods and distances so that the probability of making a suboptimal next hop decision is the same for every destination in the network. Simulations show a significant reduction of routing packets transmitted while maintaining throughput and packet delay metrics [Koltsidas, Dimitriadis and Pavlidou 2004].

- **Fisheye State Routing (FSR):**

  The FSR protocol [Iwata, Chiang, Pei, Gerla and Chen 1999] aims to reduce congestion in large networks through the use of multilevel fisheye scopes. In a similar way to HSLS, nodes using FSR exchange link-state entries with neighbors more frequently depending on the distance to a destination. It operates with the concept that nearby network changes are more important than distant network changes. Using the fisheye idea, changes in nearby nodes are seen with higher resolution more frequently, while distant nodes are seen less frequently with lower resolution. Due to its reduction in network wide topology updates, this protocol has been proven a scalable solution in MANETs. However, it does still suffer the problems with interference as other protocols and may not operate optimally in the presence of high mobility nodes depending on the network size [Pei, Gerla and wei Chen 2000]. A similar protocol is the LANMAR protocol [Pei, Gerla and Hong 2000] which combines FSR with landmark routing where a landmark is a set of nodes that move together as a group. This is very similar to the ZRP protocol [Haas, Pearlman and Samar 2002*c*] and will not be discussed further.

3.5 Interference Aware Routing Protocols

In MANETs, packets transmitted by a node are often received by many nodes that are nearby. As a result of the vast amount of wireless transmissions, signal interference may occur. Currently, there is not a unified definition for interference

since it depends on many factors such as the propagation loss model for the signal. Interference is basically the combination of signals received by the same node simultaneously which prevents the receiver from decoding valid signals. Since valid signals are disrupted, interference results in packet collisions which triggers numerous redundant retransmissions of the same packets. The retransmissions consume energy and decrease throughput in the network.

Many have studied interference within MANETs recently including [Krunz, Muqattash and Lee 2004], [Tan and Seah 2005], [Tang, Xue, Chandler and Zhang 2005], [Burkhart, Rickenbach, Wattenhofer and Zollinger 2004], and [Johansson and Carr-Motyckova 2005].

1. [Krunz, Muqattash and Lee 2004] computes the potential interference of receivers and adjusts node transmission power to reduce the probability of interfering with other nodes in the network.

2. [Tan and Seah 2005] proposes a scheme to adjust transmission range to avoid interference with non-neighbor nodes. The scheme relies on broadcasting "hello" packets to find the minimum transmission distance to the neighbor nodes and then adjust the power of the transmitter accordingly. However, since it uses broadcasted "hello" packets, the overhead for this scheme is significant.

3. [Tang, Xue, Chandler and Zhang 2005] proposes methods to calculate link and path interference. Then, based on these values, the authors propose a single

path routing algorithm based on limiting interference through the use of directional antennas.

4. [Burkhart, Rickenbach, Wattenhofer and Zollinger 2004] and [Johansson and Carr-Motyckova 2005] devise a way to minimize interference using topology control to reduce power while maintaining connectivity. The topology control methods define the interference as the coverage of a link $e = (u, v)$ to be the cardinality of the set of nodes covered by the disks induced by the range of nodes u and v.

- **Average Link Interference-Aware Routing (ALIR):**

  In the ALIR protocol [Zhang, Liu, Shi, Liu and Yu 2007], the authors present explicit definition for interference since there is not a unified definition of interference. From this definition, the authors derive formulas for computing node-interference (3.1), link-interference (3.2), and path-interference (3.3).

$$I(u) = N_1 + \alpha N_2 + \beta N_3 \tag{3.1}$$

  where $I(u)$ is the interference of node u, N_1 is the number of nodes in the dark shaded region of Figure 3.1, N_2 is the number of nodes in the lighter shaded region of Figure 3.1, and N_3 is the number of nodes in the yellow colored region of Figure 3.1. Also, α and β are less than 1 and represent the weights of N_2 and N_3 respectively.

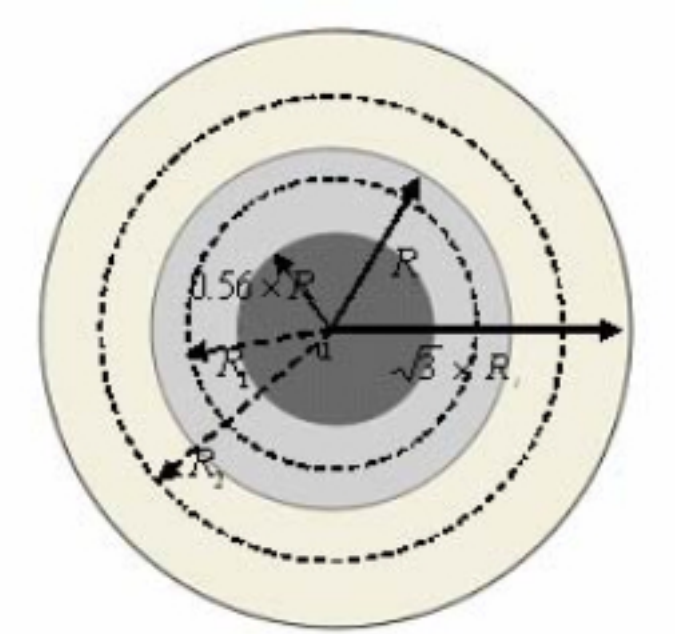

Figure 3.1: ALIR's Computation of Interference [Zhang et al. 2007]

$$I(e) = \frac{I(u) + I(v)}{2} \tag{3.2}$$

where $I(u)$ and $I(v)$ represent the node interference of nodes u and v respectively.

[Zhang, Liu, Shi, Liu and Yu 2007] define the path interference along a path as shown below. Since the path interference is dependent on the length of the path, the authors choose to define path-interference as the average link-interference of a path divided by the total number of links in the path.

$$I(P) = I(e_1) + I(e_2) + \cdots + I(e_n)$$
$$Metric(P) = \frac{I(P)}{n} \tag{3.3}$$

where $Metric(P)$ is the path-interference and n is the total number of links in the path.

The concept behind ALIR is to use DSR as a base protocol and then compute interference metrics for links. From these interference metrics, ALIR will route packets along the path with the least interference. This usually results in a path around a troubled network area at the cost of an increased path length. ALIR is shown to operate at higher throughput than DSR due to its reduction in packet collisions and retransmissions resulting from interference [Zhang, Liu, Shi, Liu and Yu 2007].

3.6 Chapter Conclusion

The contents of this chapter should provide the reader with significant background on the various types of MANET routing protocols currently in use today. First, this chapter explored proactive protocols where nodes continuously maintain routes in the network by broadcasting control packets. In an alternative approach, this chapter presented several reactive protocols that attempt to reduce control packet broadcasts by discovering routes only when necessary. As a combination of these

approaches, several hybrid protocols are discussed which attempt to find the best combination of proactive and reactive routing techniques. Lastly, this chapter presented some work related to routing in the presence of interference, however none of the approaches covered resemble the approach taken in this dissertation.

Chapter 4

Mobile Ad Hoc Network Address Auto-Configuration

4.1 Chapter Introduction

Most MANET research is focused on the development of efficient routing protocols. Most neglect the issue of addressing and assume that all nodes have a unique address in the network. This assumption works fine in the rare case that nodes are pre-configured prior to deployment and no other nodes join the network after deployment. The tasks of dynamically and automatically assigning unique addresses to every node in the MANET remains an open research problem [Weniger and Zitterbart 2004, Jeong, Oh, Kim, Park, Kim and Toh 2007]. There have been numerous attempts to solve this problem, and they are categorized below into IP-address and non-IP address assignment.

4.2 IP-Based Approaches

The main benefit to assigning unique IP addresses to nodes in a MANET comes from the ability to connect to a gateway node which has a connection to the Internet. However, most MANETs that operate in remote regions or in disaster scenarios do not require this capability. The following is an enumeration of several common approaches to this problem:

1. Automatic Configuration of IPv6 Addresses for Nodes in a MANET with Multiple Gateways [Ruffino and Stupar 2006] proposes a stateless method to automatically configure IPv6 addresses for MANET nodes in the presence of multiple gateways that are connected to the Internet backbone. It provides a mechanism for optimal address prefix-selection to improve routing. Similar approaches using Internet gateways are found in [Cha, Park and Kim 2003, Jelger, Noel and Frey 2004, Templin, Russert and Chakeres 2006, Lee, Yoo, Kang, Kim and Kang 2006, Hofmann 2006].

2. IP address Autoconfiguration for Ad Hoc Networks [Fazio, Villari and Puliafito 2006] uses random address selection to pick an IPv4 or IPv6 address. Then, it uses an address request-reply protocol to verify uniqueness of the selected address. A similar approach is taken in [Ros and Ruiz 2006].

3. IPv6 Autoconfiguration in Large Scale Mobile Ad-Hoc Networks [Weniger and Zitterbart 2002] automatically assigns IPv6 addresses in MANETs using an algorithm that elects "leader nodes" to assign addresses to a set of hosts. This approach allows most nodes to receive addresses in a way similar to DHCP. Duplicate address detection is still necessary using this protocol though. Similar approaches are taken by [Fazio, Palazzi, Das and Gerla 2006, Nesargi and Prakash 2002, Zhou, Ni and Mutka 2003].

4. Ad Hoc IP Address Autoconfiguration [Jeong, Park, Kim, Jeong and Kim 2006] assigns IPv4 addresses automatically by combining the IP address with a generated key to guarantee uniqueness in a form of weak duplicate address detection (DAD) to be discussed later.

5. IP Address Assignment in a Mobile Ad Hoc Network [Mohsin and Prakash 2002] makes every node a DHCP server that is capable of assigning addresses, then nodes exchange address lists to keep the network synchronized. A similar approach is taken in [Kim, Kim, Yu, Song and Mah 2007] and [Tayal and Patnaik 2004] where each node is given a unique set of addresses which they can assign to other nodes.

6. Address Autoconfiguration in Optimized Link State Routing Protocol [Adjih, Boudjit, Jacquet, Laouiti and Muhlethaler 2005] relies upon control messages to make sure self-assigned addresses do not conflict with other nodes on the network. This approach requires that nodes exchange their addresses for comparison. A similar approach is taken in [Sun and Belding-Royer 2003, Clausen and Baccelli 2005, Mase and Adjih 2006].

7. Global Connectivity for IPv6 Mobile Ad Hoc Networks [Wang, Li, Hwang and Chen 2005] uses the local address of the node as the 64-bit interface ID in IPv6.

8. Passive Autoconfiguration for Mobile Ad Hoc Networks (PACMAN) [Weniger 2005] assigns each node a compressed version of an IP address so the address

size is smaller than a full IPv4 or IPv6 address. The address space grows with the number of nodes in the network.

4.3 Non-IP-Based Approaches

The focus of this research is on MANET communication in a jam-resistant manner to compensate for noisy environments such as disaster zones or battlefields; therefore, the need to connect to external gateways is not necessarily a priority. Furthermore, since jam-resistant message encoding adds overhead to the communication, the use of full length IP addresses adds significant unnecessary overhead. For example, IPv4 uses 32-bit addresses which provides 4,294,967,296 unique addresses while IPv6 uses 128-bit addresses which provides 2^{128} unique addresses. It is difficult to imagine the need for a MANET to address this many nodes at one time. Consequently, the preferred solution for MANETs that will be used in this dissertation is to use a non-IP based addressing scheme to improve efficiency and eliminate overhead. As presented in [Boleng 2002], efficient network layer addressing is possible with non-IP addressing since the wasted address space is reduced to a minimal level. If connections to gateway nodes are necessary to allow communication with the broader Internet at some later time, then a protocol such as Network Address Translation (NAT) can be used at the gateway.

4.4 Duplicate Address Detection

Duplicate address detection is a necessary service when auto-configuration protocols may result in address collision. Consider the birthday paradox as an analogy to represent when two networks merge. If these two networks use the same auto-addressing scheme, there is a high probability that two nodes will have the same address and thus create an address collision. For this reason, auto-addressing protocols usually include duplicate address detection as shown in [Jeong, Oh, Kim, Park, Kim and Toh 2007], [Vaidya 2002], [Weniger and Mase 2006] and [Weniger 2003]. These protocols usually add overhead to the network since duplicate addresses must be resolved in some way. In other words, when there is an address conflict, one node must give up its address and acquire a new one. For this reason, it is better to reduce the number of statistically probable address collisions during the initial address generation phase of the auto-configuration protocol. The network layer described in this research will provide an efficient network layer auto-addressing scheme which reduces address collisions to a statistically negligible chance.

4.5 Chapter Conclusion

This chapter focused on techniques for providing nodes in a MANET with unique network-layer logical addresses. In self-organizing networks, the capability to deploy

nodes to a region without pre-configuring the addresses is quite powerful. Furthermore, dynamic addressing schemes support MANET merges on-the-fly since unique addresses can be negotiated post deployment.

Chapter 5

Adaptive Jam-Resistant Cross-Layer Protocol Initial Design

5.1 Chapter Introduction

The jam-resistant cross-layer protocol uses a combination of several disciplines to achieve its goal of providing adaptive jam-resistant communications in noisy environments where other protocols simply fail. These disciplines include wireless communications, coding theory for signal encoding (BBC concurrent codes), MANET network layer routing, and finally MANET network layer addressing. This protocol is considered to be cross-layer since it involves signal encoding which is a data-link layer function. By managing the jam-resistance parameters at the network layer, the network is more effective as a whole. In this research, a new data-link layer is created to provide the necessary feedback to the network layer.

5.2 Network Layer Routing

The initial protocol design calls for a hybrid protocol which combines proactive and reactive routing techniques. In this initial design, the protocol handles routing proactively. As an optimization, the protocol combines multiple control messages into a single message which is encoded using concurrent codes as described in BBC [Baird, Bahn and Collins 2007]. For example, when a routing table needs to be exchanged, these entries are encoded into a single message for transmission simultaneously and

the receiving nodes will decode a single message to receive the entire routing table. The reactive portion of the protocol adapts message encoding throughout the network to increase and decrease the level of jam-resistance that nodes are using. This requires broadcasting to notify nodes of the change, but flooding does not result in as many packet collisions as traditional protocols generate since the decoder is able to decode messages that have been corrupted. The reactive nature of this protocol improves throughput in the absence of jamming by reducing the level of jam resistance, but decreases throughput for the sake of availability in the presence of jamming by increasing the level of jam resistance.

In addition to the routing characteristics, the protocol allows the following network operations:

1. *Network Initiate:* Sets up the network initially after node deployment and starts address auto-configuration.

2. *Network Combine:* Allow two compatible MANETs to combine if so desired by a member of one of the networks. This does not automatically occur in case two MANETs want to move past each other and switch operating areas.

At a smaller scale, the protocol will allow the following node-level operations:

1. *Node Join:* Allows a compatible node to join an existing MANET. This join procedure may occur as a result of a node coming back into range after link failure. When a node joins, it has to generate an address that is not already in use.

2. *Node Separate:* Allows nodes to separate from the MANET. This occurs automatically due to link failure. If a node is out of contact for more than an acceptable time threshold, it is removed from the viable node list in the routing table. When it rejoins the network, the node join operation begins.

5.3 Network Layer Addressing

The basic concept of variable length addressing comes from [Boleng 2002]. In variable length addressing, there is a balancing act between the minimum address length and the need to reconfigure all nodes in the network when the address space runs out. For this reason, it is important to choose a large enough minimum address length during network initialization in order to prevent unnecessary transmissions as the network grows. Another important aspect of variable addressing is the scale of address space increase when the network does grow. For example, if the network starts with 16 nodes, technically only 4 bits are required for the addresses. However, when a 17th node wants to join the network, the addressing protocol must determine by how much to increase the address space. A logical approach is to use multiples of two, so all nodes would then increase their address size to 8 bits.

This protocol also auto-configures addresses in the network. The auto-configuration protocol allows a node to generate an address for itself where the chances of address collision are statistically negligible. This approach involves the use of a random number generator to generate the unique addresses.

5.4 Dynamic Jam-Resistance

Dynamic jam-resistance is a crucial aspect of this protocol. Adapting the BBC algorithm [Baird, Bahn and Collins 2007] to handle levels of jam-resistance is required by this feature. Altering the length of the packet will allow the protocol to adjust the degree of jam-resistance. For example, extending the packet length should increase the degree of jam-resistance. Conversely, decreasing the packet length should decrease the degree of jam-resistance and allow for more efficient messaging to take place. This feature requires modifications to the BBC encoding and decoding algorithms discussed in Chapter 2, Section 2.3.

5.5 Chapter Conclusion

The Adaptive Jam-Resistant Cross-Layer MANET protocol is designed to leverage knowledge from the wireless communications, coding theory, and computer networking areas of study. It is meant to solve many of the problems that plague MANETs today to including friendly interference caused by the hidden station and exposed station problems. Furthermore, it is built to resist intentional jamming and adapt to several levels of spectrum interference. Jam-resistance comes at a cost in throughput and latency, but using a simplified routing algorithm and an efficient variable addressing scheme should compensate. The next chapter details the final design and implementation of this communications protocol. The final protocol provides network availability in noisy environments where other protocols simply fail.

Chapter 6

Protocol Design and Implementation Phase

6.1 Chapter Introduction

During the protocol design and implementation phase, the various system components are assembled into a working prototype that is used for testing the research ideas described in this dissertation. There are many interconnected pieces to the overall system and it is important to understand how they will interact in the prototype to fully understand the complexity of this system. With a mix of hardware and software components, the problem space is multiplied as hardware limitations may arise and software bugs may affect hardware outputs. In this phase, a divide-and-conquer approach is taken to this rather large and complex communications system. In this approach, the overall system is divided into smaller problems that can be implemented and tested independently. Once the component is fully tested and verifiably correct, the component will be integrated into the overall system.

At the beginning of this research, only a basic physical layer implementation existed for the BBC algorithm [Baird, Bahn and Collins 2007]. The implementation simply took some text, encoded it with BBC, and broadcasted it continuously using a software radio. Another software radio would run the receiver script on the software radio and decode the message that was transmitted. This physical layer implementation was an initial prototype built by the creators of the BBC algorithm. This

provided a good foundation for this research, but the lack of higher-level layers in the form of a protocol stack precludes its use for a unicast communications system. Therefore, to build a MANET communications system based on the BBC algorithm required the implementation of physical, data-link and network layers, as well as a suitable application layer for testing purposes.

From an implementation standpoint, the implementation of multiple layers provides a significant challenge since the entire protocol stack from the physical to the application layer must be constructed. The focus of this research is on network layer routing in the presence of interference, but in order to build a working prototype that operates on real radios instead of a simulator, the implementation of the other layers proved absolutely necessary. The purpose of this chapter is to familiarize the reader with the design and implementation phases of this research in addition to demonstrating the level of effort involved in building this system. The remaining sections will discuss the system components, the implementation of the three lower layers, and provide a discussion of the software architecture of this prototype with applicable diagrams.

6.2 System Components

The prototype system developed during the course of this research is composed of both hardware and software components. In the following section, the specific hardware and software components that come together in this prototype will be discussed in detail. An important relationship to realize is that the chosen hardware does affect

the design of the software since the software drives the software defined radios (SDRs) in this prototype. The specific details concerning the interaction between the software and hardware components will be discussed in the software architecture section.

6.2.1 Hardware Components

- **Computer:** By far the most important hardware component in this system is the computer which drives the Universal Software Radio Peripheral (USRP) software radio device. In this case, the computer used is an Apple Macbook Pro with the relevant specifications of an Intel Core 2 Duo 2.5GHz processor, 4GB of RAM, and two USB 2.0 ports. The USB ports are critical to this prototype since any communication with the software radios takes place over the USB ports. The speed of the USB ports also affects the signal throughput on the USRP which will be discussed in detail later. The actual computer is not necessarily important to this system, except that it must be able to run the required software and operate at USB 2.0 speeds over the bus. The specific requirements for the software will be discussed in the next section, but this prototype is a cross-platform solution so any operating system on the driver computer should work without substantial modifications.

- **Universal Software Radio Peripheral (USRP):**

  The USRP represents the core of the software radio platform. This hardware device is closely coupled with the GNU Radio software framework which allows the hardware to be manipulated by software. GNU Radio communicates with

the USRP via a USB 2.0 interface with speeds approaching 32 Mb/s. The USB 2.0 specification allows for speeds up to 480Mb/s, but this is not currently possible with the FPGA. The radio signal processing is handled on the USRP while the commands to the hardware pass through the USB. In order to fully understand some aspects of this research, it is important to understand some details of the USRP circuitry. The USRP board is controlled by an Altera Cyclone Field Programmable Gate Array (FPGA) which has a customizable firmware written in the Verilog programming language. The USRP also has four 64 Mega-Sample (MS)/s 12-bit analog to digital converters (ADC) for receiving and four 128MS/s 14-bit digital to analog converters (DAC) for transmitting. The FPGA drives these converters that are provided by the AD9862 chip. The AD9862 chip is a mixed signal processor with dual receive and dual transmit channels. The AD9862 also contains the decimation and interpolation filters for the receive and transmit chains respectively. Furthermore, the AD9862 allows the USRP to process real or complex IQ signals. Additional details on the terms mentioned above, including decimation and interpolation, will be discussed in the upcoming section on the physical layer implementation to this system.

- **RFX-1200 Daughterboard:**

  The USRP supports a variety of daughterboards for many common spectrums of interest. For this research, the RFX-1200 is being used which supports any frequency between 1150-1450MHz with a transmit power of 200mW (or 23dBm).

Figure 6.1: Universal Software Radio Peripheral External View with Casing

This is a common frequency band for navigation and satellites, but it is useful for the purposes of this research because it is less crowded than the 2.4GHz band that is used by 802.11b/g Wi-Fi devices. The RFX-1200 has many useful features including a connector that is used for both transmitting and receiving, so an antenna can be shared for the two operations. With a 20MHz transmit and receive bandwidth, the board can support fairly wide signals. The daughterboard is full-duplex capable with some limitations that preclude its use in this research. The main limitation to full duplex operations are that the TX and RX side of the board have to be on separate frequencies. The board also contains an analog Received Signal Strength Indicator (RSSI) measurement via an auxiliary analog to digital converter (ADC) chip. Furthermore, this daughterboard

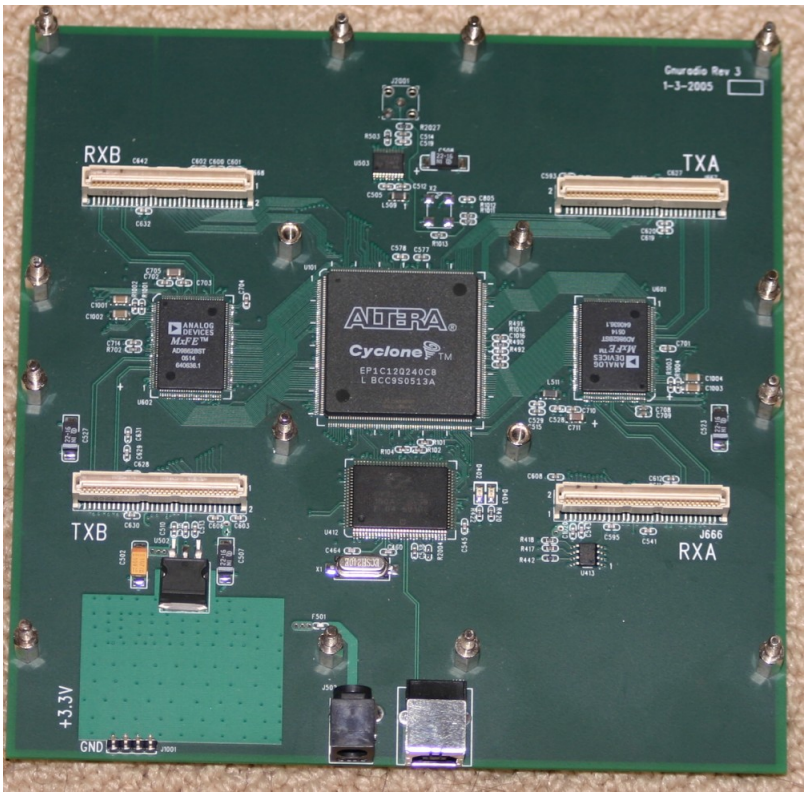

Figure 6.2: Universal Software Radio Peripheral Internal View

allows for adjustable transmit power and has a 70 dB automatic gain control (AGC) range. Another quite useful feature of this board are the independent local oscillator (LOs) for TX and RX which allows the board to transmit on one frequency and receive on another. The RFX-1200 is shown below in 6.4.

- **Omnidirectional Antenna:**

  The antenna for this system is a 7-inch vertical tri-band antenna that operates at 144MHz, 400 MHz, and 1200 MHz. The purpose of the antenna is to extend the range of the receiver or transmitter. The current setup requires an antenna

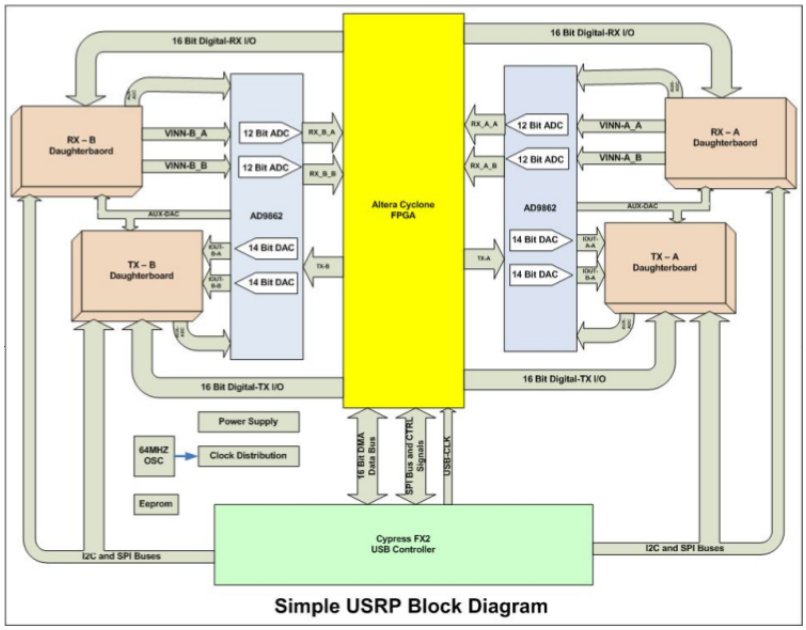

Figure 6.3: Universal Software Radio Peripheral Block Diagram

attached to each daughterboard within a USRP, so that is two antennas per USRP. A picture of the antenna in use is shown below in Figure 6.5.

6.2.2 Software Components

- **GNU Radio Platform:**

  The GNU Radio platform is a free software development toolkit that provides the signal processing runtime and processing blocks to implement software radios using the USRP as the RF hardware. The libraries are used in hobbyist, academic and commercial environments to support wireless communications

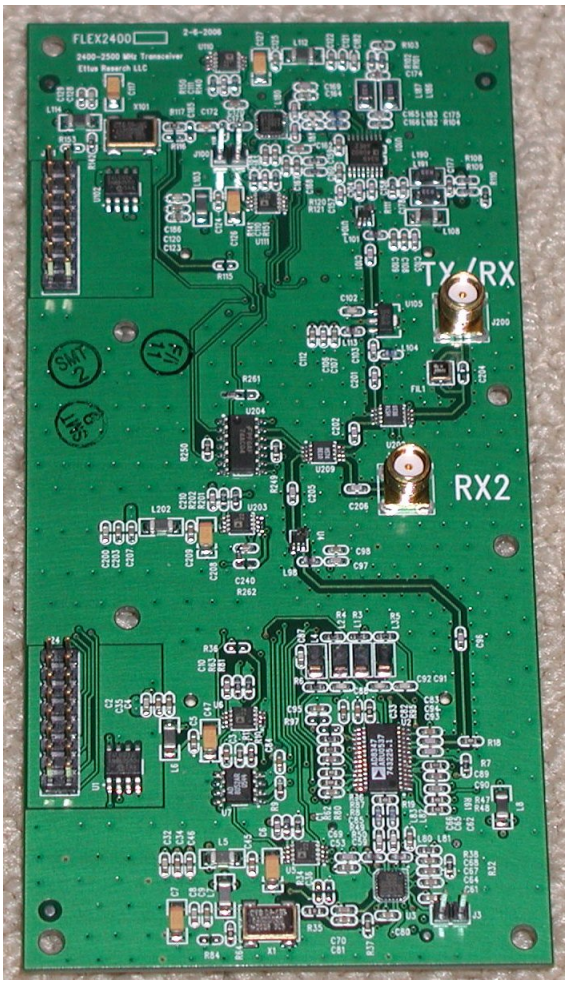

Figure 6.4: RFX-1200 Transceiver Daughterboard

research and implement prototype real-world radio systems. The research contained in this dissertation leverages the GNU Radio Platform for all transmission and receiving of RF signals. Operations with the USRP are conducted through the GNU Radio Application Programming Interface (API). The GNU Radio client applications are mostly written in Python, while the signal processing blocks are written in C++. At the time of this writing, a C++ API

Figure 6.5: VERT400 Vertical Omnidirectional 7-inch Antenna

for client applications was available but existed only in the Beta stage of development. Using this platform, a developer is able manipulate RF signals and perform complex digital signal processing operations quickly and effectively.

- **USRP API and Libraries:** As a part of the GNU Radio Platform, there is a compilation of libraries for operations involving the USRP device. The operations are specific to the USRP and control sub-devices such as daughterboards attached to the USRP. There is even a library extension that allows some access to the FPGA controls and registers. This becomes useful as more advanced signal processing techniques are employed and more control over the hardware is necessary.

- **wxPython GUI Library:**

  wxPython is a graphical user interface (GUI) toolkit for the Python programming language [Dunn 2009]. It is a Python extension of the wxWidgets cross-platform GUI library which is written in C++. Since Python is a wrapper

around the C++ libraries, it implements the same features as the wxWidgets libraries. This project is open source and is used by GNU Radio developers and other Python developers that wish to create rich GUIs for their applications. In the case of this research, the wxPython library is used to build a GUI for observing communication among USRPs and to control the parameters of the communication. As a cross-platform library, any GUI created with wxPython is well-suited to operate on any operating system that may be used to host a USRP.

- **USB Interface Library:**

  The USB library used by GNU Radio to access the USRP is the *libusb* library [Erdfelt 2008]. This library allows a user level application to search for and access connected USB devices from any operating system. The project is open source and customizable to the developer's needs. In this case, the distributed library is used without any modifications since it suites the needs of the radio platform. The main limitation with this library concerning the USRP devices is that once a USB port is claimed by a host process (such as a Python client application using GNU Radio) for reading or writing, the port cannot be opened again for that operation without the original process releasing the USB port. For example, if daughterboard A is conducting a transmission inside a USRP, and then daughterboard B wants to start transmitting as well, this operation will result in an error that the USB device cannot be claimed. The correct

way to do this is to stop all processes on the device and restart a process with instructions for both daughterboards in the same process. The details of this involve modifying some of the signal processing on the signals, but this will be discussed in detail in the physical layer implementation section.

- **Noise Signal Generator via GNU Radio:**

  To test the protocol's tolerance for noise in the spectrum, a jammer that adds noise to the spectrum is required. Rather than purchasing additional hardware, a separate USRP and RFX-1200 daughterboard acted as a signal jammer. The jammer script written in Python using the GNU Radio framework instructs the RFX-1200 to transmit either a uniform random or Gaussian random distribution of signals according to command line parameters. When in Gaussian mode, the signal is a random normally distributed Raleigh distribution with a mean about zero and a variance of one. It also takes an amplitude parameter to adjust the signal's amplitude and hence the strength of the jammer. Of course, the jammer operates on any specified frequency within the capabilities of the daughterboard. The jammer script is used to test the protocol's resistance to jamming. The source code for the jammer script is contained in the source code listing at the end of this document.

- **USRP Command and Control Application:**

  As mentioned above, the USRP must be driven by a computer that runs the GNU Radio platform. With two USRPs and one laptop, it becomes challenging

to transmit signals between two USRPs. Furthermore, it is useful to be able to get feedback from the devices when there is a successful transmission or reception of data. Otherwise, it is quite cumbersome to do testing and evaluate different parameters of the experiments. For this reason, a command and control application with a GUI was written that allows the operator of the host computer to send signals to two separate USRPs on two different USB ports. In addition, the application allows the operator to control individual daughterboards and receive textual feedback from the devices. The application is written in Python and leverages all of the software packages mentioned above including wxPython for the GUI, GNU Radio for the signaling, libusb for the communication with the devices, and the USRP libraries for giving instructions to the USRP. The application has several beneficial features that are enumerated below.

1. Allows an operator to control two USRPs and four daughterboards from a single computer.
2. Provides feedback on the success of data received or transmitted.
3. The operator may choose any data file or packet to send. Essentially, this application provides an application layer to the protocol stack.
4. Parameters for each daughterboard's transmit or receive activity may be changed separately and on-the-fly.

5. Allows access to the current internal transmit and receive packet queues in real-time for each daughterboard. Note that the packet queues are an aspect of the layer 2 implementation that will be discussed soon.
6. Keeps a historical record of transmitted or received messages for offline analysis and allows access to these queues in real-time.
7. Display meaningful feedback to the operator for informational or debugging purposes.
8. The display updates in real-time as the system changes parameters. For instance, if a radio switches from transmit to receive mode in preparation for receiving an acknowledgement, then the parameter field will update in the GUI for the operator to see.

The custom command and control application took a significant amount of time to develop, but it provides a useful application layer to the communications protocol developed in this research. The application also reduces testing overhead as it allows an operator to control twice as many USRPs as before. The application could easily be extended to support more USRPs if a computer had more than two USB ports, but this is not a priority at this point since geographically separating radios is useful for testing. Figures 6.6 and 6.7 show screenshots of the application used to control the radios from the computer. These two screenshots do not necessarily demonstrate all of the features of this application, but

merely show a preview of the interface. More screenshots are contained in the appendix section of this document.

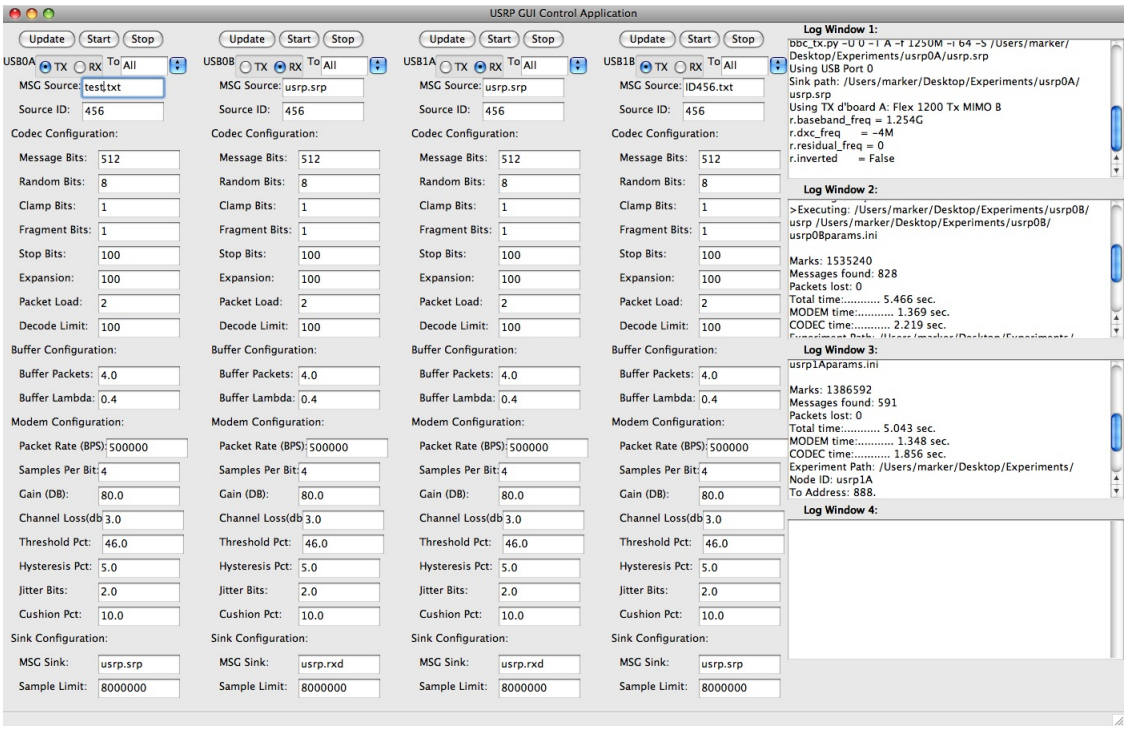

Figure 6.6: USRP Command and Control GUI Screenshot 1

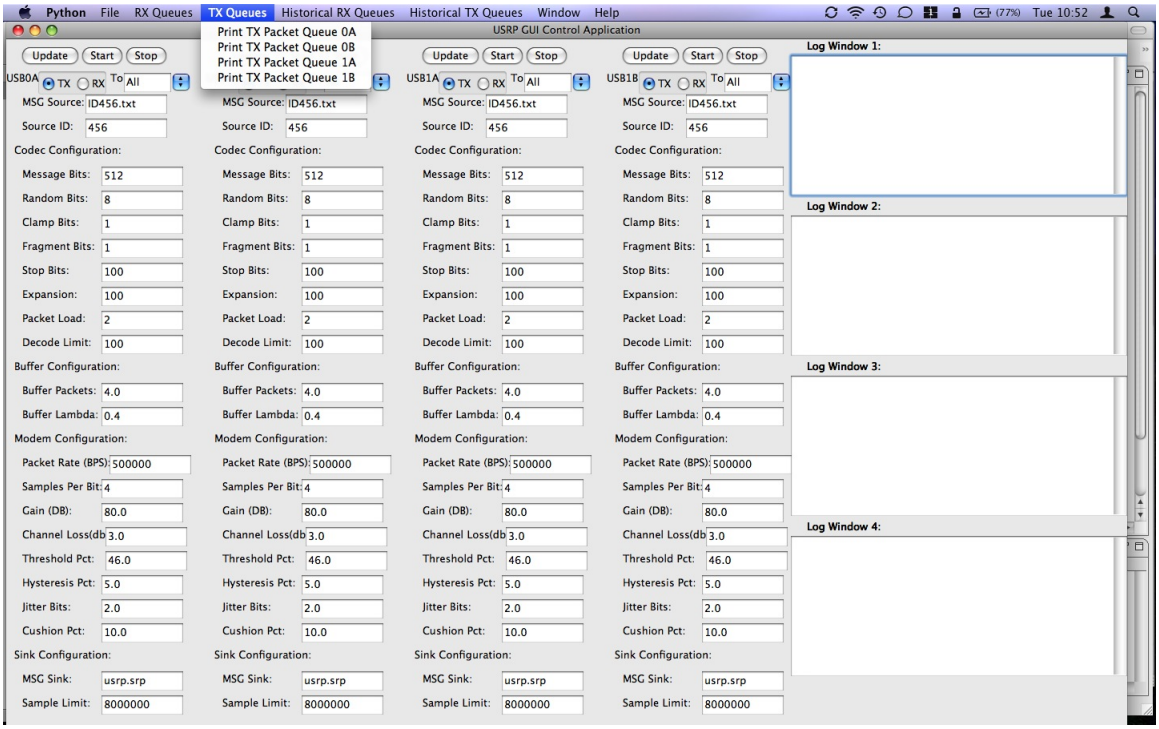

Figure 6.7: USRP Command and Control GUI Screenshot 2

- **Received Signal Strength Indicator (RSSI) Measurement Application:**

  In wireless communication systems, a device's RSSI is used to determine the amount of power present in the spectrum. Most often, this is used in carrier-sensing schemes that control access to the medium. For example, in 802.11, if the medium is sensed with a high RSSI value, that indicates that other stations are transmitting in the medium and the station performing the sensing must wait a random back-off period before attempting to transmit. In the protocol developed in the course of this research, the RSSI measurement is not used for carrier sensing. Rather, the measurement is used to determine the level of jamming or power in the spectrum at a given point in time. The presence of a jammer will undoubtedly increase the value of the signal strength indicator, while less jamming will be have the opposite effect.

  As discussed in the section on the USRP and the RFX-1200, the daughterboard has an auxiliary (AUX) analog to digital converter (ADC) which is on its own chip that does not interfere with the rest of the transmit or receive chains. This chip can be used to measure the RSSI value and determine the power within +/- 15MHz of the carrier frequency. Specifically, the AUX ADC reads the power in the analog baseband signals (power (I) + power (Q)). For an overview of complex IQ signal pairs, consult the discussion provided in Section 6.3. The RSSI measurement of the chip returns an integer value between 0 and 4095, inclusive. To change the bandwidth to something other than the +/-

15MHz requires moving some of the inductors and capacitors on the USRP, but in testing this RSSI value is accurate when correlated with the output of the signal jammer. As the level of jamming increases with the amplitude of the signal output from the signal jamming script, the RSSI measurement increases proportionally.

In order to automate this reading, a Python script was developed with use of the GNU Radio platform to calculate the average reading of the RSSI value from the AUX ADC. The reading is conducted by the FPGA in an asynchronous manner, therefore the increase accuracy it is necessary to average the RSSI reading over several thousand readings. Currently, the script averages the RSSI value over two-thousand readings and returns the average. Runtime for this script is quite fast and it returns the average RSSI value in about a second. As with other scripts run from the host computer, the program accepts parameters for the USB identification number to identify the USRP and another parameter to identify the specific daughterboard from which to take the reading. The source code for this script is contained in the source code listing.

- **BBC Encoder:**

  The BBC encoder software written in C performs the actual encoding of the data into complex IQ pairs for transmission via the USRP [Bahn 2007]. The data is encoded as binary data with a series of indelible marks that are represented by one-bits, while the absence of data is represented by a zero-bit. The

original implementation shown in Algorithm 1 [Baird, Bahn and Collins 2007] does not adapt to varied levels of interference in the spectrum. However, the newer version shown in Algorithm 2 dynamically adjusts the codeword length in response to the RSSI measurement taken at the data link layer and a route dependent threshold determination performed by the network routing layer. The threshold and codeword length determination is covered in the upcoming section on the adaptive jam-resistant network routing layer implementation.

Algorithm 1 BBCEncodeOriginal(M)

This function encodes an m-bit message M[1...m] adding k checksum bits to the end of the message. H is a hash function. The definition of H and the value of m and k are public (not secret). The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

Append k zero bits to the end of M
for $i = 1$ ... $m + k$ **do**
  Make an indelible mark at the location given by H($M[1...i]$)
end for

Algorithm 2 BBCEncodeDynamic(M, J)

This function encodes an m-bit message M[1...m] adding k checksum bits to the end of the message. H is a hash function which is truncated according to the level of jam-resistance. By default, H returns a SHA-1 hash of 160-bits. Truncate this 160-bits according to the jam-resistance level. The definition of H and the value of m and k are public (not secret). The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

Append k zero bits to the end of M
Jam resistance level is dictated by thresholding implemented at network layer and is a parameter J with a possible integer value [1..5]
for $i = 1$ ... $m + k$ **do**
  $H_{truncated} = H_{original}$ truncated to length 32*J
  Make an indelible mark at the location given by $H_{truncated}(M[1...i])$
end for

- **BBC Decoder:**

  As with the BBC encoder software, the BBC decoder is written in C in its current form [Bahn 2007]. This software was left in C for speed considerations with the bit-level manipulations. Decoding is especially taxing on the host computer because the source file that is created by the received signal from the USRP can be quite large. For example, if the receiver is in receiver mode for approximately 10 seconds, the received IQ pairs from the radio form a binary file that is about 300Mb large. In the presence of more noise, this receiver may have to receive for an extended period of time and the received signal file may exceed 1 GB.

  The original BBC decoder algorithm is shown in Algorithm 3 [Baird, Bahn and Collins 2007]. It does not adjust to the dynamic codeword length and simply proceeds through the decoding phase by translating the indelible marks back into the original file and using the checksum bits to eliminate hallucinations. The dynamic BBC decoder algorithm shown in Algorithm 4 decodes messages through a series of codeword lengths dictated by the parameter it is passed. Another variant of this decoder may use a breadth-first search algorithm to search for messages at the various codeword lengths that are used by the dynamic BBC encoder, however this has not yet been implemented.

Algorithm 3 BBCDecodeOriginal(n)

This recursive function can be used to decode all the messages found in a given packet by calling BBCDecode(1). There must be a global $M[1...m+k]$ which is a string of $m+k$ bits. The number of bits in a message is m, and the number of checksum zeros appended to the message is k. The definition of H and the value of m and k are public (not secret). The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

if $n = m + k + 1$ **then**
  print ''"One of the messages is:" M[1...m]''
else
  if $n > m$ **then**
    $limit \Leftarrow 0$
  else
    $limit \Leftarrow 1$
  end if
  for $i = 0$ ... $limit$ **do**
    $M[n] \Leftarrow i$
    if there is an indelible mark at location H(M[1...n]) **then**
      BBCDecode(M,$n+1$)
    end if
  end for
end if

Algorithm 4 BBCDecodeDynamic(n, J)

This recursive function can be used to decode all the messages found in a given packet by calling BBCDecode(1). There must be a global $M[1...m+k]$ which is a string of $m+k$ bits. The number of bits in a message is m, and the number of checksum zeros appended to the message is k. The definition of H and the value of m and k are public (not secret). H will be truncated to the value related to the jam resistance level parameter passed to the function. The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

if $n = m + k + 1$ **then**
  print `''"One of the messages is:"` M`[1...`m`]''`
else
  if $n > m$ **then**
    $limit \Leftarrow 0$
  else
    $limit \Leftarrow 1$
  end if
  for $i = 0$ ... $limit$ **do**
    $M[n] \Leftarrow i$
    $H_{truncated} = H_{original}$ truncated to length 32*J
    if there is an indelible mark at location $H_{truncated}(M[1...n])$ **then**
      BBCDecode(M,$n+1$)
    end if
  end for
end if

- **Optimized Link State Routing (OLSR) daemon:**

  A major software component to this system is a modified OLSR layer that maintains route tables for nodes in the network [Clausen, Dearlove and Jacquet 2008], [Tonnesen 2009]. Some additional background information on the OLSR protocol is contained in Chapter 3, Section 3.2. The *OLSRd* open source project has many contributors, including myself, and maintains a highly active development community unlike other routing protocols. The algorithm is a proactive routing algorithm that uses topology control and "hello" message broadcasts to share information with other nodes in the mesh. OLSR's "Hello" messages are used to find one and two hop neighbors based on the responses from other nodes. Among these neighbors, the sender selects its multipoint relay (MPR). As a criteria to be selected as an MPR, the MPR must have a path to each of the sending node's two-hop neighbors. The MPR node will then forward topology control (TC) messages containing the nodes that have selected the MPR as their relay. Unlike other link state protocols, not all inks of a node are advertised. In order to prevent problems due to significant message flooding, the nodes that have selected MPRs share their link state information. The format of the topology control and hello messages is shown below in Figure 6.8 and 6.9. The routing tables are kept current and reliable through the periodic broadcasts of the hello and topology control messages. In this research, the determination of the route is modified to choose the best route according to spectrum interference that may affect link quality. Furthermore, the routing

layer determines what is the most effective jam-resistant encoding scheme to ensure availability between the source and destination nodes.

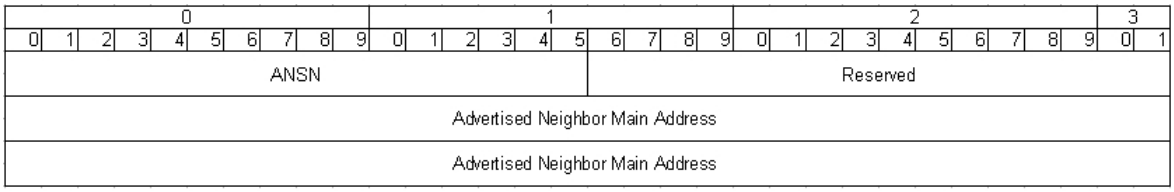

Figure 6.8: Topology Control Packet Format [Clausen, Dearlove and Jacquet 2008]

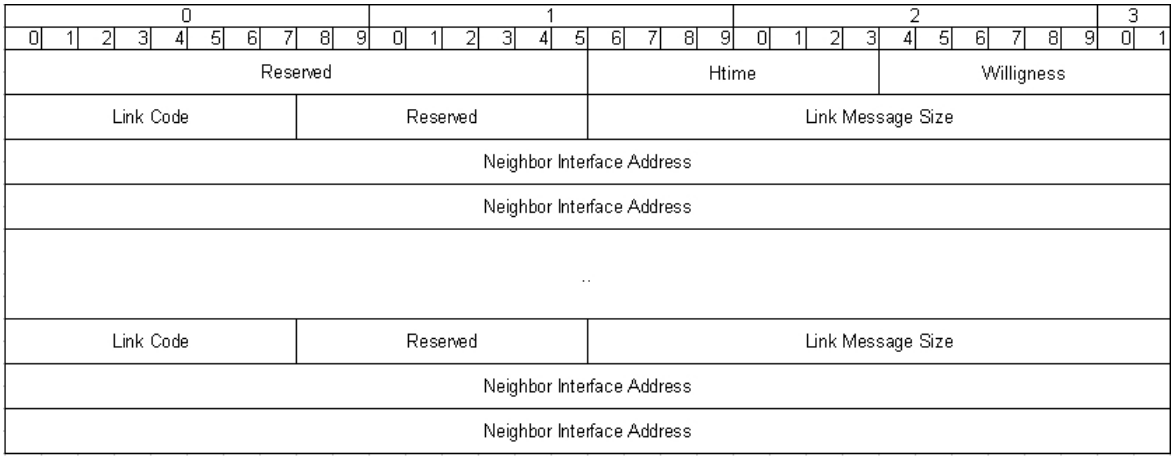

Figure 6.9: Hello Packet Format [Clausen, Dearlove and Jacquet 2008]

6.3 Physical Layer Implementation

The following section will discuss implementation details of the physical layer for this jam-resistant communication system. The detailed operation of the physical layer and implementation of specific features at this layer will be discussed. Key portions of this section include a discuss of in-phase quadrature modulation used, the

decimation and interpolation factors, and the custom implementation for activating simultaneous receive or transmit chains on the same USRP.

In this system, the data to be transmitted via the USRP is stored on a host computer. In some way, this data must be converted into a suitable format prior to transmission in the wireless medium. Radio frequency data is often stored as in-phase quadrature (IQ) data which can be represented in a file stored on the computer. IQ modulation is also known by many other names including complex or base-band modulation. An IQ signal is a mathematical representation of a real valued signal combined with a complex sinusoid, with the resulting complex data (i,q) being stored as a 32-bit signed integer with 16 bits for I and 16 bits for Q. Each IQ pair represents a sample of the complex signal. In this research project, the actual IQ modulation of the source data is handled in software by the BBC real-time engine. As mentioned in previous sections, the USRP contains DSP chips to handle reception or transmission of IQ pairs, as well as the critical operations of decimation and interpolation of these signals. To enhance understanding and readability, it is important to define some of the terms mentioned above.

First, decimation is the process of reducing the sample rate. Decimation is usually accomplished through the use of a low-pass filter and tossing some of the samples out. A similar activity is known as down-sampling which reduces the sample rate by throwing away some samples without the use of a low-pass filter. When using decimation with the USRP, a decimation factor is provided on the command line. The decimation factor is the ratio of the input rate to the output rate. The most

common reason for using decimation or down-sampling is to reduce the cost of signal processing.

The opposite process of decimation, interpolation is used in this case on the transmitter to increase the original sampling rate to a higher rate. During this process, zero-valued samples are inserted between the original samples to increase the sampling rate. This insertion process, sometimes called up-sampling, adds undesired spectral images to the signal in integer multiples of the original sampling rate. Interpolation is specifically performing up-sampling on a signal and filtering the results to remove the undesired spectral images. At the end of this process, the result is a signal that appears as if it had been sampled at a higher rate than it was originally sampled. When using interpolation with the USRP, an interpolation factor is provided on the command line of the transmitter. The most common reason to interpolate a signal is to increase the sampling rate at the output of one system so that another system operating at a higher sampling rate can input the signal, but also provides some cost savings on signal processing. Figure 6.10 provides a visual example of interpolation in practice.

The relationship between decimation and interpolation is dictated by the Nyquist Sampling Theorem. The theorem states that for a band-limited signal with maximum frequency, f_{max}, the sampling frequency f_s must be greater than twice the maximum frequency f_{max} in order to have the signal reconstructed without aliasing. Another name for aliasing is under sampling. Under sampling causes frequency components that are higher than half of the sampling frequency to overlap with lower frequency

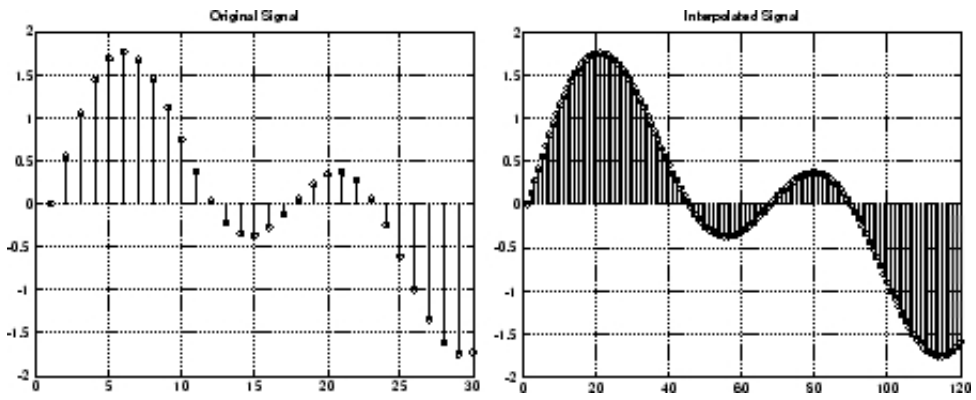

Figure 6.10: Interpolation Example [IEEE 1979, MathWorks 2009]

components causing distortion of the reconstructed signal. Figure 6.11 provides a visual example of under sampling and illustrates how the reconstructed signal does not match the original signal when the Nyquist Theorem is not followed. When it comes to transmission between USRPs, a decimation factor of 64 is used on the receiver while an interpolation factor of 128 is used on the transmitter to allow the receiver to receive a complete signal while saving on signal processing costs. As is required by the Nyquist Sampling Theorem, if we want N samples per second on the receiver, we need to choose a decimation rate such that the decimated frequency is greater than or equal to N/2. Therefore, a decimation rate of 64 on the receiver should allow us to sample 1M samples per second when the original sampling frequency set by an interpolation rate of 128 by the transmitter produces also 1M samples per second which works well. The higher sampling rate is not absolutely necessary, but increased sampling rates reduce the signal-to-noise-ratio (SNR) in analog to digital conversion operations. With a current USB maximum bus speed of 32MB/s, and each complex IQ sample taking 4 bytes, it is important to make sure the sampling rates

do not exceed the capacity of the bus. Of course, all receive and transmit channels share the same USB when communicating with the computer.

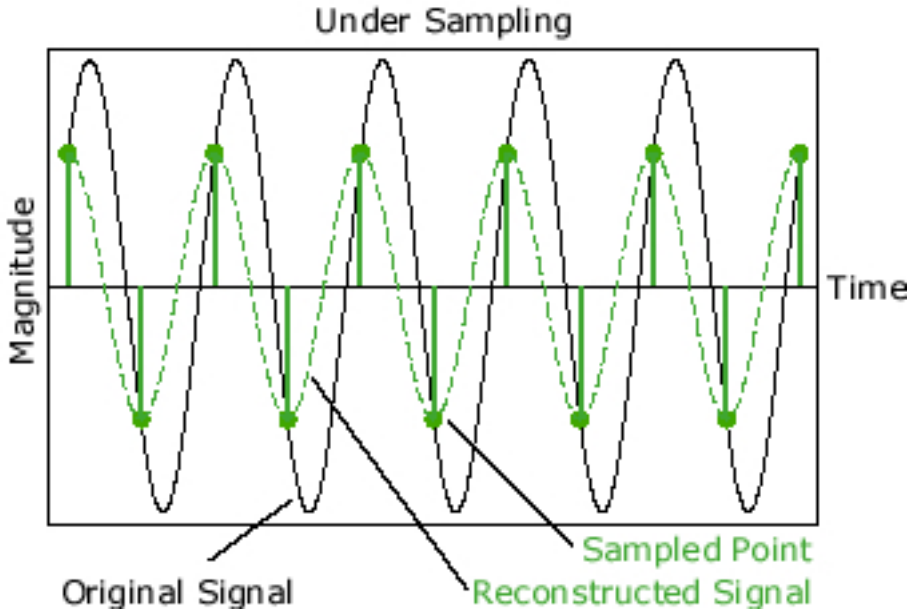

Figure 6.11: Under Sampling Example [EFunda 2009]

The current setup for the USRP in this research involves placing two RFX-1200 daughterboards on a single USRP unit. As one would expect from a communications system, it became necessary for both daughterboards inside the USRP to be in a transmit state or receive state simultaneously. The case where board A is in a TX state and board B is a RX state works by default, however the case of simultaneous and identical states required special signal processing to take place. In order for both daughterboards inside a USRP to transmit or receive at the same time, multiple channels must be used on the USRP. The board supports up to four real-valued signal channels and up to two complex-IQ signal channels.

In the transmit case, two channels are setup on the USRP board. Each channel is assigned a source for the transmission. In the case of this protocol, the source is a file that contains the IQ modulated signal ready for transmission. To transmit from both channels simultaneously, the signals are interleaved. The AD9862 signal processing chip contains hardware to do the interleaving on the USRP rather than the host computer which saves significant processing costs. At transmission time, if two IQ signals are interleaved, the data will appear as the following: I0 Q0 I1 Q1 I0 Q0 I1 Q1 and so on. The essence of this process in software is shown in the code excerpt below. The full source code for this operation is located in the source code listing.

Listing 6.1: Signal Interleaving for Simultaneous Transmissions

```
#establish transmit source for the two channels
self.txfileA = gr.file_source (gr.sizeof_gr_complex, sink_path, 1)
self.txfileB = gr.file_source (gr.sizeof_gr_complex, sink_path_B, 1)
#establish the USRP object with appropriate USB port
self.usrp = usrp.sink_c(which=usb_num, interp_rate=self.interp,nchan=nchan)
#setup interleaver
intl = gr.interleave(gr.sizeof_gr_complex)
#connect components
self.connect(self.txfileA, (intl, 0))
self.connect(self.txfileB, (intl, 1))
self.connect(intl,self.usrp)
```

On the receive side, the receiver must deinterleave the signals and process them separately. A similar process takes place but it must occur in reverse. In order for both receivers to receive simultaneously, multiple channels are setup. Multiple channel sinks are setup as well so that each channel stores its received IQ signals in a separate location. Then, as the receiving operation starts, the signals are deinterleaved and

separated into different file sinks containing separate IQ data pairs. Continuing with the example described above, an interleaved signal will be of the form I0 Q0 I1 Q1 I0 Q0 I1 Q1. Then, the received signal after being deinterleaved will be I0 Q0 I0 Q0 for the first channel and I1 Q1 I1 Q1 for the second channel. The essence of this process in software is shown in the code excerpt below. The full source code for this operation is located in the source code listing.

Listing 6.2: Signal Interleaving for Simultaneous Receptions

```
#establish separate file sinks for the two channels
self.dst_A = gr.file_sink(gr.sizeof_gr_complex, filename_A)
self.dst_B = gr.file_sink(gr.sizeof_gr_complex, filename_B)
#establish the USRP object with appropriate USB port
self.u = usrp.source_c(which=options.usb_num,decim_rate=options.decim)
#setup deinterleaver
di = gr.deinterleave(gr.sizeof_gr_complex)
#connect components
self.connect(self.u, di)
self.connect((di,0),self.dst_A)
self.connect((di,1),self.dst_B)
```

As discussed above, the source data is put in complex IQ pair form prior to transmission in the physical medium. As a series of binary data comprised of 0's and 1's, the signal modulation scheme currently in use is a simple pulse-based system. A zero-bit is transmitted as a flat zero-gain pulse and a one-bit is transmitted as a high-pulse according the the gain. More complicated schemes are available, but the current scheme is working as desired. More sophisticated schemes of interest include direct sequence spread spectrum (DSSS), but this was not implemented in this version of the physical layer because it requires some synchronization between the waveforms which is not well supported with the current USRP setup. Specifically, the clock

pulse is shared between both daughterboards in the USRP and synchronization is accomplished through a physical wire between the two boards or between USRPs connected to an external clock. A better solution to the waveform synchronization problem is necessary to accommodate synchronization among multiple USRPs that contain multiple daughterboards driven by multiple host machines.

6.4 Data Link Layer Implementation

In order to test the adaptive network layer, a simple data-link layer proved to be a necessary component. For this reason, a data-link layer was implemented with the goal in mind to be as simple as possible in order to reduce packet processing overhead. This goal led to the investigation of using the simple Aloha protocol for the data-link layer, and perhaps even the slotted aloha protocol. A contention based protocol that uses carrier sensing to determine whether to transmit or not would likely not operate well in the presence of jamming since the medium would always be sensed as busy. Therefore, a scheme similar to what 802.11 networks use with request-to-send (RTS) and clear-to-send (CTS) broadcasts seemed inappropriate for the likely use case of this jam-resistant communications protocol stack. With these principles in mind, a simple Aloha medium access control (MAC) protocol was implemented in Python to leverage the GNU Radio framework and the GUI interfaces via wxPython.

The Aloha protocol is quite simple to understand in comparison to some other modern schemes. Devised in the 1970s to enable wireless communications between the Hawaiian Islands, the Aloha protocol dictates that a network node will transmit

whenever it has a packet to send. If the packet successfully reaches the destination, the next packet can be sent. Otherwise, the packet is retransmitted since a collision occurred as other nodes attempted to transmit at the same time. Successful reception of a packet is determined by the successful reception of an acknowledgement from the receiver that the packet has indeed arrived. This form of Aloha, coined "Pure Aloha", has a maximum theoretical throughput of 18%.

Given the dismal throughput of the original Aloha protocol, a variant of this concept improved this protocol by using time slots to assign discrete transmission times to nodes. The improved protocol, known as "Slotted Aloha", dictated that transmissions must occur within a specific slot which is equal to the amount of time necessary for a fixed length packet to be transmitted. Therefore, when a packet is ready to be sent by a node, the node must wait until the start of the next time slot. In this scheme, packets overlap completely or not at all. Slotted Aloha improves upon Pure Aloha by 100% with a theoretical throughput of 36%. The Slotted Aloha approach did not seem necessary to implement at this point in time since synchronization among USRPs is limited in some aspects by the hardware device.

The typical analysis of the Aloha family of protocols is particularly critical of the performance since the throughputs of the two approaches are extremely low compared to the bandwidth of the channel. For this reason, carrier sensing schemes are created which use the request-to-send (RTS) and clear-to-send (CTS) messages to make sure the channel is clear prior to transmission. Retransmissions are still frequent under the RTS/CTS style since nodes out of the range of these beacon packets are unaware

that other nodes are transmitting as well. The hidden station and exposed station problems are rooted in this critical flaw in the RTS/CTS style of contention resolution.

In the communications protocol developed in this research, resistance to jamming is a critical component. As a part of this resistance to jamming, there is an added capability supplied by the BBC algorithm which allows a receiver to decode several messages at the same time from the same received signal. Given this capability, collisions are not as much of a problem since messages will still be decoded in significant levels of noise caused by the collisions with other nodes. Pure Aloha has poor throughput since it relies on transmitting one message at a time and waiting on an acknowledgement for each message. Instead of this approach, the jam-resistant network stack simply combines all of the messages in the queue into a single message for transmission all at once. Furthermore, with jam-resistance built into the signal encoding, the likelihood that a message has to be retransmitted due to corruption from a collision is reduced significantly. All of these factors combine to increase the throughput of the Pure Aloha MAC protocol used with the USRPs. As USRPs receive messages, they decode them and analyze their contents. If the message is original and addressed to them, the message is accepted and added to the receive queue. After it is added to the receive queue, the node sends a unicast acknowledgement to the appropriate node. If a node receives an acknowledgement packet for a message it transmitted, that previously transmitted message is moved to a historical record keeping queue. However, if no acknowledgement is received within a predetermined amount of time, the message will be retransmitted. This process continues until there

are no more messages in any node's transmit queue. The main limitation of using this type of Aloha protocol on USRPs is that a daughterboard cannot transmit and receive at the same time. Therefore, if a daughterboard A on USRP 0 is transmitting a message to daughterboard A on USRP 1, no reception will occur while these two operations overlap since none of the nodes are in a receiving mode. This state is unlikely given the Aloha protocol since the default state for the nodes is the receive mode and both nodes would have to be synchronized almost exactly since transmission does not take longer than a few seconds even for large messages. In either case, experiments have proven that reliable transmission is possible between four daughterboards operating on the same frequency and all exchanging multiple messages among each other.

Throughput of this MAC protocol is difficult to measure since there is a significant amount of time spent on message processing once a message is received. Experiments have shown that it takes between 4 and 12 seconds to run the decode algorithm on a 4KB packet since the received signal is approximately 300Mb if the receiver operates for 10 seconds. Packet analysis is conducted in negligible time and encoding takes between 0.5 and 5 seconds depending on the packet size of course. The upper limit of the throughput is dictated by the speed of the USB which is shared among the USRPs connected to the driving computer. In any case, the main goal of this jam-resistance protocol is not to increase the speed of wireless communications, but rather to guarantee delivery of messages in the presence of noise where other protocols fail. The data link layer protocol based on Pure Aloha that was implemented

and tested in this research has shown that guaranteed delivery of messages is possible with significant levels of noise. The details of this experimentation will be discussed in future sections.

An additional feature added to the data link layer is logical addressing. A simple logical addressing scheme using random addresses is used to assign unique addresses to every node from the control application. The random addresses are generated using standard library random number generator functions. Even considerably large sets of these pseudorandom numbers did not present any duplicate addresses. The system time is used as a seed to the generator and sets of up to twenty thousand did not present any duplicates. It is highly unlikely that a network will contain this many nodes, but it is good to push the addressing scheme to unforeseen limits. The layer automatically handles addressing collisions if one does occur. The address is integrated into the packet header prior to the BBC encoding process. The header currently has several fields which are listed below that are assigned by the light-weight MAC layer.

- **checksum** of 4 Bytes (32 Bits)
- **sequence number** of 2 Bytes (16 Bits)
- **packet id number** of 2 Bytes (16 Bits)
- **to address** of 2 Bytes (16 Bits)
- **from address** of 2 Bytes (16 Bits)

- **RSSI measurement** of 2 Bytes (16 Bits)

The USRP has a media access control (MAC) hardware address, but this is not guaranteed to be unique in a larger network. Therefore, it is not being used for addressing at any layer. For the sake of simplicity in the addressing scheme, the logical addressing performed at the network layer is translated down to the data-link layer directly. Essentially, there is only one addressing scheme for a daughterboard which can transmit and receive in the network. If there is an address collision, the node's address will match the "to address" field of a received packet and this triggers the node to assign itself another address at random. Currently, this is the only form of duplicate address detection (DAD), but more sophisticated schemes could be employed in the future as the network grows. However, having collisions of random 16-bit addresses is rather unlikely even in networks with hundreds of nodes. Furthermore, there does not seem to be a need for additional addressing at the hardware level such as that used in typical data-link layer frames. By reducing the addressing overhead and reducing the size of the address by 50% (IPv4 has 32-bit addresses), the jam-resistant protocol compensates for some of the overhead imposed by the BBC encoding scheme.

In summary, the light-weight data link layer created in this research is meant to add the additional functionality to the pure physical layer wireless communications. This data link layer disassembles data to be transmitted into packets of 512 bytes (equal to the size of a USB frame), and adds a header with fields designed to build reliable messaging into an unreliable wireless transmission medium. With a Pure Aloha design, simple logical addressing, and an acknowledgement framework, this

layer has proven through experimentation that reliable messaging is possible in the presence of significant spectrum interference. These experimental results are covered in the Phase I Results chapter. In addition to the physical and data link layers discussed thus far, an additional network layer that makes global routing decisions based on spectrum interference is built on top of these two layers. The network layer accommodates networks with more nodes by incorporating intelligent routing and instructs the data link layer how to most effectively encode the transmitted packets for successful reception by a receiver. The specifics of the routing protocol adapted for use in this wireless communications stack will be discussed in the next section.

6.5 Network Layer Implementation

As discussed in Chapter 3 on MANET routing protocols, a popular proactive routing protocol is the Optimized Link State Routing (OLSR) routing protocol (Section 3.2). OLSR attempts to reduce the number of transmissions required for control traffic by efficiently flooding through nodes selected as multi-point relays (MPRs). OLSR version 2 [Clausen, Dearlove and Jacquet 2008] improves throughput in MANETs since it creates bi-directional routes with control traffic instead of unidirectional routes. This protocol has been selected for adaptation and use within the jam-resistant protocol stack because it has been tested in mobile IP-based networks with some success. Furthermore, choosing a protocol as popular as OLSR as the foundation for the jam-resistant network layer developed in this research will hopefully encourage its use and adoption in the developer community.

Modifying existing software is usually not an easy task. As expected, modifying the existing OLSR implementation for use on software defined radios such as the USRP is complex. For starters, the existing OLSR implementation connects to the outside world through Ethernet ports only. It assigns IPv4 and IPv6 addresses to connected interfaces and broadcasts control packets only over the predefined interfaces. The current design is problematic for use with the USRP because all communications with the USRP pass through the USB 2.0 ports, rather than the standard Ethernet interfaces. Consequently, rewriting the current OLSR implementation to handle communications with the USB ports of Unix and Linux operating systems is a challenge. Moreover, OLSR uses sockets to open specific ports and even uses *iptables* firewall rules to route packets inside the system which increases the number of dependencies that must be changed.

The modified OLSR implementation, coined *"BBC-OLSR"*, makes the following changes:

1. All communications with external nodes pass through the USB port in order to use the USRP software defined radios.

2. IPv4 addresses are converted to a smaller address space that uses 16-bits. IPv6 addresses are not used. The address conversion is handled in a similar way to Network Address Translation (NAT). A table is maintained that translates between the IPv4 and 16-bit address formats, but only the 16-bit addresses are

transmitted and used with the software radios. The data link layer uses the 16-bit addresses as well, but OLSRd uses the IPv4 addresses internally to maintain the capability for network extension to the Internet if necessary in the future.

3. Packet headers are modified to include the necessary BBC message header information

4. Routing decisions incorporate feedback from the data-link layer pertaining to the RSSI measurement to determine the level of spectrum interference present.

5. The data-link layer is instructed to use a specific codeword length to account for spectrum interference and to guarantee node availability. As noise increases, so does the codeword length (level of jam-resistance). As noise decreases, the codeword length decreases accordingly.

6. Node-to-node delivery is handled by the data-link layer. Global routing between nodes is handled by the network protocol. As such, control traffic includes noise measurements to guide network-wide jam-resistance initiatives to guarantee availability.

7. Topology Control (TC) and "Hello" messages are grouped into sets of ten or more packets before being encoded with the BBC algorithm and transmitted on the radios. The reason for this is to alleviate some of the delays from the decoding and encoding step. For example, in the current implementation, the "hello" messages are transmitted every half-second. If the message was passed

to the encoder directly, many more messages would be in the queue while encoding is taking place on the first packet. Furthermore, these messages are only twenty-bytes so it is more efficient to group them in a bundle for encoding and transmission.

In order to provide adaptive jam-resistance, BBC-OLSR analyzes the RSSI measurements taken at each node and transmitted with every message exchanged in the network. Therefore, when BBC-OLSR receives a packet from another node, it will know the noise level present in the spectrum at the time the packet is transmitted from the node. With this knowledge, BBC-OLSR determines the proper level of jam-resistance to employ at the data link layer in order to ensure message delivery to an intended recipient. For example, if there are four nodes in the network A, B, C, and D. And a path from A to D passes through B and C, so the route to be taken by a message is A-B-C-D. The BBC-OLSR agent at A will have relatively recent (dependent on broadcast interval) RSSI values for itself and the other nodes. Then, a simple scheme of resisting the highest level of noise among all nodes on the path is taken. By setting the level of jam resistant to accommodate the highest known level of spectrum interference along a path, the protocol ensures that message delivery and node availability are given the highest priority. Other schemes could surely be employed, but if the jamming source is mobile, then it is likely other nodes along the path may experience increased levels of noise after transmission from the originating node. For this reason, the scheme of resisting the highest known noise level is taken.

The experimental results involving the deployment and use of BBC-OLSR and the associated physical and data link layers are detailed in Chapter 9.

6.6 Software Architecture

The purpose of this section is to detail the software architecture of this communications system. Without design documentation, understanding the relationships among the various software and hardware components becomes rather complex. As the software has evolved, so have these diagrams. Furthermore, the diagrams contained in this section represent only the most useful representations of the software that have been deemed necessary to the understanding the interactions among components.

First, the physical layer state diagram is provided below in Figure 6.12. At the physical layer, the radio is either idle, receiving, or transmitting. These high-level states represent the physical radio states that are relevant to the upper layers.

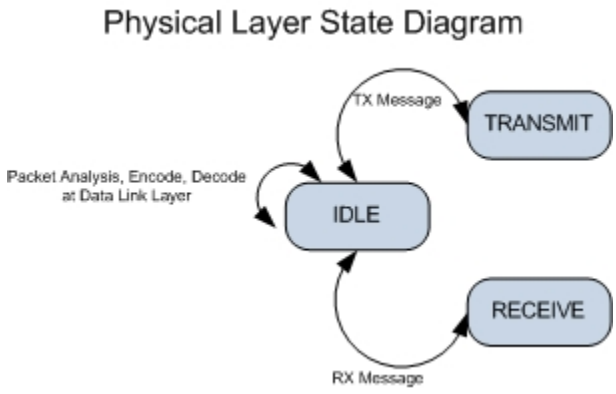

Figure 6.12: Physical Layer State Diagram

Next, the data-link layer state diagram is shown below in Figure 6.13. These diagram also incorporates the states of the physical layer so the interactions between the two layers can be illustrated. The legend provides a key to identify which states belong to which layer. A clear distinction between the layers is necessary to promote loose-coupling among components to promote future integration efforts with externally developed layers.

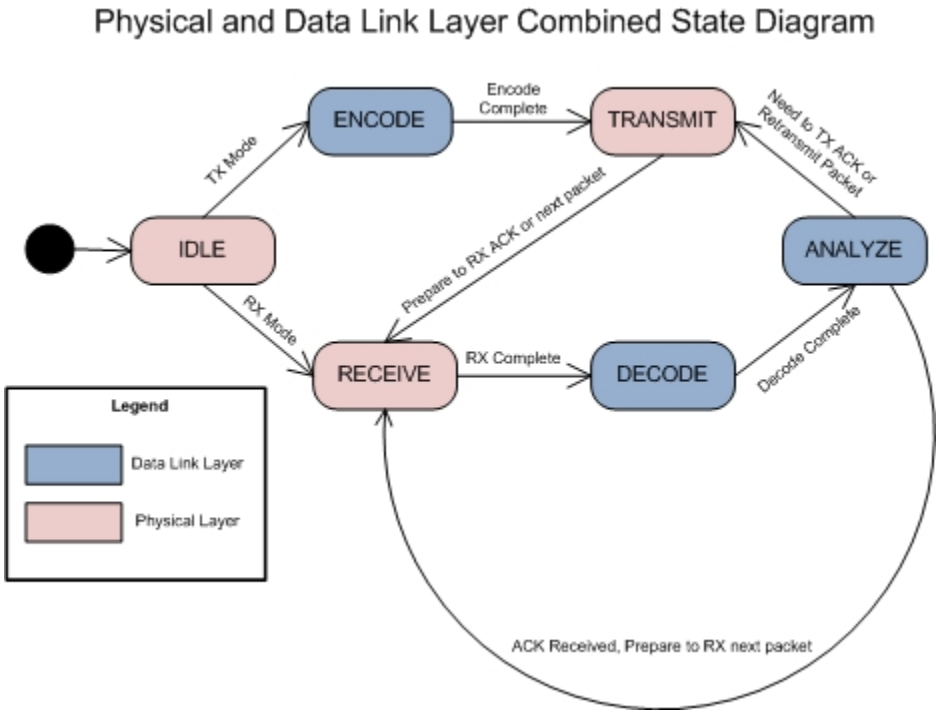

Figure 6.13: Physical and Data Link Layer State Diagram

The state diagram for the network layer built on top of the previously discussed physical and data link layers is shown below in Figure 6.14. The various states of the layers are shown as well as the interactions between the three layers. The network layer's main function is to maintain routing information for the network. However,

the network layer in this research also controls data encoding at the data link layer to adjust the level of jam resistance throughout the network. At any one time, the various layers may be in any number of states. For example, the data link layer could be decoding a recently received message while the physical layer is transmitting another message. If messages are to be transmitted but the physical layer is in a receive state, the messages are queued for transmission for the next time the physical layer enters the transmit state.

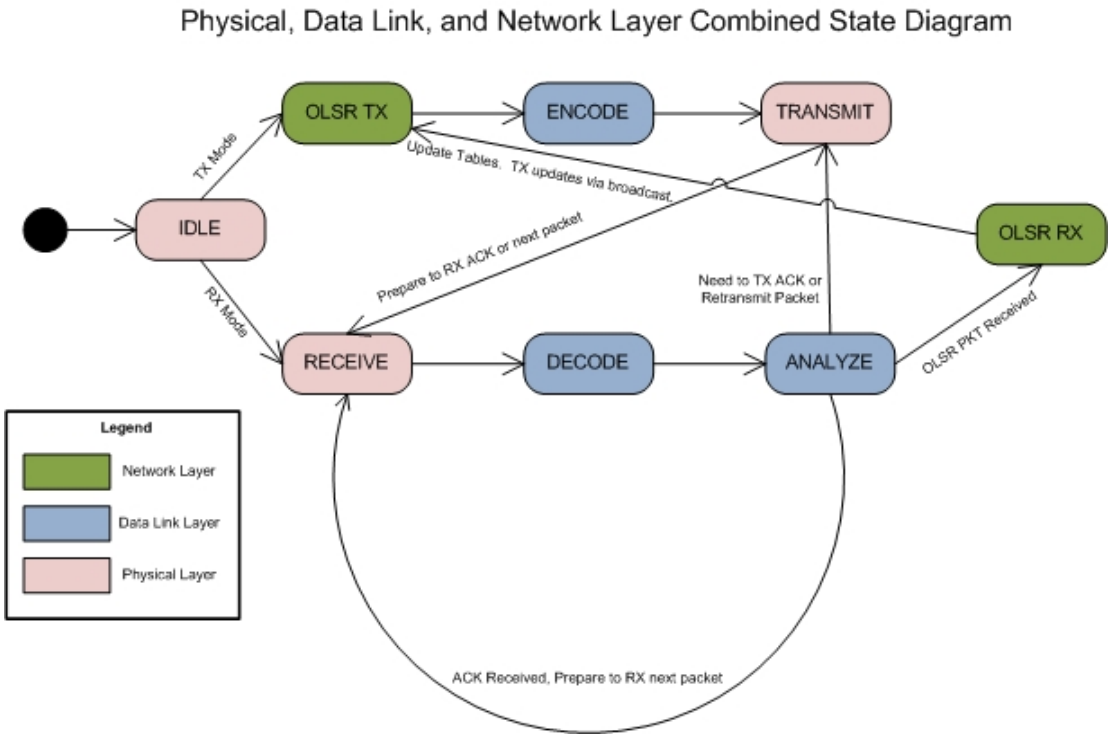

Figure 6.14: Physical, Data Link, and Network Layer State Diagram

6.7 Chapter Conclusion

The purpose of this chapter is to explain the implementation of the communications protocol developed in this research. We began the chapter with a discussion of the hardware and software components. Then, implementation details of the physical, data link, and network layers are discussed. In addition, the application layer command and control application is introduced which provides the necessary visualization layer for testing the protocol. Near the end of the chapter, some software architecture diagrams are provided to enhance understanding of the design. As with any large research or engineering effort, there will always be pieces that can be improved upon or designed in a different way. The implementation described in this chapter is meant to provide a working prototype to prove the technical feasibility of the research ideas. Future development efforts will of course result in improvements to this system as the software and hardware components evolve over time.

Chapter 7

Phase I Experimental Results

7.1 Chapter Introduction

Phase I results demonstrate that a BBC-enabled protocol stack for wireless devices provides a solution to the hidden station and exposed station problems that plague nodes in a wireless network. Wireless nodes suffer friendly jamming as a consequence of multiple nodes attempting to transmit at the same time as detailed in Chapter 2, Section 2.2.4. The traditional solution to this problem has been carrier-sensing where a node will use request-to-send (RTS) and clear-to-send(CTS) packets to claim the wireless medium for their transmission. The hidden station problem exists since a node out of range will not receive the notification that another node is transmitting and therefore a collision occurs as it transmits. In a similar scenario, the exposed station problem occurs when a node wishing to transmit is exposed to another station's transmission and senses the medium as busy when it is in fact free. In the presence of jamming where the spectrum is full of noise, carrier sensing schemes will always detect the medium as busy and therefore cease to operate. Jamming also causes collisions such as those that occur in the hidden station issue. The BBC-enabled protocol stack uses jam-resistant message encoding to resist jamming and is well-suited to solve the problem of friendly station interference.

7.2 Hidden Station Problem Experiment

In this experiment, the main goal is to prove a solution to the common hidden station problem among wireless devices. Figure 7.1 shows the node setup for this experiment and Table 7.1 shows the parameters used in the experiment. Since no carrier-sensing schemes are used in the data link layer, there are not RTS or CTS packets transmitted between nodes. Instead, packets transmit when they are ready to transmit. In this experiment, as node USRP 0A is transmitting to USRP1A, USRP 0B starts to transmit to USRP 1A as well. The trials were conducted through the use of the application layer GUI to control the nodes and repeat this scenario fifty times. In traditional Wi-Fi networks, the collision would result in USRP 1A not receiving any messages since it would be unable to decode the corrupted signal caused by the simultaneous transmission. Instead, in this experiment, USRP 1A decoded the message from USRP 0A successfully through 50 trials. Through the use of the BBC-enabled protocol stack, the hidden station problem is solved.

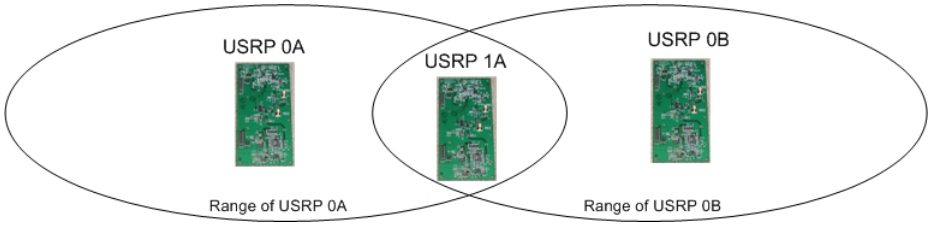

Figure 7.1: Hidden Station Problem Experimental Setup with USRP Devices

Parameter	Value
Frequency	1200MHz
Decimation Factor (RX)	64
Interpolation Factor (TX)	128
RX Time	10 sec
TX Time	10 sec
Trials	50
Successful Receptions	50

Table 7.1: Hidden Station Experiment Parameters

7.3 Exposed Station Problem Experiment

In this experiment, the main goal is to prove a solution to the common exposed station problem that causes loss of throughput in wireless networks. Figure 7.2 shows the node setup for this experiment and Table 7.2 shows the parameters used in the experiment. As previously mentioned, the data link layer does not contain carrier sensing medium access controls. As a result, nodes transmit when they are ready to transmit messages instead of asking permission beforehand. Under this experiment, nodes are exposed to many transmissions and are forced to decode messages despite numerous collisions. Specifically, USRP 0B transmitted a message while USRP 1B transmitted a message. Simultaneously, USRP 0A and USRP 1A were in receive mode so they could receive any available transmissions. The application layer GUI is used to control this scenario and conduct the trials. In every trial of the 50 total trials, the receiving nodes decoded messages. The messages were not always intended for them, but messages were received successfully on every trial. In other words, USRP 0A would decode messages from USRP 0B in some trials and in USRP 1B in other

trials. Which message it received is related to which message the decoder encountered first in the received signal data and which it could assemble completely without any retransmissions. To explain this further, the receiver at USRP 0A is on a timer and stops receiving after the specified amount of time. Then, the decoder determines what messages have been received. Consequently, USRP 0A may not receive complete messages from USRP 0B and USRP 1B. Therefore, the receiver decodes messages from USRP 0B in some trials and USRP 1B in other trials. Given that the collisions between the two simultaneous message transmissions did not corrupt the reception of at least one message at the receiver and there is no carrier sensing that prevented the transmission, the exposed station problem did not arise. For these aforementioned reasons, the exposed station problem is solved through the use of the BBC-enabled protocol stack.

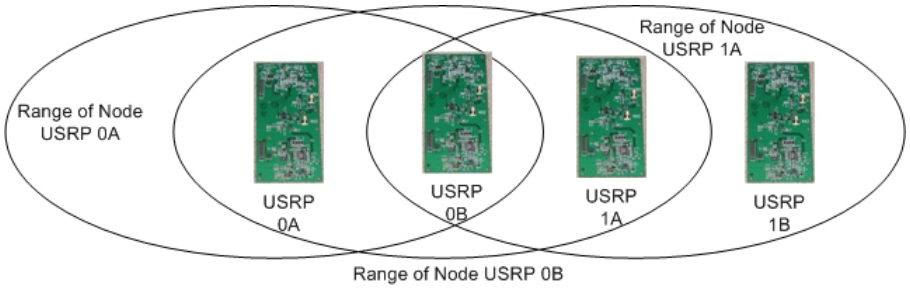

Figure 7.2: Exposed Station Problem Experimental Setup with USRP Devices

Parameter	Value
Frequency	1200MHz
Decimation Factor (RX)	64
Interpolation Factor (TX)	128
RX Time	10 sec
TX Time	10 sec
Trials	50
Successful Receptions	50

Table 7.2: Exposed Station Experiment Parameters

7.4 Chapter Conclusion

The focus of this chapter is on the hidden and exposed station problems that commonly occur when wireless nodes are densely populated. Migrating away from carrier-sensing medium access schemes is necessary in any environment where there exists significant spectrum interference or potential adversarial jamming sources. With this in mind, the BBC protocol stack developed in this research does not incorporate any carrier-sensing techniques. It should be clarified that the experiments conducted in this chapter involve only the physical and data link layers of the BBC-enabled protocol stack. When collisions occur or noise affects the received signal in other ways, the BBC algorithm compensates for packet collisions with error correcting codes. When collisions occur as nodes transmit simultaneously, the algorithm is able to correct for this and allow wireless nodes to correctly decode messages. This chapter presents a solution to two major problems that plague wireless nodes in today's world where ubiquitous wireless communications are becoming ever more popular.

Chapter 8

Phase II Experimental Results

8.1 Chapter Introduction

In Phase II, an experiment is devised to compare traditional 802.11G [IEEE 2007] wireless communications with the BBC-enabled protocol stack developed in this research. Specifically, the goal of the experiment is to measure the tolerance to interference in the spectrum for each protocol and make a comparison between the two protocols in terms of node availability. In theory, 802.11G should not tolerate jamming sources as well as the BBC-enabled protocol due to its lack of sufficient error correction coding and its contention-based data link layer. Conversely, the BBC-enabled protocol should operate effectively in high levels of spectrum interference due to the jam-resistant encoding scheme and a non-contention based data link layer. The experimental results described in this chapter demonstrate unequivocally that a BBC-enabled protocol stack resists jamming more effectively than 802.11G and guarantees node availability in the presence of interference where nodes using the 802.11G protocol simply fail. The upcoming sections in this chapter will discuss the experiment setup for the two protocols and provide a comparison between the two in the presence of varying levels of spectrum interference.

8.2 Wi-Fi 802.11G Experimental Setup

The goal of this part of this experiment is to collect data on an 802.11G based wireless system to facilitate making a comparison to the system running the BBC protocol created in this research. The experiment involves transmitting 1000 messages between two 802.11G nodes and determining the message delivery success rate. In the first scenario, the messages will be transmitted without a signal jammer adding noise to the spectrum. In the second scenario, a jammer will add noise to the spectrum while the nodes attempt to communicate. Various levels of noise will be used during the second scenario for the sake of comparison. The first scenario serves as a control group so that the differences between the noiseless and noisy environments become apparent. In an effort to reduce external interference sources that may affect the outcome of the experiment, the physical environment for this experiment is a clear grassy field away from visible sources of electronic interference. At the experiment site, no Wi-Fi hotspots were found using Wi-Fi network location software built into a laptop computer.

The noise generator for the 802.11G protocol used in this experiment is a conventional microwave oven operating at the 2.4GHz frequency which is shared with 802.11G devices. The average consumer can test this jamming source effectively by placing their Wi-Fi devices in close proximity to their microwave oven. However, a more technical discussion of this interference is necessary. The residential microwave oven contains a single magnetron that periodically turns on and off as the 60Hz AC

line voltage oscillates from positive to negative as shown in Figure 8.1. The magnetron generates RF signals to cook food at in the 2.4GHz band and operates in a similar fashion to Frequency Modulated (FM) signals [Taher et al. 2006]. This RF energy from the magnetron leaks from the case of the microwave and causes interference with 802.11G devices. The strength of the signal emitted varies with the power used in the microwave. Therefore, a microwave with 10 different power settings will emit 10 different signals of varying power. Essentially, the microwave oven acts like any jammer and adds power to the spectrum about a certain carrier frequency [Taher et al. 2008] and prevents an eligible receiver from correctly decoding the received signal. The specific center carrier frequency varies by model, but the microwave used in this experiment has a center frequency of 2.45GHz. The idea of using a microwave oven to test 802.11G networks' tolerance for interference and jamming has been studied before by [Pietikainen et al. 2005], [Kamerman and Erkocevic 1997], [Taher et al. 2006].

In this experiment, two 802.11G nodes were positioned 10-feet from a microwave oven jamming source on opposite sides of the jammer as shown in Figure 8.2. The experiment was conducted outside in an open field as mentioned above and the power was supplied via an automobile nearby equipped with a DC-to-AC converter. The oven used is capable of 10 different power settings and therefore 10 different jamming signal strengths. The nodes exchanged UDP messages in an echo-echo reply message exchange pattern and the results were logged for offline analysis. Figure 8.3 below

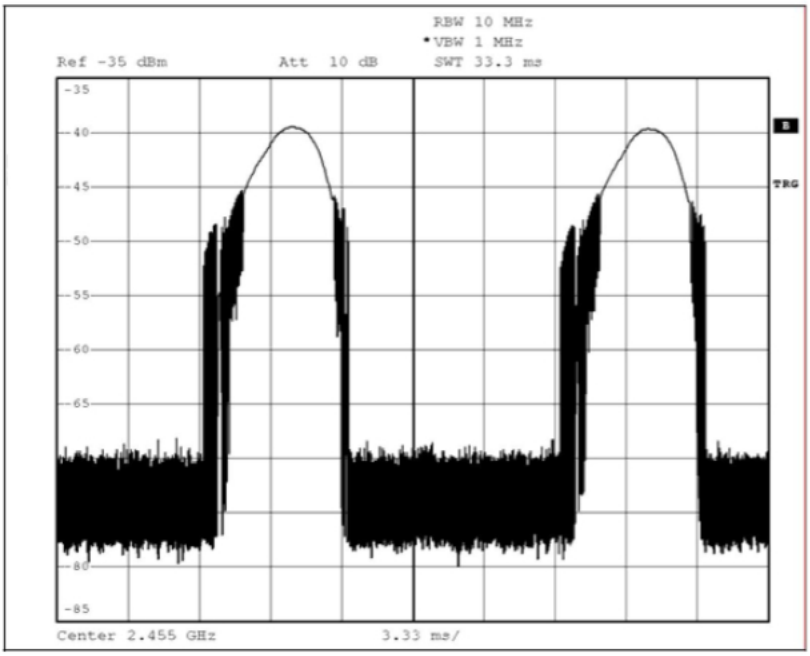

Figure 8.1: Microwave Oven Signal in Time Domain [Taher et al. 2006]

represents the data taken during the experiment where 1000 messages were sent and acknowledged.

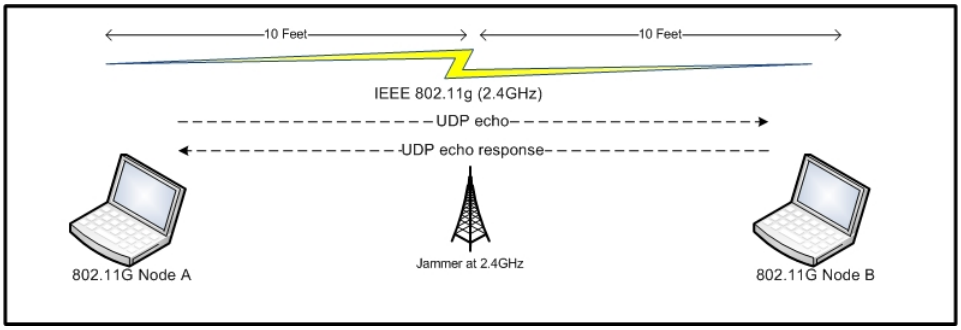

Figure 8.2: 802.11G Experiment Setup

A packet error rate greater than 100% indicates that a message had to be retransmitted multiples times. Since the BBC-enabled data link layer uses acknowledgements

Jamming Power	Messages Transmitted	Message Retransmissions	PER
10	1000	2530	253%
9	1000	2350	235%
8	1000	2190	219%
7	1000	1950	195%
6	1000	1740	174%
5	1000	1510	151%
4	1000	1290	129%
3	1000	1120	112%
2	1000	880	88%
1	1000	760	76%
0	1000	160	16%

Figure 8.3: Wi-Fi 802.11G Experiment Data

to confirm the receipt of messages and trigger retransmissions if necessary, the messages exchanged in the 802.11G experiment were also acknowledged and retransmitted until the message arrived successfully. The microwave generates signals in a periodic "on" and "off" state, so it is quite likely that messages were retransmitted during an "off" state and therefore the delivery was successful. At high power levels, the "on" state lasts longer than the "off" state and thus more energy is added to the spectrum. For example, at a 50% power level, the "on" state splits equal time with the "off" state. In a cooking example, if a meal is put in the microwave for 2 minutes on 50% power, then only 1 minute of that time will the food receive microwave energy. The food is cooked slower as the microwave alternates "on" and "off" states and the food cools during the "off" states. In the case of an adversarial jammer, the jammer would likely output power constantly and with the maximum energy possible. This type of jammer would consume massive amounts of energy and would result in an 802.11G device not being able to communicate any packets successfully. Clearly, this experiment shows that 802.11G devices do not operate effectively in environments with significant spectrum interference.

8.3 BBC-Enabled Protocol Experimental Setup

The goal of this part of the experiment is to collect data using the USRPs and the BBC-enabled protocol layers developed in this research in order to make a comparison to popular 802.11G protocol devices. As with the previously discussed 802.11G experiment, this experiment involves transmitting 1000 messages between two USRPs with RFX-1200 daughterboards to determine the message delivery success rate. In the first scenario, the messages will be transmitted without a signal jammer adding noise to the spectrum. In the second scenario, a jammer will add noise to the spectrum while the nodes attempt to communicate. In a similar approach to the 802.11G experiment, varying levels of noise will be used during the second scenario to gather data useful in comparing resistance to jamming. The first scenario serves as a control group so that the differences between the noiseless and noisy environments is evident. In an effort to reduce external interference sources that may affect the outcome of the experiment, the physical environment for this experiment is a clear grassy field away from visible sources of electronic interference such as radio towers. At the experiment site, an RSSI measurement is taken to determine the environmental interference present for the control group and no significant noise is discovered. The experiment is conducted using a frequency of 1.2GHz which is within the capabilities of the RFX-1200 daughterboard.

To generate the noise, the signal generator script discussed in the software components portion of Section 6.2 is used. As a parameter, the noise generator takes a

value to indicate the amplitude of the signal to generate. There is a direct correlation between the signal amplitude and the signal strength. Therefore, a larger amplitude value will cause a more powerful signal to be generated. The signal into the USRP is a 16-bit signed number where 32767 represents the full scale amplitude possible given the hardware. Using this maximum as the highest level of noise, the experiment is conducted with various levels of noise offset from this maximum value. Table 8.1 shows the notional power level in conjunction with the amplitude value supplied to the jammer and the perceived noise level given by the RSSI measurement from the auxiliary analog-to-digital converter (ADC). The RSSI measurement is taken at the intended receiver of the message and averaged over the trials. The purpose of the RSSI measurement is to provide confirmation that the jammer is working as specified and the jamming signal is indeed being modulated by the USRP as desired.

In this experiment, two USRP nodes were positioned 10-feet from the host computer on opposite sides of the host as shown in Figure 8.4. As with the previous experiment with 802.11G, this experiment was conducted outside in an open field and the power was supplied via an automobile nearby equipped with a DC-to-AC converter. The USRP nodes exchanged messages in an echo-echo reply message exchange pattern and the results were logged for offline analysis. Figure 8.5 shown below represents the data taken during the experiment where 1000 messages were sent and acknowledged. The nodes in this experiment transmitted their messages for 10 seconds and the receivers only received data for 10 seconds. Varying this parameter is possible, but for consistency and reducing variables in the experiment, 10 seconds on

both the transmitter and the receiver worked effectively. The nodes alternate between the transmit mode and receive mode since a single daughterboard of the RFX-1200 series cannot be in full duplex operation simultaneously on the same frequency.

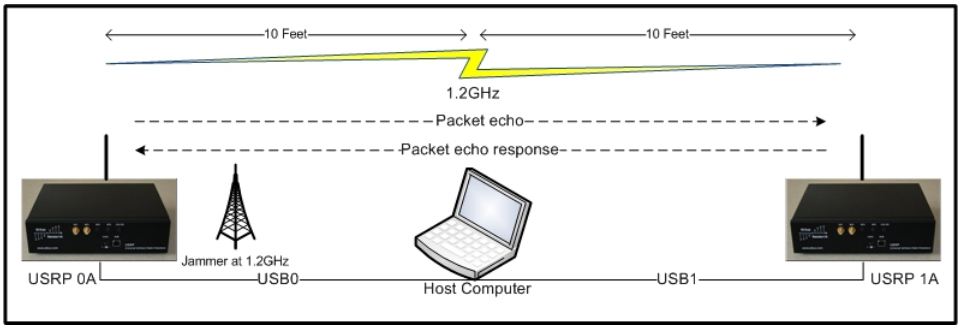

Figure 8.4: BBC-enabled Protocol Experiment Setup

Jamming Power	Messages Transmitted	Message Retransmissions	PER
10	1000	41	4%
9	1000	32	3%
8	1000	31	3%
7	1000	23	2%
6	1000	10	1%
5	1000	9	1%
4	1000	0	0%
3	1000	0	0%
2	1000	0	0%
1	1000	0	0%
0	1000	0	0%

Figure 8.5: BBC Experiment Data

From the data collected in this experiment, the conclusion is that a BBC-enabled protocol stack is highly resistant to jamming due to its message encoding. Even with maximum power from the jammer, messages were rarely corrupted and few retransmissions were necessary.

Jamming Power	Noise Signal Power (Amplitude)	RSSI [0,4095]
10	32000	4081
9	28000	4065
8	24000	3939
7	20000	3424
6	16000	2411
5	12000	2116
4	8000	1507
3	4000	542
2	2000	180
1	1000	163
0	0	155

Table 8.1: Signal Jamming Power Correlated to RSSI

8.4 Wi-Fi 802.11G at 1.2GHz Experimental Setup

The goal of this experiment is to solidify a comparison between the 802.11G protocol and the BBC-enabled protocol stack developed in this research. As with the previous experiments, this experiment involves transmitting 1000 messages between two USRPs with RFX-1200 daughterboards to determine the message delivery success rate. In the first scenario, the messages will be transmitted without a signal jammer adding noise to the spectrum. In the second scenario, a jammer will add noise to the spectrum while the nodes attempt to communicate. As in the BBC experiment above, varying levels of noise will be used during the second scenario to gather data useful in comparing resistance to jamming. Moreover, the noise source is the same jammer used in the previous experiment for BBC. The first scenario serves as a control group so that the differences between the noiseless and noisy environments is evident.

In this experiment, two USRP nodes were positioned 10-feet from the host computer on opposite sides of the host as shown in Figure 8.6. As with the BBC experiment discussed previously, these nodes operate at 1.2GHz. The data for this experiment consists of random 1500-byte 802.11G frames captured from the host computer's Wi-Fi connection. These 802.11 packets are used in order to determine how varying levels of noise affects them and to make a comparison to the impact of noise on the packets of the BBC-protocol stack.

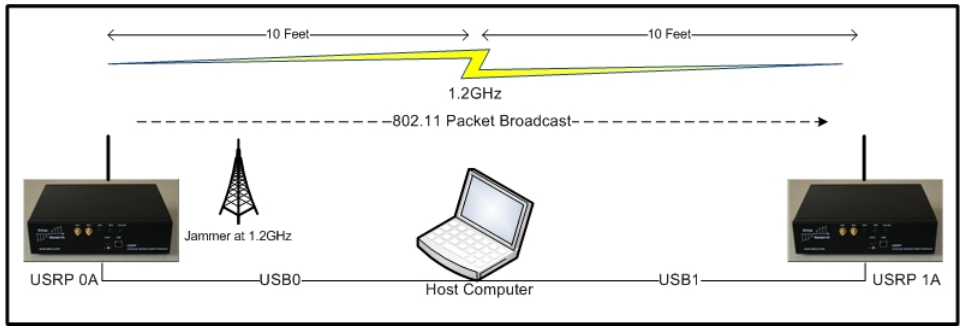

Figure 8.6: 802.11G at 1.2GHz Experiment Setup

Jamming Power	Messages Transmitted	Messages Received	PER
10	1000	0	100.00%
9	1000	0	100.00%
8	1000	0	100.00%
7	1000	0	100.00%
6	1000	0	100.00%
5	1000	0	100.00%
4	1000	0	100.00%
3	1000	64	93.60%
2	1000	364	63.60%
1	1000	557	44.30%
0	1000	998	0.20%

Figure 8.7: 802.11G at 1.2GHz Experiment Data

From the data collected in this experiment, clearly the 802.11G packets are highly susceptible to corruption in the presence of jamming. The packets only contain a 32-bit cyclic redundancy check (CRC) to alert the protocol that there is an error in the packet. The 32-bit CRC is an error-detecting code and is not able to correct multiple bit errors in the same packet. In contrast, the BBC encoding is an error-correcting code which is able to successfully correct for multiple bit errors in a message. In this experiment, the receiver computed CRCs for every packet to determine if there was an error. With any level of noise from the jammer, the CRC for all 1000 messages detected an error. On an actual 802.11G device, any CRC mismatch automatically triggers a retransmission of the packet, so this would be quite devastating from a performance standpoint. As the data in Figure 8.7 shows, the receiver was able to decode messages with jamming power below 4, but with higher levels of jamming the receiver was unable to decode any messages. At a jamming power level of 3, the packet error rate rose to 93.6%. Clearly, the 802.11G protocol is unable to successfully operate in areas with significant noise.

8.5 Comparison of Wi-Fi 802.11G and BBC-Enabled Protocol Stack Experiments

The data collected clearly demonstrates that 802.11G does not operate effectively in the presence of noise. Figure 8.8 shows a comparison of the data collected in the 802.11G experiment at 2.4GHz to the BBC experiment at 1.2GHz. In addition, Figure 8.9 shows a comparison of the data collected in the 802.11G experiment at

1.2GHz to the BBC experiment at 1.2GHz. In the 802.11G experiment with the microwave as the jammer, only the lowest power settings on the jammer allowed messages to be received on the first attempt. With jamming power above level 2, all packets transmitted would have been lost due to error if not for the retransmission capability provided by the data link layer. In the 802.11G experiment at 1.2GHz, the receiver did not receive any messages past jamming level 3. In comparison, the BBC-enabled protocol stack operates effectively in even the highest levels of noise. With the jamming source at maximum power, a mere 4% of packets had to be retransmitted due to errors.

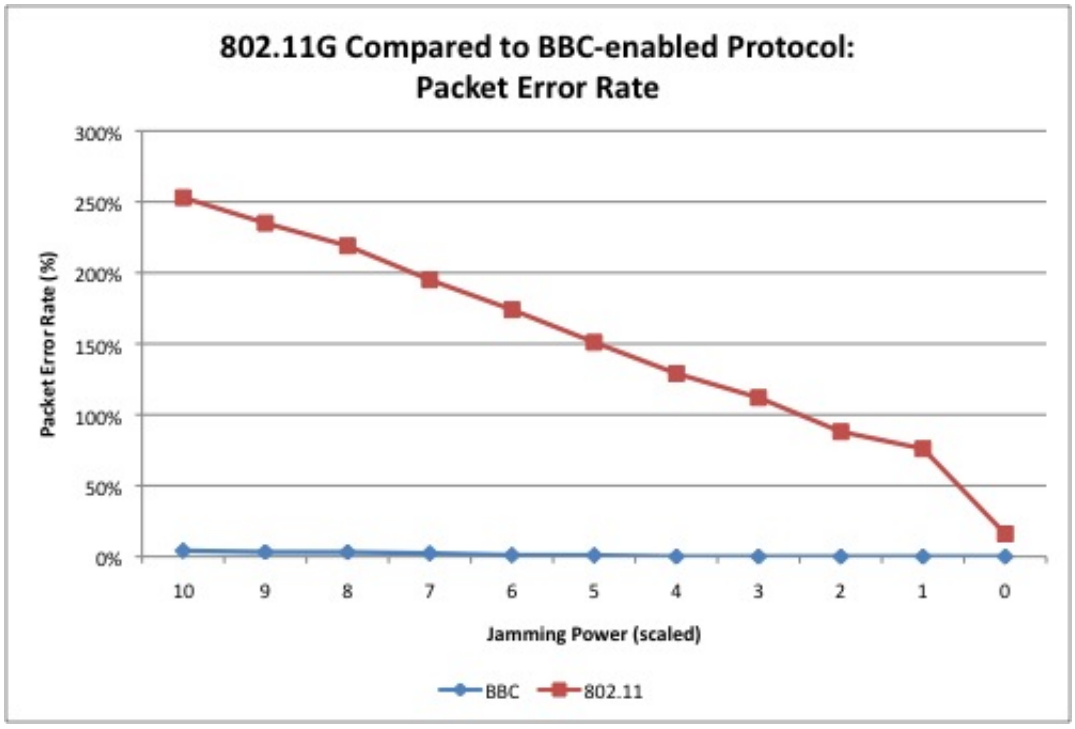

Figure 8.8: 802.11G Versus BBC Protocols: Packet Error Rate

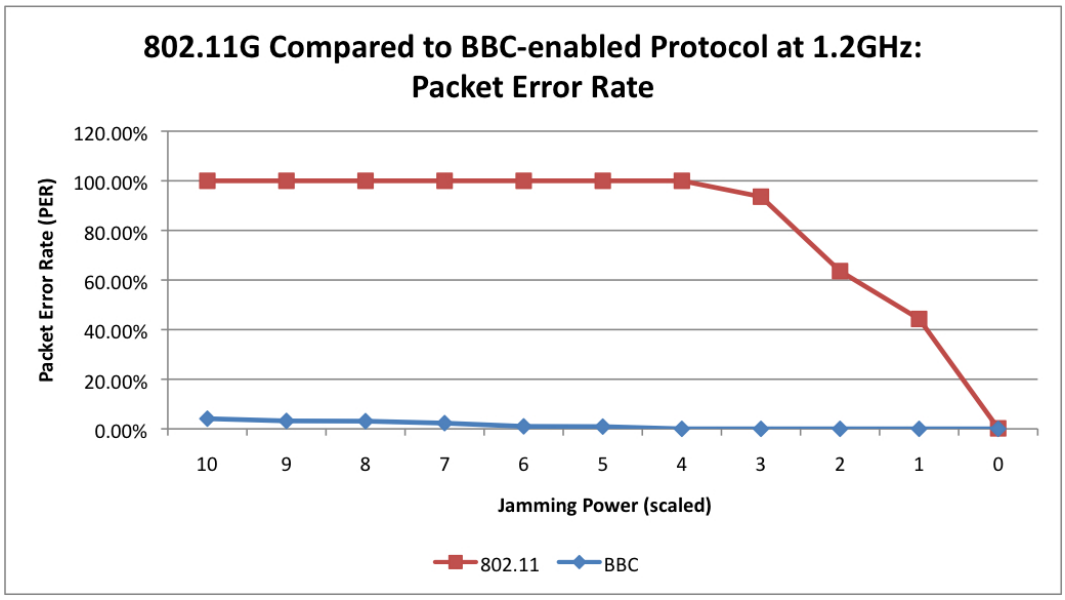

Figure 8.9: 802.11G at 1.2GHz Versus BBC Protocol: Packet Error Rate

From these experiments, the BBC-enabled protocol developed in this research has shown significantly improved node availability over the popular Wi-Fi 802.11G communications protocol. In disaster or battlefield scenarios, node availability is of the utmost importance and this capability is provided by the BBC-enabled protocol stack.

8.6 Chapter Conclusion

Clearly, improvements are necessary to the 802.11G protocol and encoding scheme in order to prevent jamming in this increasingly crowded spectrum. Common consumer appliances should not interfere with wireless networks in such a dramatic way. Even while operating in an uncrowded spectrum at 1.2GHz, the 802.11G protocol did not perform well in the presence of noise. 802.11G's poor tolerance to spectrum

interference precludes its use in environments with significant noise such as the battlefield or even densely populated areas with many 802.11G devices. Channelization does help mitigate some of the interference with 802.11G devices; however, a robust jammer or one that operates with a wide-bandwidth such as the microwave will causes 802.11G devices to stop working due to bit errors at the receiver [Pietikainen et al. 2005]. The BBC-enabled jam-resistant protocol developed in this research is a step toward wireless devices that tolerate jamming and can guarantee node availability in environments with significant spectrum noise or intentional jamming.

Chapter 9

Phase III Experimental Results

9.1 Chapter Introduction

In Phase III of this research, the network routing layer implementation is tested on the USRP software radio platform. The BBC-OLSR protocol combines the strengths of both OLSR and BBC to create a jam-resistant network layer with effective routing in the presence of noise. Unlike other network protocols, this network layer will operate in noisy environments due to the jam-resistant BBC encoding taking place at the data link layer. Furthermore, BBC-OLSR improves the reliability of OLSR since it will not lose as many messages to jamming or interference. Network availability is the primary concern for this protocol since it is designed to operate in environments with significant spectrum interference.

As stated in the implementation discussion of Chapter 6, BBC-OLSR analyzes the RSSI measurements taken at each node prior to the transmission of a message. Therefore, when it comes time to assign a message a path to take in the network, the network layer agent has interference information available for each hop along the route. Using this knowledge, BBC-OLSR instructs the data link layer to use a certain level of jam resistance in the BBC encoding.

Currently, there are five levels of jam-resistance corresponding to four codeword lengths to be used in the BBC algorithm as shown in Table 9.1 below. BBC-OLSR

determines the appropriate level to use based on the scheme discussed earlier to ensure delivery to the node with the greatest amount of interference present along a certain path. The implementation of the BBC encoding and decoding with the various levels of jam-resistance is still under development via a parallel research effort into a new data link layer specifically created for this purpose. Until this is implemented, the encoding and decoding of the various levels is not testable, however this is a data link layer function and not a network layer issue. The BBC-OLSR network layer is concerned with the use of the RSSI measurements collected to ensure adaptation to interference in the network as a whole.

Jam-Resistance Level	**Codeword Length**	**RSSI Range**
5	160	3500-4092
4	128	2500-3500
3	96	1500-2500
2	64	750-1500
1	32	0-750

Table 9.1: Threshold Levels for Jam-Resistance Based on RSSI Measurements

The BBC-OLSR feature of using RSSI measurements to make adaptive routing decisions is tested and proven in this chapter. In the remainder of this chapter, the experiments with OLSR and BBC-OLSR are discussed. As expected, the chapter concludes that BBC-OLSR allows nodes in the network to adapt to varying levels of noise throughout the network. Furthermore, the BBC-OLSR protocol ensures node availability with message delivery where the original OLSR protocol simply fails in the presence of noise.

9.2 Experiment 1: USRPs Only

The purpose of this experiment is to test the effectiveness of this BBC-OLSR protocol on the software defined radios. Many networking protocols are constructed and only tested in a simulated environment such as NS-2 (refer to Chapter 3), but it is important to this research for this protocol stack to operate well on actual hardware radios. Delays due to encoding, decoding, transmission, and reception become more important with real hardware and are not usually discovered in the design phase. In addition, signaling issues such as timing and synchronization with waveforms do not become apparent without the use of real hardware. For these reasons, this protocol stack from the application layer to the physical layer is tested in this experiment on the four USRP devices on hand.

9.2.1 Experiment Setup

In order to make a comparison between the OLSR and BBC-OLSR protocols, there is a shared experimental setup between the two. First, there will only be four nodes in this experiment as shown in Figure 9.1. Each node is an RFX-1200 daughterboard and the nodes are mounted on two USRPs containing two separate FPGAs as discussed in previous chapters. The nodes will be distributed as far apart as possible similarly to the experiments conducted in Phase II. The data link and physical layers used by each network layer protocol will be the ones developed in this research and described in the preceding chapters. Therefore, the only experimental variable is the network layer being used. During the experiments, the application layer will

drive the exchange of messages in the network and provide a visualization of the protocol's operation. In the experiment with noise, the noise generator script will be used to add interference at the 1200 MHz operating frequency. The main goal of this experiment is to determine if the network layer successfully uses the RSSI measurements included in the transmitted messages to change the level of jam-resistance that it instructs the data link layer to use. As a secondary goal, the experiment should expose any differences in routing decisions between the two protocols in the presence of noise.

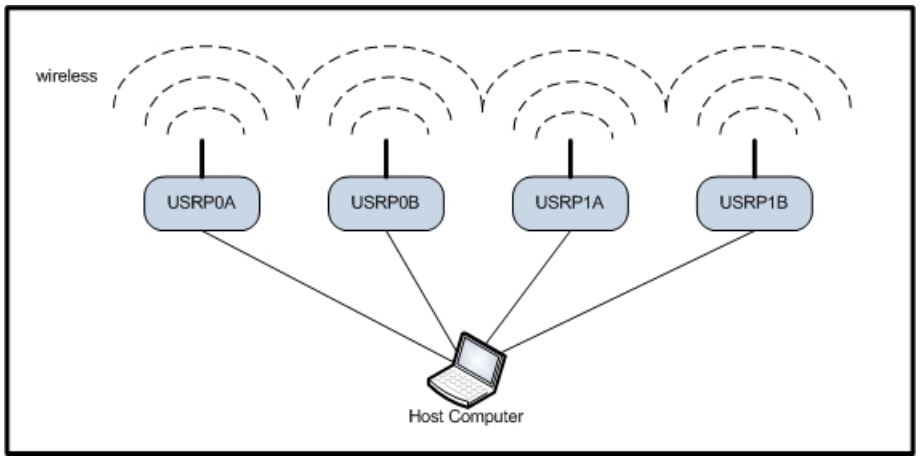

Figure 9.1: USRP Network Topology

9.2.2 OLSR vs. BBC-OLSR Without Interference

Without interference, the protocols behave in an almost identical manner. BBC-OLSR's threshold for increasing the level of jam-resistance is not triggered unless there is an interference source nearby. As expected, both protocols choose the same routes and receive all of the messages that are transmitted when there is no mobility. Since

BBC-OLSR bundles control messages into groups of ten messages before encoding and transmitting them, it does not build up as large of a transmit queue. Furthermore, it allows nodes running BBC-OLSR to receive more timely updates to topology changes since more control messages arrive at a time. By the time the original OLSR protocol is finished encoding a control message (1.5 sec) and transmitting (0.5 sec), there are already on average 5 more messages in the transmit queue if the broadcast interval is 0.5 sec. Mobility is simulated in this experiment by dropping packets from a specific node on the host computer so that the routing agent does not receive messages from that node. When this occurs in the experiment, BBC-OLSR removes the node from any routes faster since it bundles the control messages. This experiment provides a good control study to show the problems noise can cause in wireless communications since it is a straightforward comparison of the two protocols without any message corruption due to interference. The data for this experiment is shown in Figure 9.2 for OLSR and Figure 9.3 for BBC-OLSR in the row where the jamming power is zero.

9.2.3 OLSR vs. BBC-OLSR With Interference

In this experiment with the USRPs, noise is added to the spectrum through the use of the signal generator script to add random noise. Adding power to a signal will flip 0's to 1's and prevent the successful decoding of messages. At the network layer, the main concern is that noise will corrupt control messages and routes will become stale. To avoid this, BBC-OLSR uses the RSSI values sent from nodes in the network to adjust the message's BBC encoding to improve reliability. To simulate

routes changing due to mobility, packets are dropped for certain nodes on the host computer.

In this experiment, OLSR did not use the RSSI value at all. OLSR's individual control messages became corrupted by the interference because they were only 20 bytes in length. With a 20 byte message, the decoder has a difficult time distinguishing the noise signal and the actual signal since the received signal is about 150Mb worth of IQ pairs. At RSSI values over 3100, OLSR failed to decode any messages and all of the routes became stale.

In contrast, BBC-OLSR continued to operate even in the highest levels of jamming (RSSI over 4000) since the messages were bundled together to form a longer message. A longer message meant that more of the received signal is message content and not noise. For all practical purposes, some of the message bits were probably made stronger with the addition of noise by mere coincidence. As an added benefit, the routes were updated more frequently than pure OLSR as discovered in the previous experiment. The RSSI values from the nodes on a path were used to trigger thresholds for the levels of jam resistance. The greatest RSSI value along a route is used to determine the level of jam-resistance to use for maximum availability. As expected, the data link layer successfully received the threshold value prior to encoding the messages and could therefore effect the change prior to transmission. With the level of jam-resistance assigned, the encoder will use a different codeword length in response. The decoders at the receiver side do not need modification since they will search for the proper codeword length if necessary during the decoding process.

The following data tables in Figures 9.2 and 9.3 illustrate a sample run for noise added to USRP nodes running OLSR and BBC-OLSR respectively. In addition, a comparison chart is shown in Figure 9.4.

Jamming Power	Messages Transmitted	Message Retransmissions	PER
10	120	120	100%
9	120	114	95%
8	120	98	82%
7	120	93	78%
6	120	81	68%
5	120	66	55%
4	120	59	49%
3	120	45	38%
2	120	37	31%
1	120	9	8%
0	120	0	0%

Figure 9.2: OLSR With Interference Data

Jamming Power	Messages Transmitted	Message Retransmissions	PER
10	120	7	6%
9	120	5	4%
8	120	3	3%
7	120	2	2%
6	120	1	1%
5	120	0	0%
4	120	0	0%
3	120	0	0%
2	120	0	0%
1	120	0	0%
0	120	0	0%

Figure 9.3: BBC-OLSR With Interference Data

From this experiment, it is concluded the BBC-OLSR operates more reliably in the presence of noise than pure OLSR. BBC-OLSR enabled nodes to receive topology updates quicker and resist higher levels of jamming more effectively. Furthermore, BBC-OLSR's determination for the best level of jam-resistant encoding operates effectively on real hardware and in the presence of real signal jammers.

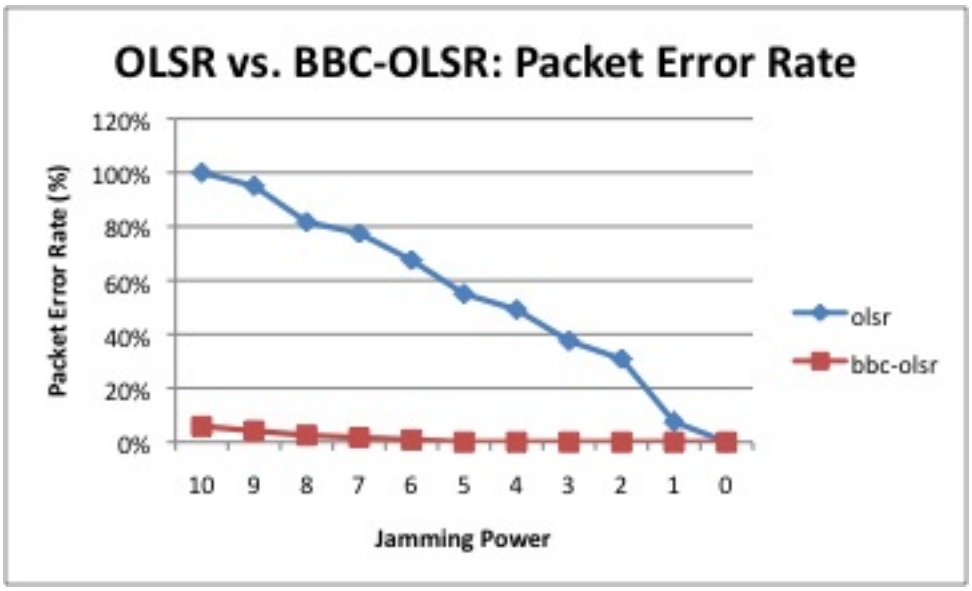

Figure 9.4: OLSR vs. BBC-OLSR Chart

9.3 Experiment 2: USRPs and Virtual Nodes

In order to test the protocol stack's ability to handle more dispersed nodes due to mobility and scale to a larger number of nodes, more than four nodes are necessary. For this experiment, the four real radios are being used in conjunction with six simulated nodes running on virtual network interfaces. To identify the nodes in this discussion, the real radios are identified by *USRP0A*, *USRP0B*, *USRP1A*, and *USRP1B*. The virtual nodes are identified by *virt1* through *virt6*. The BBC protocol stack used in the previous experiment will also be used in this experiment, except the six virtual nodes will be using virtual network interfaces instead of the USB. The topology for this experiment is shown in Figure 9.5 below.

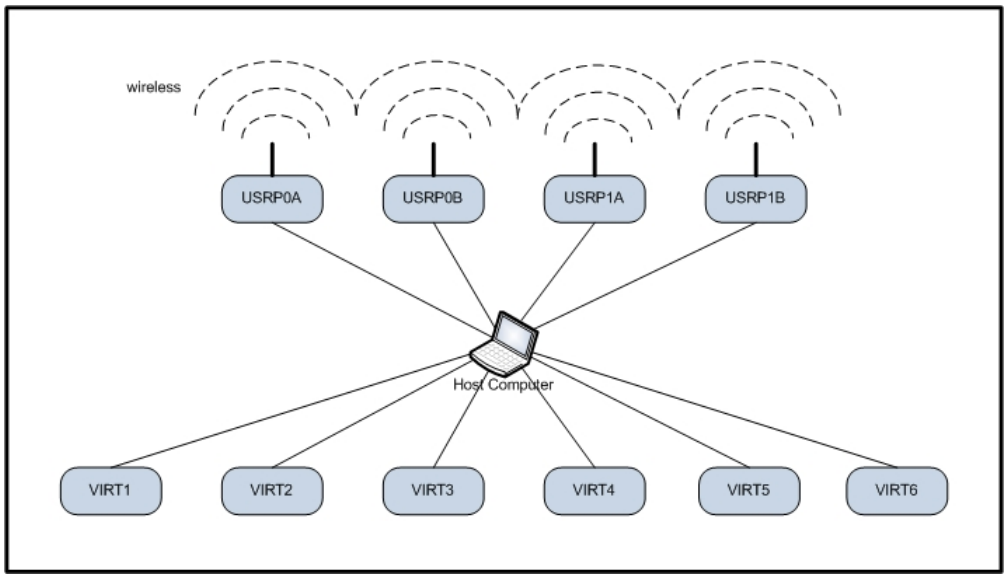

Figure 9.5: USRPs and Virtual Nodes Network Topology

9.3.1 Experiment Setup

The main purpose of this experiment is still to make a comparison between pure OLSR and BBC-OLSR at the network layer. By using ten nodes, instead of four, the addressing scheme and routing issues will become more pronounced. Although six of the nodes are operating in software, they are still using virtual network interfaces that act like real hardware. The host computer does not distinguish between the virtual interfaces used and the hardware interfaces since it is just a file descriptor object to the operating system. The main difference between the two protocol stacks remains the network layer and this is where a comparison between OLSR and BBC-OLSR is made. The noise used will be from the actual radios and from the virtual nodes as well. In addition to simulating mobility, dropping packets of a certain node can simulate noise at that point in the network as well. An example of node loss

due to simulated mobility or noise is shown in Figure 9.6 where packets are dropped for the disconnected nodes. Random RSSI values are reported to the BBC-OLSR protocol when a node's packets are being dropped due to a simulated noise event. From this, the protocol determines the most suitable level of jam-resistance for the communications path.

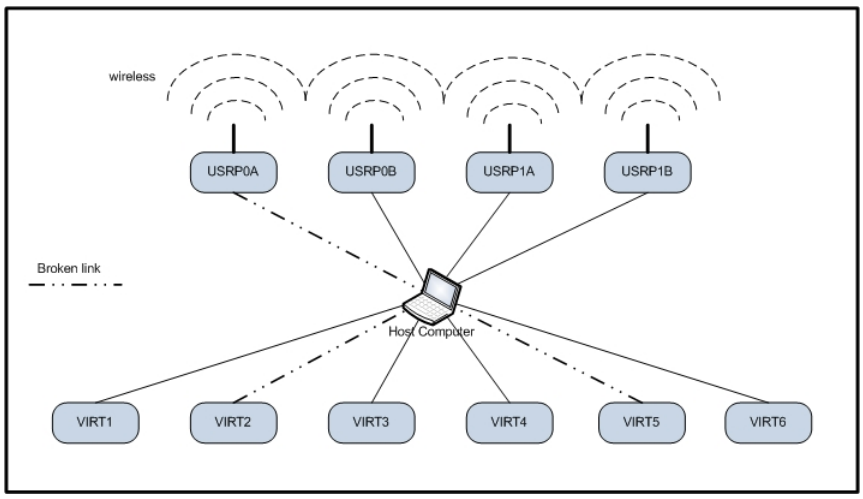

Figure 9.6: USRPs and Virtual Nodes Network Topology with Broken Links

With more than a few nodes in the network, addressing may become an issue. All of the nodes carry randomly assigned addresses. In this protocol stack, addresses are IP-based at the network layer, but at the data link layer they are randomly assigned integers. An address translation is completed between the two layers to ensure that the non-IP based addressing scheme is used over the wireless link for efficiency. As discussed in Chapter 6, address collisions are extremely rare but are resolved effectively.

The following sections discuss the experimental results of the network layer comparison with and without interference.

9.3.2 OLSR vs. BBC-OLSR Without Interference

In this experiment, the ten nodes are brought online in an environment without interference. The nodes communicate with each other through the topology control and "hello" messages successfully under both protocols. Ordinarily, there would be message collisions with so many messages being exchanged in the network. However, there are transmit and receive message queues at the data link layer which allow a node to queue messages for processing. The pure OLSR protocol consistently builds up hundreds of packets in the receive and transmit queues because it generates messages every 0.5 seconds on the default broadcast interval. On the other hand, BBC-OLSR combines ten messages at a time and delivers them all at once. So while the messages are larger, they are decoded and analyzed faster because there is less switching time between messages. With the simulated mobility, the results are the same as the experiment with four nodes. BBC-OLSR received the updates faster than OLSR due to shorter queuing delays.

9.3.3 OLSR vs. BBC-OLSR With Interference

This experiment is similar to the previous experiment, except that two sources of noise are introduced to the network. One noise source is on a real radio and the other is on a virtual node. The virtual node's packets will be dropped by the operating

system kernel of the host computer when the noise source is active and nodes on either side of the virtual node will report higher RSSI values. For example, consider virtual node 4 is the jammer. Then virtual node 3 and 5 will report increased RSSI values to BBC-OLSR. The actual RSSI value reported does not necessarily matter as long as BBC-OLSR registers the change and modifies its jam-resistance level accordingly for the affected routes. For the real radios in the network, the actual RSSI values will be used as measured by the auxiliary ADC.

In this experiment, pure OLSR built up extremely large transmit queues of several hundred messages in just a few minutes. Although not directly related to the noise, this allowed routes to become stale since the significant queuing delays did not allow topology updates to be delivered in a timely manner. With a jammer active on a USRP node and a virtual node, the successful delivery of topology updates became sporadic. This delivery of corrupted messages from the radios increased the queuing delay since the decoder spent time attempting to decode a corrupted message (ending in failure) while the receive queue continued to build. In a similar fashion to Experiment 1, the small size of the messages allowed them to be easily corrupted. With the larger network, more messages traversed the network and exacerbated the topology update problems since the queuing delays were significant at each node and the message delivery was not reliable. When simulated mobility took down three of the virtual nodes by dropping their packets, the network routes did not stabilize due to the queuing problems created by the noise.

In contrast to the results of pure OLSR, BBC-OLSR excelled in the larger network size. The transmit or receive queues did not exceed fifteen messages on average at any time. Bundling the frequent but small topology control and hello messages together prior to encoding is an important aspect to the success of this protocol. The BBC algorithm needs time to encode and decode, and therefore it is important to make these steps proceed as efficiently as possible. Furthermore, the same experiments described above with pure OLSR were conducted with BBC-OLSR. With the two jammers active, topology updates were still successfully delivered due to the larger message size. Through observation, the RSSI values were properly reported from the USRP sensors as well as the random RSSI values from the virtual nodes. The threshold algorithm worked properly with several routes traversing the various nodes that each reported different RSSI values. The data link layer registered the level of jam-resistance assigned by the network layer and confirmed the encoding scheme selected. With simulated mobility, the packets of three virtual nodes were dropped as these nodes moved out of range. The routes in the network stabilized within thirty-seconds as sets of ten control messages were delivered successfully.

To further illustrate the results, Figures 9.7 and 9.8 show the results of a network scenario similar to the one described above. In the scenario represented below, a USRP node's packets are dropped to simulate it moving out of range and then it acts as a jammer to interfere with the transmissions from other nodes. Then, two virtual nodes move out of range and their packets are dropped. With noise added to the spectrum, BBC-OLSR messages are still decoded and the routing tables stabilize

properly. On the other hand, pure OLSR experiences problems in the decoding stage and consequently large transmit queues are built up which do not allow the routing tables to stabilize properly. Complete figures are quite large and must be shown in landscape mode. Therefore, these diagrams are included as Appendix Figures B.1 and B.2.

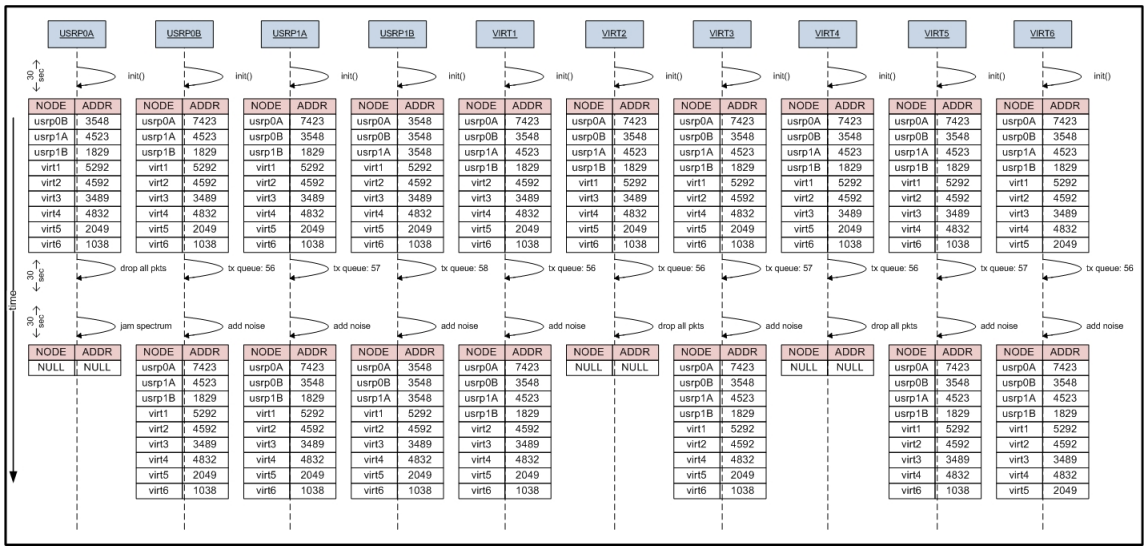

Figure 9.7: Pure OLSR With Noise Diagram

From these experiments, it is concluded the BBC-OLSR operates more reliably and scales better in the presence of noise than pure OLSR. BBC-OLSR enabled nodes receive topology updates more consistently as mobility increased. The higher levels of jamming were resisted by bundling messages together, but this also had the added benefit of reducing queueing delays for the protocol. Lastly, as discovered in the previous experiment with the four nodes, BBC-OLSR's determination for the best

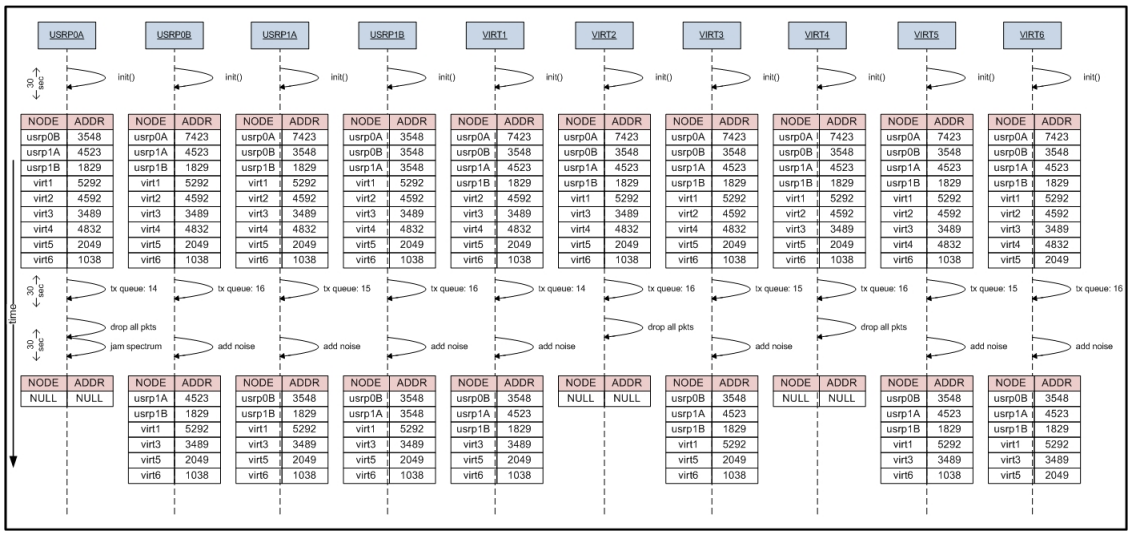

Figure 9.8: BBC-OLSR With Noise Diagram

level of jam-resistant encoding operates effectively on real hardware. It also operates effectively on virtual nodes and in the presence of message corruption.

9.4 Discussion of Results

The experiments conducted in Phase III are meant to validate the design of the adaptive jam resistant network routing layer that is part of the jam resistant protocol stack. Building on top of the physical and data link layers already described, the purpose of the network layer is to determine the level of jam resistance required to guarantee message delivery in the presence of interference. Each network route consists of nodes that are geographically distributed and this geographic dispersion may place some nodes in areas of interference. Furthermore, node mobility causes interference since signals become diffracted or reflected from environmental obstacles.

The purpose of this protocol stack is to ensure availability and message delivery in this type of noisy environment. Clearly, as proven by these experiments, a regular network layer protocol such as pure OLSR will not be effective with this encoding scheme due to the inherent delays of the algorithm.

9.5 Interference Adaptive Routing Discussion

As discussed in Chapter 3, the ALIR protocol [Zhang, Liu, Shi, Liu and Yu 2007] attempts to route data around noisy areas by extending the path length. Instead of this approach, BBC-OLSR adjusts the level of jam resistance used for message encoding and uses the same path despite interference levels. If the message is not acknowledged, another message is sent along a different path. In the mean time though, it is highly likely that topology control messages have already updated the route with a new path. However, in an infrastructure-less network with few nodes, such as the battlefield scenario shown in Figure 1.1 at the beginning of this dissertation, it is not always possible to find a route around the area of interference. For this reason, BBC-OLSR does not attempt to route around areas of interference and chooses instead to leverage the capabilities of the BBC algorithm to overcome noise in the spectrum.

9.6 Chapter Conclusion

The experiments conducted in Phase III described in this chapter focus on the effectiveness of the BBC-OLSR network layer implemented in this research. While

the focus is on the network layer in these experiments, the experiments provide a capstone testing event for the physical and data link layers as well. Most newly developed network protocols are only tested through simulation which may lead to protocols that do not operate well in real-world environments on actual hardware. In this research, the design and implementation of this jam-resistant communications protocol is validated on USRP software defined radios which adds some creditability to the implementation. The physical, data link, and network layers work cooperatively to provide a network protocol stack that resists jamming and provides node availability in the presence of spectrum interference where current network protocols fail.

Chapter 10

Key Contributions

- Created a cross-layer communications protocol for Mobile Ad-Hoc Networks (MANETs) that proactively determines routes for message delivery yet reactively adjusts to spectrum interference.

- Implemented a working prototype of a mobile jam-resistant communications system on software defined radios (USRPs) including physical, data link, and network layers.

- Constructed a collision-resistant addressing scheme involving the assignment of pseudo-random addresses and duplicate address detection mechanisms.

- Proved that the BBC algorithm can be used in conjunction with other protocol layers to provide network availability in the presence of spectrum interference or noise.

- Improved the reliability of the OLSR protocol by incorporating a global link quality determination and encoding adjustment based on data link layer feedback on the level of noise in the spectrum at the time of physical layer transmission.

- Provided a solution to the hidden station problem through the use of the BBC algorithm at the data link layer.

- Provided a solution to the exposed station problem through the use of the BBC algorithm at the data link layer.
- Improved the efficiency of the Optimized Link State Routing (OLSR) protocol by using a variable-length address space.
- Explored various ways to modify the BBC algorithm to compensate for variable amount of jamming. Further research with colleagues into this aspect of the protocol is ongoing.
- Explored various ways to modify the BBC algorithm to allow for variable length messages. Further research with colleagues into this aspect of the protocol is ongoing.
- Performed an extensive literature review on the BBC algorithm, MANET Network Layer Routing, and MANET Network Layer Addressing.

Chapter 11

Conclusion

The research described in this dissertation has crossed several fields of study including wireless communications, coding theory, signal jamming, mobile ad-hoc network routing, and mobile ad-hoc network addressing. In each of these domains, a detailed explanation of the previous work in the field has been presented. As any user of consumer wireless network devices can attest, current wireless networks are plagued by interference sources in their spectrum of operations. In mobile ad-hoc networks, the spectrum interference may be caused by some environmental or disaster situation. Or, it may be a result of adversarial jamming in a military communications example. Moreover, another common source of interference are the transmissions of nearby friendly stations. Currently, the solution to the interference problem is a human-based frequency deconfliction scheme where devices operating near each other are set to utilize different frequencies. These schemes are problematic at best and quite effort intensive on the part of engineers. In many cases, these deconfliction schemes involve agreements between the administrators of the various wireless systems to ensure unintentional jamming does not occur among cooperating systems. Clearly, a better scheme is needed to ensure unintentional jamming does not occur due to human error in these spectrum assignments.

This research effort creates a cross-layer protocol that can be deployed on many nodes and platforms to provide ad-hoc communications in these aforementioned situations. To date, there is not an adaptive jam-resistant scheme in existence that uses the approach described in this dissertation. The research described in this dissertation creates a network protocol stack that allows for jam-resistant communications in areas with high spectrum interference caused by either friendly station transmissions or adversarial signal jammers. Through the development of custom physical, data link, and network layers to form a jam-resistant communications protocol stack, the overall goal of creating an adaptive jam-resistant protocol has been achieved. The physical layer handles the signal processing, the data link layer ensures the delivery of messages hop-to-hop, and the network layer determines the best level of jam-resistance to employ to ensure network-wide node availability. Moreover, an application layer drives the rest of the layers in this working prototype. Unlike many other newly developed communications protocols, the prototype developed in this research operates on real hardware devices instead of a computer simulation. Furthermore, it has the capability to be deployed on many nodes in a self-organizing mobile ad-hoc network that can operate in environments where current 802.11-based protocols fail due to noise.

Bibliography

Adjih, C., Boudjit, S., Jacquet, P., Laouiti, A. and Muhlethaler, P. [2005], 'Address autoconfiguration in optimized link state routing protocol', *Internet Engineering Task Force (IETF) draft* .

Agrawal, D. and Zeng, Q.-A. [2006], *Introduction to Wireless and Mobile Systems*, Thomson Canada Limited, pp. 303–331.

Aron, I. D. and Gupta, S. K. S. [2001], 'On the scalability of on-demand routing protocols for mobile ad hoc networks: An analytical study', *Journal of Interconnection Networks* **2**, 5–29.

Ayanoglu, E., Chih-Lin, I., Gitlin, R. D. and Mazo, J. E. [1993], 'Diversity coding for transparent self-healing and fault-tolerant communication networks', *IEEE Transactions on Communications* **41**, 1677–1686.

Bahn, W. [2007], 'Bbc real-time engine', http://www.williambahn.com/bbc/.

Baird, L., Bahn, W. and Collins, M. [2007], Jam-resistant communication without shared secrets through the use of concurrent codes, Technical report, U.S. Air Force Academy.

Boleng, J. [2002], 'Efficient network layer addressing for mobile ad hoc networks', *Proceedings of the International Conference on Wireless Networks* pp. 271–277.

Boukerche, A. [2001], 'Performance comparison and analysis of ad hoc routing algorithms', *Proceedings of IEEE International Conference on Performance, Computing and Communications* pp. 171–178.

Burkhart, M., Rickenbach, P. V., Wattenhofer, R. and Zollinger, A. [2004], 'Does topology control reduce interference?', *Proceedings of ACM MobiHoc* pp. 9–19.

Castenada, R. and Das, S. [1999], 'Query localization techniques for on-demand routing protocols in ad hoc networks', *Mobile Computing and Communications Conference* .

Cha, H.-W., Park, J.-S. and Kim, H.-J. [2003], 'Extended support for global connectivity for ipv6 mobile ad hoc networks', *Internet Engineering Task Force (IETF) draft* .

Chakeres, I. and Perkins, C. [2008], 'Dynamic manet on-demand (dymo) routing', *Internet Engineering Task Force (IETF) draft* .

Chiang, C.-C., Wu, H.-K., Liu, W. and Gerla, M. [1997], 'Routing in clustered multihop, mobile wireless networks with fading channel', *IEEE Singapore International Conference on Networks (SICON'97)* pp. 197–211.

Chin, K.-W., Judge, J., Williams, A. and Kermode, R. [2002], Implementation experience with manet routing protocols, *in* 'ACM SIGCOMM Computer Communications Review', Vol. 32, ACM.

Chlamtac, L. and Lerner, A. [1986], 'Link allocation in mobile radio networks with noisy channel', *Proceedings of the IEEE INFOCOM* pp. 1243–1257.

Clausen, T., Dearlove, C. and Dean, J. [2008], 'Manet neighborhood discovery protocol (nhdp)', *Internet Engineering Task Force (IETF) draft* .

Clausen, T., Dearlove, C. and Jacquet, P. [2008], The optimized link state routing protocol version 2, Technical report, Internet Engineering Task Force (IETF), http://tools.ietf.org/html/draft-ietf-manet-olsrv2-07.

Clausen, T. H. and Baccelli, E. [2005], 'Simple manet address autoconfiguration', *Internet Engineering Task Force (IETF) draft* .

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. [2001], *Introduction to Algorithms, 2nd Edition*, The MIT Press.

Corson, M. S. and Park, V. D. [1997], 'An internet manet encapsulation protocol (imep) specification', *Internet Engineering Task Force (IETF) draft* .

Dube, R. [1997], 'Signal stability based adaptive routing for ad hoc mobile networks', *Proceedings of IEEE Personal Communications* pp. 36–45.

Dunn, R. [2009], 'wxpython gui toolkit', http://www.wxpython.org/index.php.

EFunda [2009], 'Under sampling', http://www.efunda.com.

Erdfelt, J. [2008], 'Usb library', http://libusb.wiki.sourceforge.net/.

Fazio, M., Palazzi, C. E., Das, S. and Gerla, M. [2006], Automatic ip address configuration in vanets, *in* 'VANET '06: Proceedings of the 3rd international workshop on Vehicular ad hoc networks', ACM, New York, NY, USA, pp. 100–101.

Fazio, M., Villari, M. and Puliafito, A. [2006], 'Ip address autoconfiguration in ad hoc networks: design, implementation and measurements', *Computer Networks: The International Journal of Computer and Telecommunications Networking* **50**, 898–920.

Forouzan, B. A. [2007], *Data Communications and Networking*, McGraw Hill.

Haas, Z. and Pearlman, M. [1998], 'The performance of query control schemed for the zone routing protocol', *Proceedings of ACM SIGCOMM '98* pp. 360–368.

Haas, Z., Pearlman, M. and Samar, P. [2002*a*], 'The interzone routing protocol (ierp) for ad hoc networks', *Internet Engineering Task Force (IETF) draft* .

Haas, Z., Pearlman, M. and Samar, P. [2002*b*], 'The intrazone routing protocol (iarp) for ad hoc networks', *Internet Engineering Task Force (IETF) draft* .

Haas, Z., Pearlman, M. and Samar, P. [2002*c*], 'The zone routing protocol (zrp) for ad hoc networks', *Internet Engineering Task Force (IETF) draft* .

Hofmann, P. [2006], 'Multihop radio access network (mran) protocol specification', *Internet Engineering Task Force (IETF) draft* .

IEEE [1979], *Programs for Digital Signal Processing*, IEEE Press.

IEEE [2007], Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications, Technical report, IEEE.

Iwata, A., Chiang, C. C., Pei, G., Gerla, M. and Chen, T. W. [1999], 'Scalable routing strategies for ad hoc networks', *IEEE Journal on Selected Areas of Communications* pp. 1369–1379.

Jelger, C., Noel, T. and Frey, A. [2004], 'Gateway and address autoconfiguration for ipv6 ad hoc networks', *Internet Engineering Task Force (IETF) draft* .

Jeong, H., Oh, S., Kim, D., Park, J., Kim, H. and Toh, C. [2007], 'Passive duplicate address detection for on-demand routing protocols', *Internet Engineering Task Force (IETF) draft* .

Jeong, J., Park, J., Kim, E., Jeong, H. and Kim, D. [2006], 'Ad hoc ip address autoconfiguration', *Internet Engineering Task Force (IETF) draft* .

Johansson, T. and Carr-Motyckova, L. [2005], 'Reducing interference in ad hoc networks through topology control', *Proceedings of the ACM/SIGMOBILE Workshop on Foundations of Mobile Computing* .

Johnson, D. and Maltz, D. [1996], *Dynamic Source Routing in Ad Hoc Wireless Networks*, Kluwer Academic Publishers, pp. 153–181.

Jubin, J. and Truong, T. [1987], Distributed algorithm for efficient and interference-free broadcasting in radio networks, *in* 'Proceedings of IEEE INFOCOM'87', Vol. 3, pp. 21–32.

Kamerman, A. and Erkocevic, N. [1997], 'Microwave oven interference on wireless lans operating in the 2.4 ghz ism band', *Personal, Indoor and Mobile Radio Communications* **3**, 1221–1227.

Kim, H., Kim, S. C., Yu, M., Song, J. and Mah, P. [2007], 'Dap: Dynamic address assignment protocol in dap: Dynamic address assignment protocol in mobile ad-hoc networks', *IEEE International Symposium on consumer Electronics (ISCE)* .

Koltsidas, G., Dimitriadis, G. and Pavlidou, F. [2004], 'A performance study of the hsls routing algorithm for ad hoc networks', *Vehicular Technology Conference (VTC 2004)* .

Krunz, M., Muqattash, A. and Lee, S. [2004], 'Transmission power control in wireless ad hoc networks: Challenges, solutions, and open issues', *IEEE Network* pp. 8–14.

Lee, D., Yoo, J., Kang, H., Kim, K. and Kang, K. [2006], 'Distributed ipv6 addressing technique for mobile ad-hoc networks', *ACM SAC* .

Lee, S. J. and Gerla, M. [2000], Aodv-br: Backup routing in ad hoc networks, *in* 'Proceedings of the IEEE, Wireless Communications and Networking Conference 2000 (WCNC 2000)', Vol. 3, pp. 1311–1316.

Lee, S. J. and Gerla, M. [2001], 'Split multipath routing with maxmimally disjoint paths in ad hoc networks', *Proceedings of the IEEE International Conference on Communications 2001 (ICC 2001)* **10**.

Mase, K. and Adjih, C. [2006], 'No overhead autoconfiguration olsr', *Internet Engineering Task Force (IETF) draft* .

MathWorks [2009], 'Signal processing toolbox: Interpolation', http://www.mathworks.com.

Mohsin, M. and Prakash, R. [2002], 'Ip address assignment in a mobile ad hoc network', *IEEE MILCOM* .

Murthy, S. and Garcia-Luna-Aceves, J. J. [1996], 'An efficient routing protocol for wireless networks', *ACM Mobile Networks and Applications Journals* pp. 183–197.

Nasipuri, A. and Das, S. R. [1999], On-demand multipath routing for mobile ad hoc networks, *in* 'Proceedings of Eighth International Conference on Computer Communications and Networks', pp. 64–70.

Nesargi, S. and Prakash, R. [2002], 'Manetconf: Configuration of hosts in a mobile ad hoc network', *Proceedings of IEEE INFOCOM* .

Ogier, R., Templin, F. and Lewis, M. [2004], Topology dissemination based on reverse-path forwarding (tbrpf), Technical report, Internet Engineering Task Force (IETF).

Park, V. D. and Corson, M. S. [1997], 'A highly adaptive distributed routing algorithm for mobile and wireless networks', *Proceedings of the IEEE INFOCOM'97* pp. 103–112.

Park, V. D. and Corson, M. S. [2001], 'Temporally-ordered routing algorithm (tora) version 1 functional specification', *Internet Engineering Task Force (IETF) draft* .

Pei, G., Gerla, M. and Hong, X. [2000], 'Lanmar: Landmark routing for large scale wireless ad hoc networks with group mobility', *ACM MobiHoc* .

Pei, G., Gerla, M. and wei Chen, T. [2000], Fisheye state routing: A routing scheme for ad hoc wireless networks, *in* 'IEEE INFOCOM', pp. 70–74.

Perkins, C. and Belding-Royer, E. [1999], 'Ad hoc on-demand distance vector (aodv) routing', *IEEE Workshop on Mobile Computing Systems and Applications* .

Perkins, C. E., Belding-Royer, E. M. and Das, S. R. [2002], 'Ad hoc on-demand distance vector (aodv) routing', *Internet Engineering Task Force (IETF) draft* .

Perkins, C. E. and Bhagwat, P. [1994], 'Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers', *Computer Communications Review* pp. 234–244.

Pietikainen, K., Silvennoinen, A., Hall, M. and Haggman, S.-G. [2005], 'Ieee 802.11g tolerance to narrowband jamming', *Military Communications Journal* .

Ros, F. and Ruiz, P. [2006], 'Extensible manet auto-configuration protocol (emap)', *Internet Engineering Task Force (IETF) draft* .

Royer, E. and Toh, C. [1999], A review of current routing protocols for ad hoc mobile wireless networks, *in* 'IEEE Personal Communications', Vol. 7, IEEE, pp. 46–55.

Ruffino, S. and Stupar, P. [2006], 'Automatic configuration of ipv6 addresses for nodes in a manet with multiple gateways', *Internet Engineering Task Force (IETF) draft* .

Santivanez, C. A., Mcdonald, B., Stavrakakis, I. and Ramanathan, R. [2002], On the scalability of ad hoc routing protocols, *in* 'IEEE Conference on Computer Communications (INFOCOM)', IEEE.

Santivanez, C. A., Ramanathan, R. and Stavrakakis, I. [2001], 'Making link-state routing scale for ad hoc networks', *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing* .

Santivanez, C. and Ramanathan, R. [2003], Hazy sighted link state (hsls) routing: A scalable link state algorithm, Technical report, BBN Technologies, Cambridge, MA.

Sun, Y. and Belding-Royer, E. M. [2003], Dynamic address configuration in mobile ad hoc networks, Technical Report 2003-11, University of California - Santa Barbara, Santa Barbara, CA.

Taher, T. M., Al-Banna, A. Z., Ucci, D. R. and LoCicero, J. L. [2006], 'Characterization of an unintentional wi-fi interference device: the residential microwave oven', *Military Communications Journal* pp. 1–7.

Taher, T. M., Misurac, M. J., LoCicero, J. L. and Ucci, D. R. [2008], 'Microwave oven signal interference mitigation for wi-fi communication systems', *Consumer Communications and Networking Conference (CCNC)* **10**, 67–68.

Tan, H. and Seah, W. [2005], 'Dynamic topology control to reduce interference in manets', *Proceedings of Second International Conference on Mobile Computing and Ubiquitous Networking* .

Tang, J., Xue, G., Chandler, C. and Zhang, W. [2005], 'Interference-aware routing in multihop wirelesss networks using directional antennas', *Proceedings of IEEE INFOCOM* .

Tayal, A. P. and Patnaik, L. M. [2004], 'An address assignment for the automatic configuration of mobile ad hoc networks', *Personal and Ubiquitous Computing* **8**, 47–54.

Templin, F., Russert, S. and Chakeres, I. [2006], 'Manet autoconfiguration using dhcp', *Internet Engineering Task Force (IETF) draft* .

Toh, C. K. [1997], 'Associativity-based routing for ad-hoc networks', *Wireless Personal Communications Journal, Special Issue on Mobile Networking and Computing Systems* **4**, 103–139.

Tonnesen, A. [2009], 'olsrd: an ad-hoc wireless mesh routing daemon', http://www.olsr.org/.

Tseng, Y.-C., Ni, S.-Y., Chen, Y.-S. and Shieu, J.-P. [2002], The broadcast storm problem in a mobile ad hoc network, *in* 'Wireless Networks', Vol. 8, Kluwer Academic Publishers, pp. 153–167.

Vaidya, N. H. [2002], Weak duplicate address detection in mobile ad hoc networks, *in* 'The ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)', ACM.

Valera, A., Seah, W. K. G. and Rao, S. [2003], 'Cooperative packet caching and shortest multipath routing in mobile ad hoc networks', *IEEE INFOCOM* pp. 260–269.

Wang, C.-Y., Li, C.-Y., Hwang, R.-H. and Chen, Y.-S. [2005], 'Global connectivity for mobile ipv6-based ad hoc networks', *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA)* .

Weniger, K. [2003], 'Passive duplicate address detection in mobile ad hoc networks', *Proceedings of IEEE WCNC* .

Weniger, K. [2005], 'Pacman: Passive autoconfiguration for mobile ad hoc networks', *IEEE JSAC, Special Issue on Wireless Ad Hoc Networks* .

Weniger, K. and Mase, K. [2006], 'Pdad-olsr: Passive duplicate address detection for olsr', *Internet Engineering Task Force (IETF) draft* .

Weniger, K. and Zitterbart, M. [2002], Ipv6 autoconfiguration in large scale mobile ad-hoc networks, *in* 'In Proceedings of European Wireless 2002', pp. 142–148.

Weniger, K. and Zitterbart, M. [2004], 'Address autoconfiguration in mobile ad hoc networks: Current approaches and future directions', *IEEE Network* pp. 6–11.

Yao, Z., Jiang, J., Fan, P., Cao, Z. and Li, V. O. [2003], 'A neighbor-table-based multipath routing in ad hoc networks', *The 57th IEEE Semiannual Vehicular Technology Conference (VTC 2003)* .

Yao, Z., Ma, Z. and Cao, Z. [2003], 'A multipath routing scheme combating with frequent topology changes in wireless ad hoc networks', *Proceedings of the International Conference in Communication Technology 2003 (ICCT 2003)* pp. 1250–1253.

Zhang, X., Liu, Q., Shi, D., Liu, Y. and Yu, X. [2007], An average link interference-aware routing protocol for mobile ad hoc networks, *in* 'Proceedings of the Third International Conference on Wireless and Mobile Communications (ICWMC'07)', IEEE, pp. 10–16.

Zhou, H., Ni, L. M. and Mutka, M. W. [2003], 'Prophet address allocation for large scale manets', *Proceedings of IEEE INFOCOM* **2**, 1304–1311.

Appendices

Appendix A

USRP Command and Control GUI Screenshots

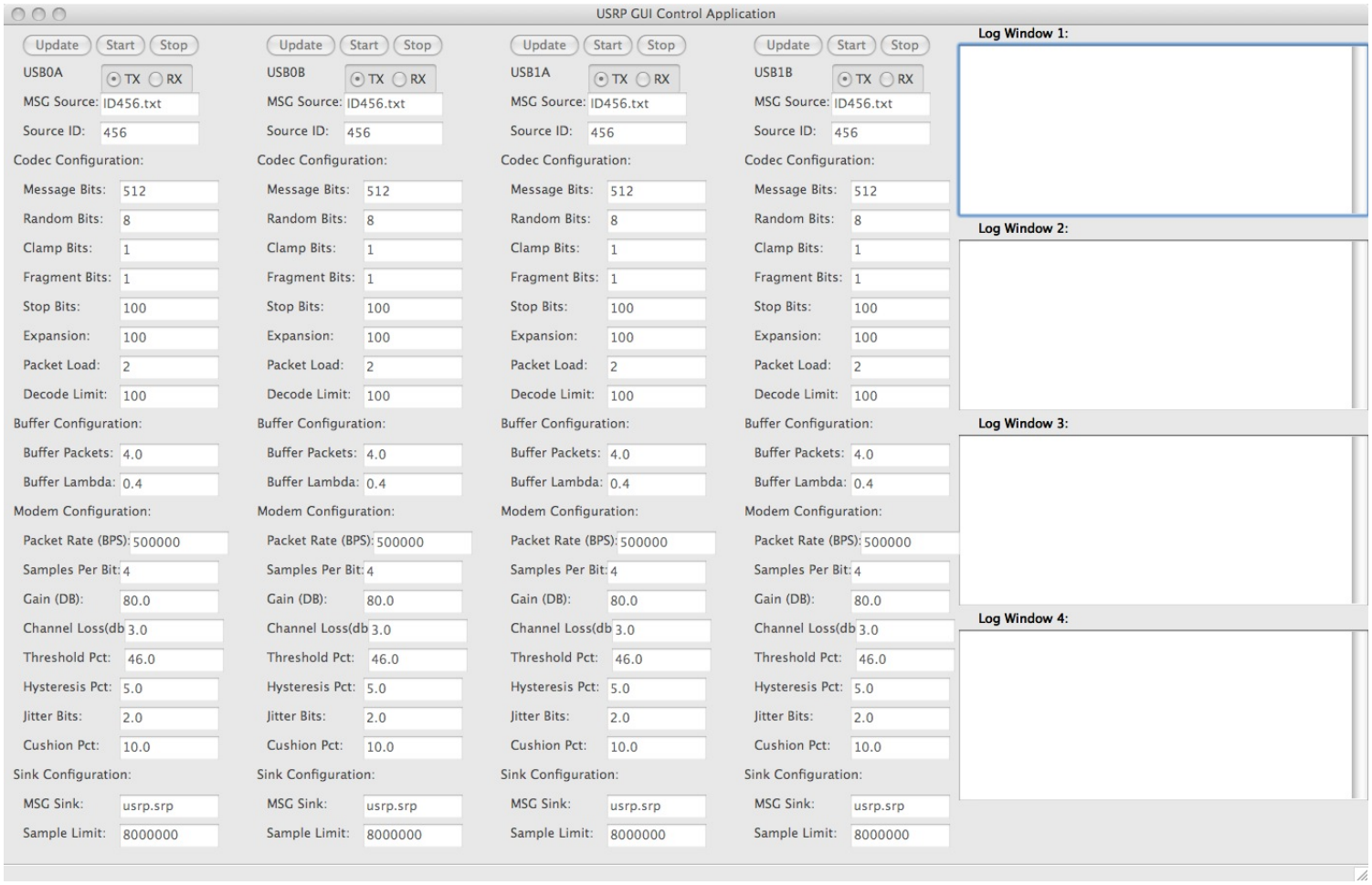

Figure A.1: USRP Command and Control GUI Screenshot 1

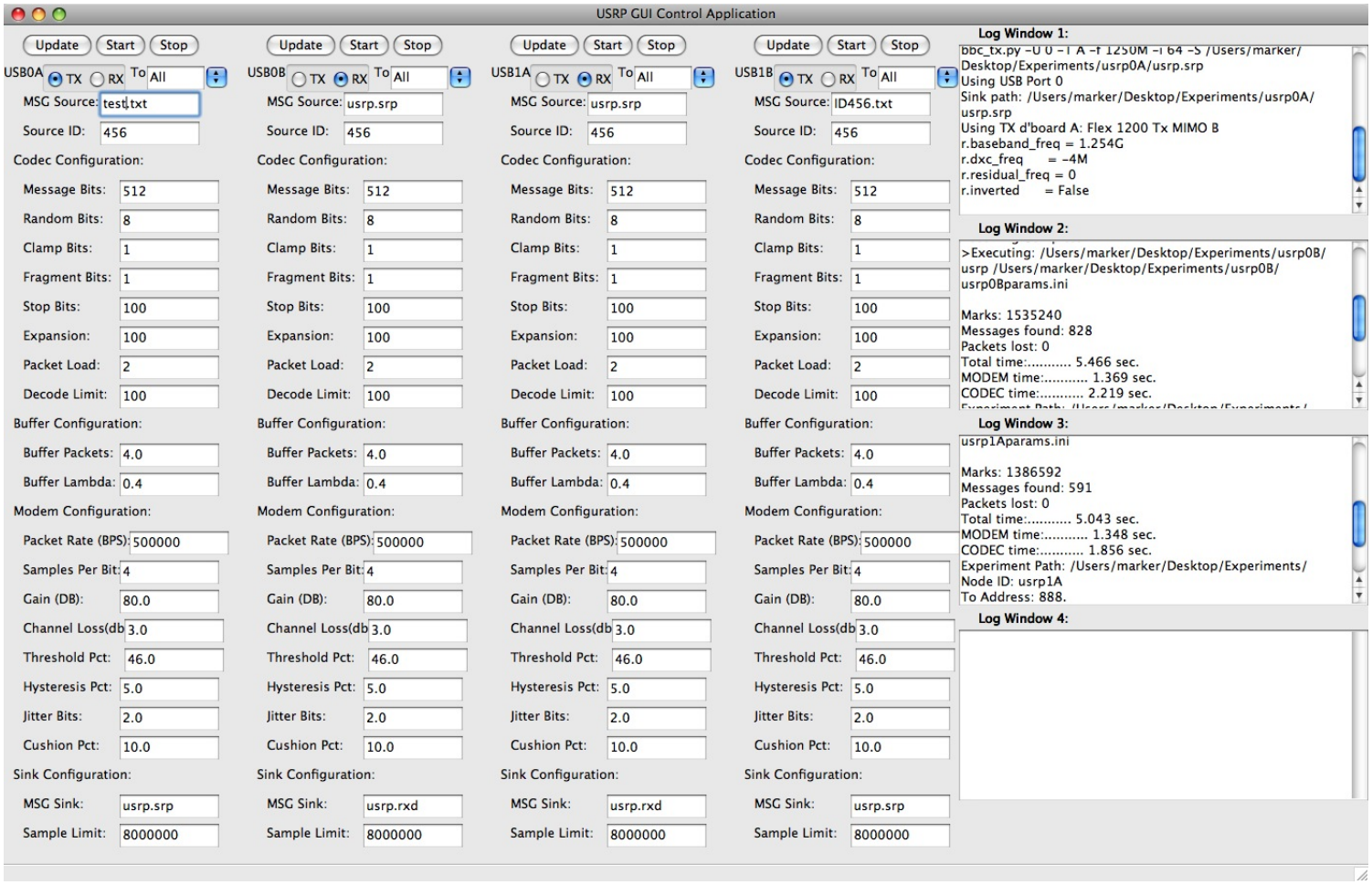

Figure A.2: USRP Command and Control GUI Screenshot 2

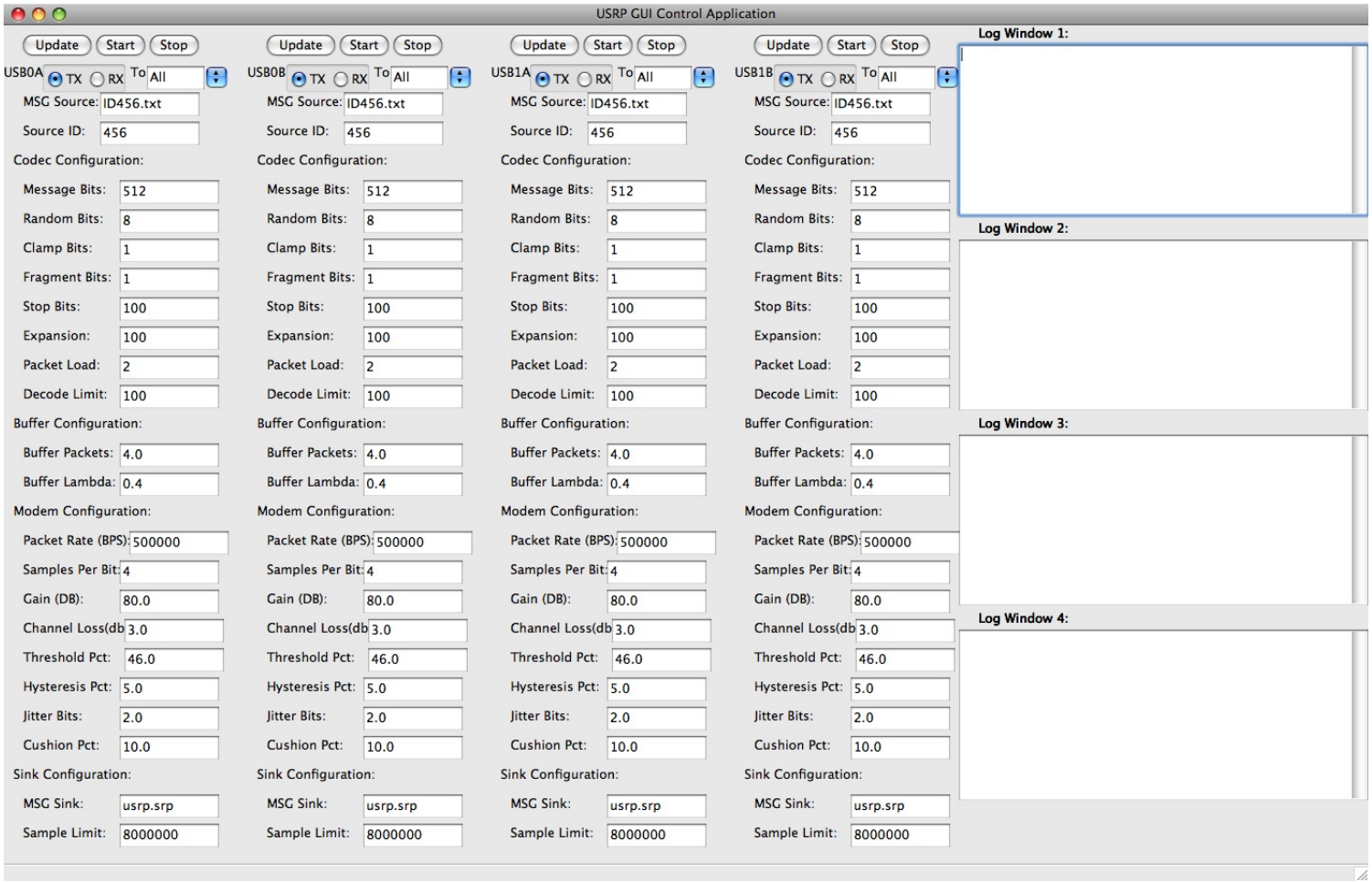

Figure A.3: USRP Command and Control GUI Screenshot 3

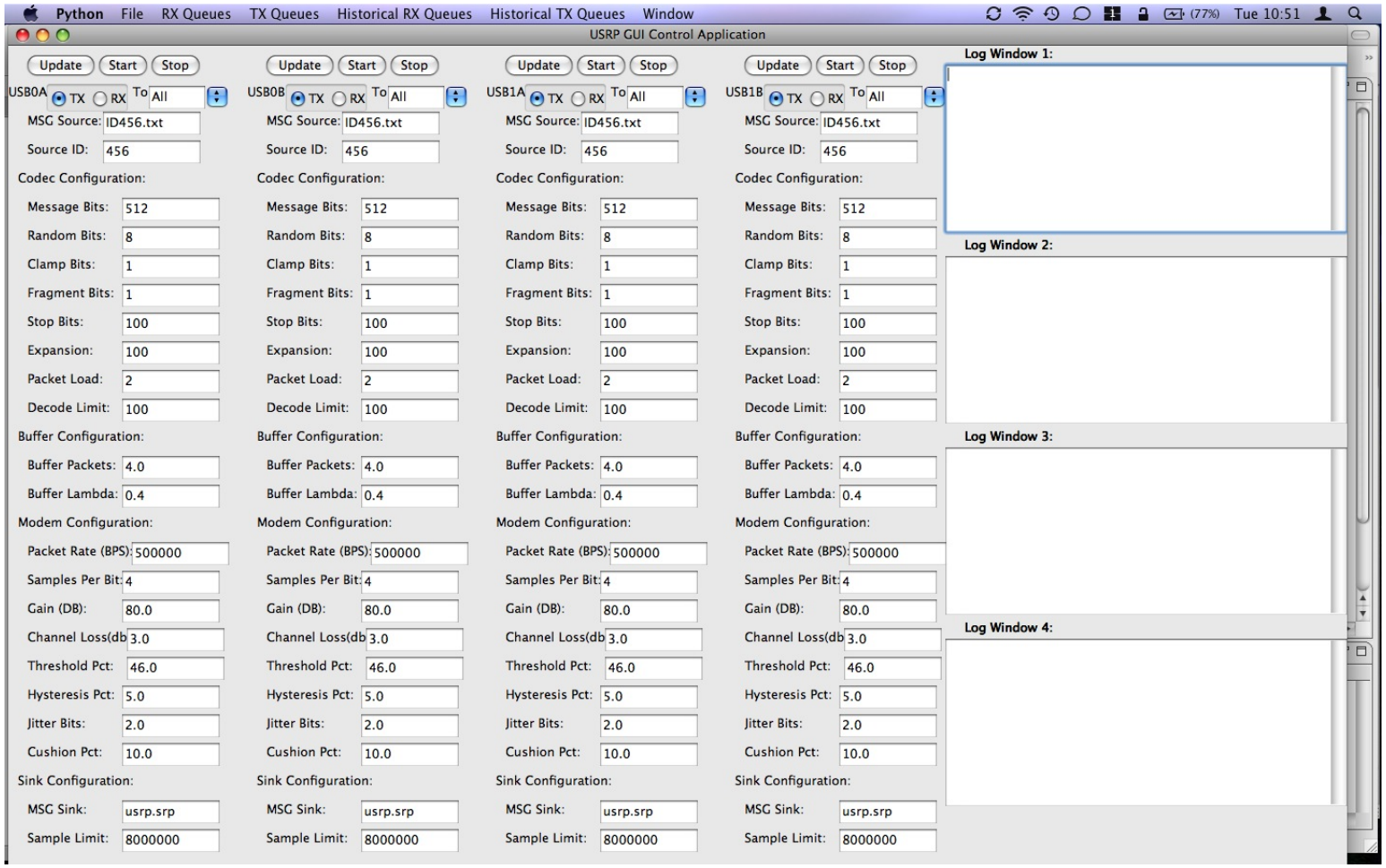

Figure A.4: USRP Command and Control GUI Screenshot 4

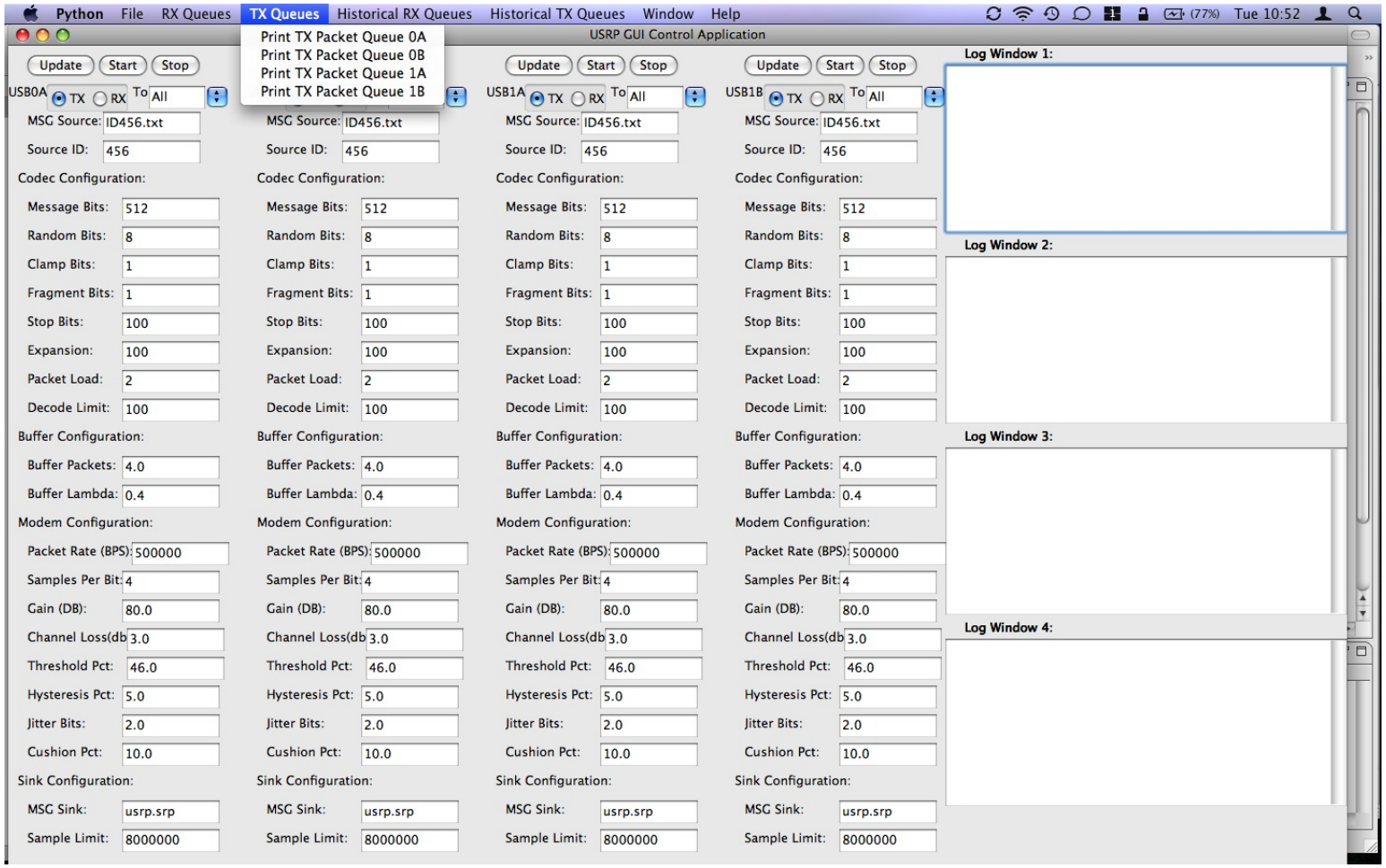

Figure A.5: USRP Command and Control GUI Screenshot 5

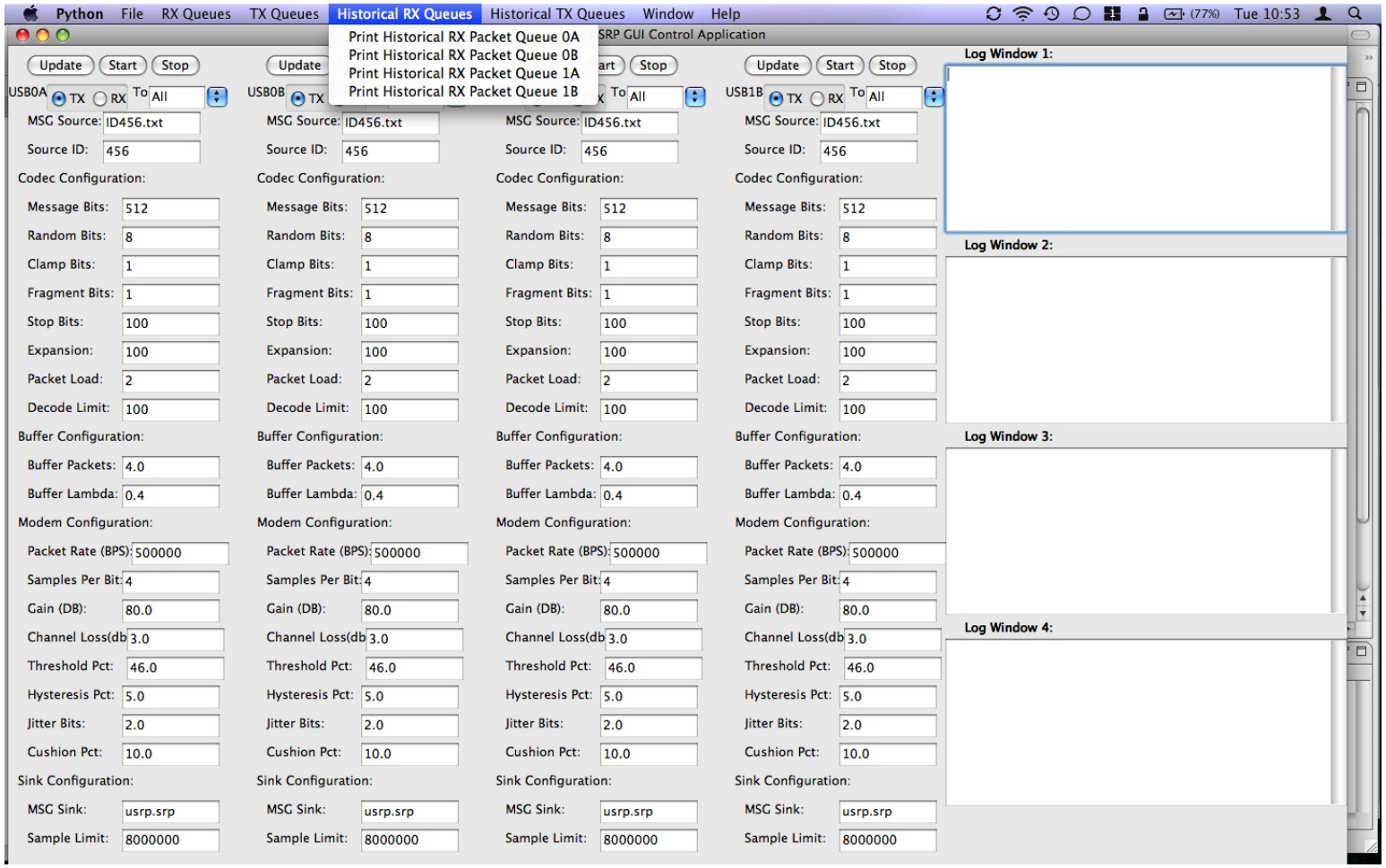

Figure A.6: USRP Command and Control GUI Screenshot 6

Appendix B

Phase III Diagrams

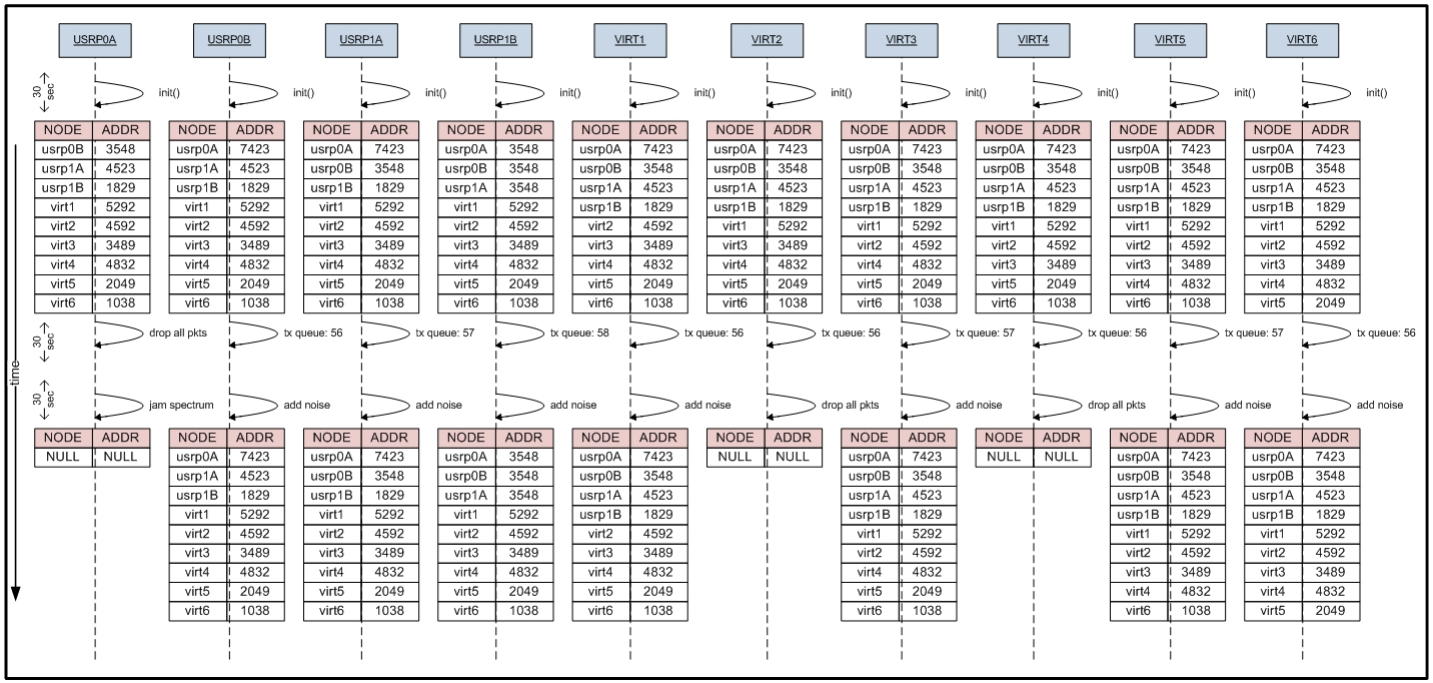

Figure B.1: Pure OLSR With Noise Diagram

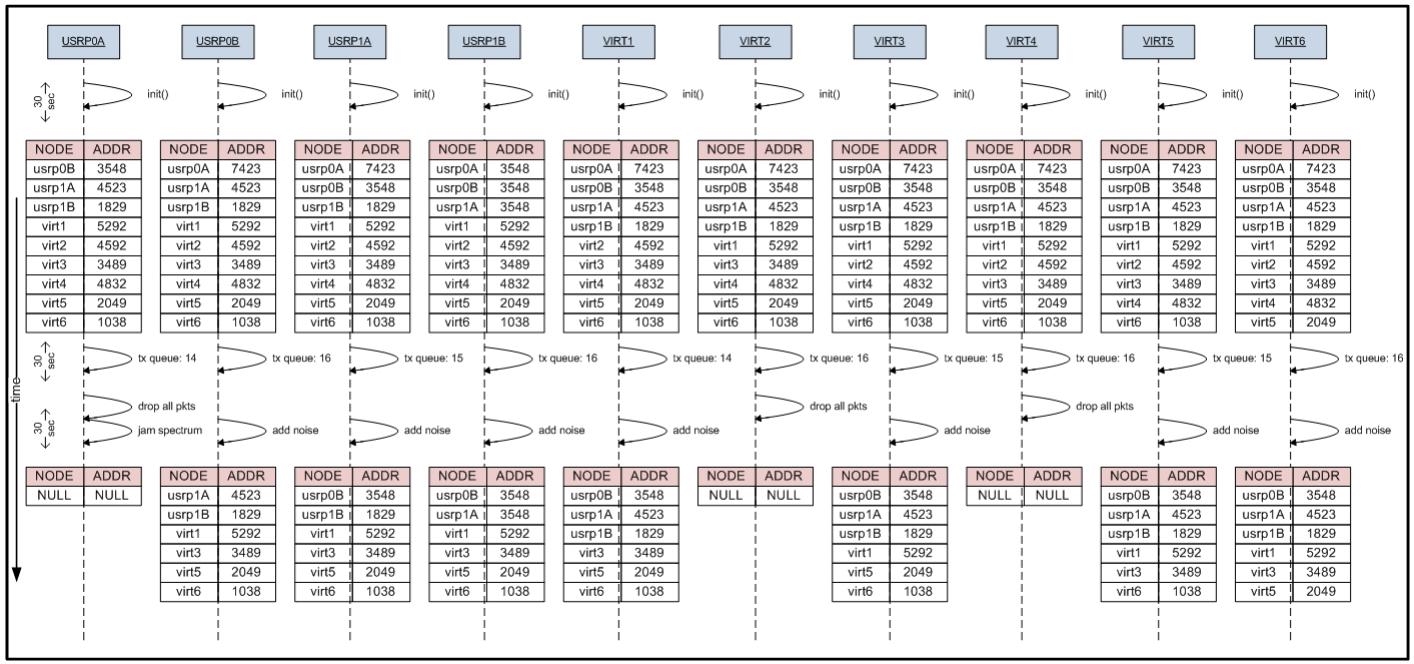

Figure B.2: BBC-OLSR With Noise Diagram

Appendix C

Source Code Listing

C.1 USRP C2 GUI and Data-Link Layer

```
1  #!/usr/bin/env python
2  # -*- coding: <<encoding>> -*-
3  #-----------------------------------------------------------------------------
4  #  Mark Kuhr
5  #  USRP GUI Control Application
6  #-----------------------------------------------------------------------------
7
8  import wxversion
9  wxversion.select("2.8")
10 import wx, wx.html
11 import sys
12 import os
13 import time
14 import struct
15 from fcntl import ioctl
16 import array
17 import threading
18 import subprocess
19 import re
20 import packet
21 import random
22
23 #unique identifiers for varous elements
24 ID_UPDATE_USB0A=0
25 ID_UPDATE_USB0B=1
26 ID_UPDATE_USB1A=2
27 ID_UPDATE_USB1B=3
28 ID_UPDATE_VIRT1=260
29 ID_UPDATE_VIRT2=261
30 ID_UPDATE_VIRT3=262
31 ID_UPDATE_VIRT4=263
32 ID_UPDATE_VIRT5=264
33 ID_UPDATE_VIRT6=265
34
35 ID_RADIO_USB0A=4
36 ID_RADIO_USB0B=5
```

```
37  ID_RADIO_USB1A=6
38  ID_RADIO_USB1B=7
39  ID_LOGWIN_USB0A=8
40  ID_LOGWIN_USB0B=9
41  ID_LOGWIN_USB1A=10
42  ID_LOGWIN_USB1B=11
43
44  ID_TXRX_USB0A=12
45  ID_TXRX_USB0B=13
46  ID_TXRX_USB1A=14
47  ID_TXRX_USB1B=15
48  ID_TXRX_VIRT1=266
49  ID_TXRX_VIRT2=267
50  ID_TXRX_VIRT3=268
51  ID_TXRX_VIRT4=269
52  ID_TXRX_VIRT5=270
53  ID_TXRX_VIRT6=271
54
55  ID_SRC_NAME_USB0A=16
56  ID_SRC_NAME_USB0B=17
57  ID_SRC_NAME_USB1A=18
58  ID_SRC_NAME_USB1B=19
59  SRC_ID_USB0A=20
60  SRC_ID_USB0B=21
61  SRC_ID_USB1A=22
62  SRC_ID_USB1B=23
63  ID_MSG_BITS_USB0A=24
64  ID_MSG_BITS_USB0B=25
65  ID_MSG_BITS_USB1A=26
66  ID_MSG_BITS_USB1B=27
67  ID_RAND_BITS_USB0A=28
68  ID_RAND_BITS_USB0B=29
69  ID_RAND_BITS_USB1A=30
70  ID_RAND_BITS_USB1B=31
71  ID_CLAMP_BITS_USB0A=32
72  ID_CLAMP_BITS_USB0B=33
73  ID_CLAMP_BITS_USB1A=34
74  ID_CLAMP_BITS_USB1B=35
75  ID_FRAG_BITS_USB0A=36
76  ID_FRAG_BITS_USB0B=37
77  ID_FRAG_BITS_USB1A=38
78  ID_FRAG_BITS_USB1B=39
79  ID_STOP_BITS_USB0A=40
80  ID_STOP_BITS_USB0B=41
81  ID_STOP_BITS_USB1A=42
```

```
82  ID_STOP_BITS_USB1B=43
83  ID_EXPANSION_USB0A=44
84  ID_EXPANSION_USB0B=45
85  ID_EXPANSION_USB1A=46
86  ID_EXPANSION_USB1B=47
87  ID_PKT_LOAD_USB0A=48
88  ID_PKT_LOAD_USB0B=49
89  ID_PKT_LOAD_USB1A=50
90  ID_PKT_LOAD_USB1B=51
91  ID_DECODE_LIM_USB0A=52
92  ID_DECODE_LIM_USB0B=53
93  ID_DECODE_LIM_USB1A=54
94  ID_DECODE_LIM_USB1B=55
95  ID_BUF_PKT_USB0A=56
96  ID_BUF_PKT_USB0B=57
97  ID_BUF_PKT_USB1A=58
98  ID_BUF_PKT_USB1B=59
99  ID_BUF_LAMBDA_USB0A=60
100 ID_BUF_LAMBDA_USB0B=61
101 ID_BUF_LAMBDA_USB1A=62
102 ID_BUF_LAMBDA_USB1B=63
103 ID_PKT_RATE_USB0A=64
104 ID_PKT_RATE_USB0B=65
105 ID_PKT_RATE_USB1A=66
106 ID_PKT_RATE_USB1B=67
107 ID_SAMPLES_PER_BIT_USB0A=68
108 ID_SAMPLES_PER_BIT_USB0B=69
109 ID_SAMPLES_PER_BIT_USB1A=70
110 ID_SAMPLES_PER_BIT_USB1B=71
111 ID_GAIN_USB0A=72
112 ID_GAIN_USB0B=73
113 ID_GAIN_USB1A=74
114 ID_GAIN_USB1B=75
115 ID_CHANNEL_LOSS_USB0A=76
116 ID_CHANNEL_LOSS_USB0B=77
117 ID_CHANNEL_LOSS_USB1A=78
118 ID_CHANNEL_LOSS_USB1B=79
119 ID_THRESHOLD_PCT_USB0A=80
120 ID_THRESHOLD_PCT_USB0B=81
121 ID_THRESHOLD_PCT_USB1A=82
122 ID_THRESHOLD_PCT_USB1B=83
123 ID_HYSTERESIS_USB0A=84
124 ID_HYSTERESIS_USB0B=85
125 ID_HYSTERESIS_USB1A=86
126 ID_HYSTERESIS_USB1B=87
```

```
127  ID_JITTER_USB0A=88
128  ID_JITTER_USB0B=89
129  ID_JITTER_USB1A=90
130  ID_JITTER_USB1B=91
131  ID_CUSHION_PCT_USB0A=92
132  ID_CUSHION_PCT_USB0B=93
133  ID_CUSHION_PCT_USB1A=94
134  ID_CUSHION_PCT_USB1B=95
135  ID_SINK_NAME_USB0A=96
136  ID_SINK_NAME_USB0B=97
137  ID_SINK_NAME_USB1A=98
138  ID_SINK_NAME_USB1B=99
139  ID_SINK_SAMPLE_LIM_USB0A=100
140  ID_SINK_SAMPLE_LIM_USB0B=101
141  ID_SINK_SAMPLE_LIM_USB1A=102
142  ID_SINK_SAMPLE_LIM_USB1B=103
143
144  ID_START_USB0A=104
145  ID_START_USB0B=105
146  ID_START_USB1A=106
147  ID_START_USB1B=107
148  ID_START_VIRT1=280
149  ID_START_VIRT2=281
150  ID_START_VIRT3=282
151  ID_START_VIRT4=283
152  ID_START_VIRT5=284
153  ID_START_VIRT6=285
154
155  ID_PROCESS_1=108
156  ID_PROCESS_2=109
157  ID_PROCESS_3=110
158  ID_PROCESS_4=111
159  ID_PROCESS_5=248
160  ID_PROCESS_6=249
161  ID_PROCESS_7=250
162  ID_PROCESS_8=251
163  ID_PROCESS_9=252
164  ID_PROCESS_10=253
165
166  ID_STOP_USB0A=112
167  ID_STOP_USB0B=113
168  ID_STOP_USB1A=114
169  ID_STOP_USB1B=115
170  ID_STOP_VIRT1=254
171  ID_STOP_VIRT2=255
```

```
172  ID_STOP_VIRT3=256
173  ID_STOP_VIRT4=257
174  ID_STOP_VIRT5=258
175  ID_STOP_VIRT6=259
176
177  ID_PROCESS_ENC_1=116
178  ID_PROCESS_ENC_2=117
179  ID_PROCESS_ENC_3=118
180  ID_PROCESS_ENC_4=119
181  ID_PROCESS_ENC_5=200
182  ID_PROCESS_ENC_6=201
183  ID_PROCESS_ENC_7=202
184  ID_PROCESS_ENC_8=203
185  ID_PROCESS_ENC_9=204
186  ID_PROCESS_ENC_10=205
187
188  ID_PROCESS_DEC_1=120
189  ID_PROCESS_DEC_2=121
190  ID_PROCESS_DEC_3=122
191  ID_PROCESS_DEC_4=123
192  ID_PROCESS_DEC_5=206
193  ID_PROCESS_DEC_6=207
194  ID_PROCESS_DEC_7=208
195  ID_PROCESS_DEC_8=209
196  ID_PROCESS_DEC_9=210
197  ID_PROCESS_DEC_10=211
198
199  ID_RECIPIENT_USB0A=124
200  ID_RECIPIENT_USB0B=125
201  ID_RECIPIENT_USB1A=126
202  ID_RECIPIENT_USB1B=127
203  ID_RECIPIENT_VIRT1=212
204  ID_RECIPIENT_VIRT2=213
205  ID_RECIPIENT_VIRT3=214
206  ID_RECIPIENT_VIRT4=215
207  ID_RECIPIENT_VIRT5=216
208  ID_RECIPIENT_VIRT6=217
209
210  ID_PRINT_RX_QUEUE_USB0A=128
211  ID_PRINT_RX_QUEUE_USB0B=129
212  ID_PRINT_RX_QUEUE_USB1A=130
213  ID_PRINT_RX_QUEUE_USB1B=131
214  ID_PRINT_RX_QUEUE_VIRT1=218
215  ID_PRINT_RX_QUEUE_VIRT2=219
216  ID_PRINT_RX_QUEUE_VIRT3=220
```

```
217 ID_PRINT_RX_QUEUE_VIRT4=221
218 ID_PRINT_RX_QUEUE_VIRT5=222
219 ID_PRINT_RX_QUEUE_VIRT6=223
220
221 ID_PRINT_TX_QUEUE_USB0A=132
222 ID_PRINT_TX_QUEUE_USB0B=133
223 ID_PRINT_TX_QUEUE_USB1A=134
224 ID_PRINT_TX_QUEUE_USB1B=135
225 ID_PRINT_TX_QUEUE_VIRT1=224
226 ID_PRINT_TX_QUEUE_VIRT2=225
227 ID_PRINT_TX_QUEUE_VIRT3=226
228 ID_PRINT_TX_QUEUE_VIRT4=227
229 ID_PRINT_TX_QUEUE_VIRT5=228
230 ID_PRINT_TX_QUEUE_VIRT6=229
231
232 ID_PRINT_RX_QUEUE_HISTORICAL_USB0A=136
233 ID_PRINT_RX_QUEUE_HISTORICAL_USB0B=137
234 ID_PRINT_RX_QUEUE_HISTORICAL_USB1A=138
235 ID_PRINT_RX_QUEUE_HISTORICAL_USB1B=139
236 ID_PRINT_RX_QUEUE_HISTORICAL_VIRT1=230
237 ID_PRINT_RX_QUEUE_HISTORICAL_VIRT2=231
238 ID_PRINT_RX_QUEUE_HISTORICAL_VIRT3=232
239 ID_PRINT_RX_QUEUE_HISTORICAL_VIRT4=233
240 ID_PRINT_RX_QUEUE_HISTORICAL_VIRT5=234
241 ID_PRINT_RX_QUEUE_HISTORICAL_VIRT6=235
242
243 ID_PRINT_TX_QUEUE_HISTORICAL_USB0A=140
244 ID_PRINT_TX_QUEUE_HISTORICAL_USB0B=141
245 ID_PRINT_TX_QUEUE_HISTORICAL_USB1A=142
246 ID_PRINT_TX_QUEUE_HISTORICAL_USB1B=143
247 ID_PRINT_TX_QUEUE_HISTORICAL_VIRT1=236
248 ID_PRINT_TX_QUEUE_HISTORICAL_VIRT2=237
249 ID_PRINT_TX_QUEUE_HISTORICAL_VIRT3=238
250 ID_PRINT_TX_QUEUE_HISTORICAL_VIRT4=239
251 ID_PRINT_TX_QUEUE_HISTORICAL_VIRT5=240
252 ID_PRINT_TX_QUEUE_HISTORICAL_VIRT6=241
253
254 ID_CLEAR_LOGWIN_ALL=144
255 ID_CLEAR_LOGWIN_1=145
256 ID_CLEAR_LOGWIN_2=146
257 ID_CLEAR_LOGWIN_3=147
258 ID_CLEAR_LOGWIN_4=148
259 ID_CLEAR_LOGWIN_5=242
260 ID_CLEAR_LOGWIN_6=243
261 ID_CLEAR_LOGWIN_7=244
```

```
262 ID_CLEAR_LOGWIN_8=245
263 ID_CLEAR_LOGWIN_9=246
264 ID_CLEAR_LOGWIN_10=247
265
266 ID_RESET=149
267 ID_ABOUT=150
268 ID_EXIT=151
269 ID_WAIT_PROCESS=152
270
271 ID_PROCESS_OLSR_1=153 #real interface
272 ID_PROCESS_OLSR_2=154 #real interface
273 ID_PROCESS_OLSR_3=155 #real interface
274 ID_PROCESS_OLSR_4=156 #real interface
275 ID_PROCESS_OLSR_5=157 #virtual interface
276 ID_PROCESS_OLSR_6=158 #virtual interface
277 ID_PROCESS_OLSR_7=159 #virtual interface
278 ID_PROCESS_OLSR_8=160 #virtual interface
279 ID_PROCESS_OLSR_9=161 #virtual interface
280 ID_PROCESS_OLSR_10=162#virtual interface
281
282 ID_VIEW_OLSR_1=163 #virtual interface viewer
283 ID_VIEW_OLSR_2=164 #virtual interface viewer
284 ID_VIEW_OLSR_3=165 #virtual interface viewer
285 ID_VIEW_OLSR_4=166 #virtual interface viewer
286 ID_VIEW_OLSR_5=167 #virtual interface viewer
287 ID_VIEW_OLSR_6=168 #virtual interface viewer
288
289 ID_VIEW_TABLE_USRP0A = 270
290 ID_VIEW_TABLE_USRP0B = 271
291 ID_VIEW_TABLE_USRP1A = 272
292 ID_VIEW_TABLE_USRP1B = 273
293 ID_VIEW_TABLE_VIRT1 = 274
294 ID_VIEW_TABLE_VIRT2 = 275
295 ID_VIEW_TABLE_VIRT3 = 276
296 ID_VIEW_TABLE_VIRT4 = 277
297 ID_VIEW_TABLE_VIRT5 = 278
298 ID_VIEW_TABLE_VIRT6 = 279
299
300 #OLSR SUPPORT FUNCTIONS
301 #SUPPORT FOR ETHERNET AND IP DATAGRAMS
302 #DEVELOPED WITH SUPPORT OF D. SANDERS.
303
304 experiment_path = "/Users/marker/Desktop/sdr/"
305
306 class ethernet_frame:
```

```
307     def __init__(self, raw_frame):
308       self.raw_frame = raw_frame
309       self.dest_addr = raw_frame[:6].encode("hex")
310       self.src_addr = raw_frame[6:12].encode("hex")
311       #tmp = struct.unpack("H", raw_frame[12:14])
312       self.type = raw_frame[12:14].encode("hex") #tmp[0]
313       self.datagram = None
314       if self.type == "0800":
315         self.datagram = ip_datagram(raw_frame[14:])
316       else:
317         self.datagram = raw_frame[14:]
318
319     def toString(self):
320       return self.dest_addr+" "+self.src_addr+" "+self.type
321
322     def __repr__(self):
323       if self.type== "0800":
324         s = self.toString() + "\n" + self.datagram.toString()
325       else:
326         s = self.toString() + "\n" + self.datagram
327       return s
328
329   class ip_datagram:
330     def __init__(self, raw_dg):
331       self.raw_dg = raw_dg
332       self.ver_hlen = raw_dg[:1].encode("hex")
333       self.version = self.ver_hlen[:1]
334       self.hlen = self.ver_hlen[1:]
335       self.service = raw_dg[1:2].encode("hex")
336       self.total_length = raw_dg[2:4].encode("hex")
337       self.id = raw_dg[4:6].encode("hex")
338       self.flags_fragmentation = raw_dg[6:8].encode("hex")
339       self.ttl = raw_dg[8:9].encode("hex")
340       self.protocol = raw_dg[9:10].encode("hex")
341       self.hd_checksum = raw_dg[10:12].encode("hex")
342       self.sourceip = raw_dg[12:16].encode("hex")
343       self.destip = raw_dg[16:20].encode("hex")
344       tmp = struct.unpack("B", self.hlen)
345       self.option = None
346       if tmp[0] > 20:
347         self.option = raw_dg[20:tmp[0]].encode("hex")
348       self.packet = None
349       #if self.protocol == "0011":
350       self.packet = udp_packet(raw_dg[tmp[0]:])
351       #elif self.protocol == "0006":
```

```
352       #    self.packet = tcp_packet(raw_dg[tmp[0]:])
353       #elif self.protocol == "0001":
354       #    self.packet = icmp_packet(raw_dg[tmp[0]:])
355       #elif self.protocol == "0002":
356       #    self.packet = igmp_packet(raw_dg[tmp[0]:])
357
358     def toString(self):
359       s = " ".join((self.version, self.hlen, self.service, self.total_length))
360       s+='\n'
361       s+= " ".join((self.id, self.flags_fragmentation))
362       s+='\n'
363       s+= " ".join((self.ttl, self.protocol, self.hd_checksum))
364       s+='\n'
365       s+=self.sourceip
366       s+='\n'
367       s+=self.destip
368       return s + "\n" + self.packet.toString()
369
370   class tcp_packet:
371     def __init__(self, raw_packet):
372       self.raw_packet = raw_packet
373
374     def toStringt(self):
375       return self.raw_packet.encode("hex")
376
377   class udp_packet:
378     def __init__(self, raw_packet):
379       self.raw_packet = raw_packet
380
381     def toString(self):
382       return self.raw_packet.encode("hex")
383
384   class icmp_packet:
385     def __init__(self, raw_packet):
386       self.raw_packet = raw_packet
387
388   class igmp_packet:
389     def __init__(self, raw_packet):
390       self.raw_packet = raw_packet
391
392
393
394   ################################################
395   ## TAP INTERFACE OPEN & CONFIGURE GLOBAL ##
396   ################################################
```

```
397
398 class tap_device(threading.Thread):
399     def __init__(self, device, address):
400         self.running = 1
401         self.device = device
402         self.tap_fd = os.open("/dev/"+device, os.O_RDWR)
403
404         #print self.tap_fd
405         subprocess.Popen([r"ifconfig",device,address]).wait()
406         threading.Thread.__init__ ( self )
407
408     def run(self):
409         while self.running:
410             #read from tap device
411             count = 10
412             payload_bundle=""
413             payload_stack = []
414
415             #bundle packets into groups of 10
416             while(count >= 0):
417                 payload = os.read(self.tap_fd, 1500)
418                 payload_stack.append(payload)
419                 count = count - 1
420
421             #setup local path for file prior to transmission
422             file = open(experiment_path+self.device_to_radio_id(self.device)+"/pkt_"+ self.device+
                    "_"+str(time.time())+".pcap", "w")
423
424             #write pkt to file to be source of USRP TX
425             #file.write(payload)
426
427             while(len(payload_stack)>0):
428                 current_payload = payload_stack.pop()
429                 file.write(current_payload)
430                 os.write(self.tap_fd, current_payload)
431
432             #print str(time.time())+" Received frame on ", self.device
433             #frame = ethernet_frame(payload)
434             #print frame
435             #print "\n"
436
437             #print "Sending Ethernet Frame:", frame.raw_frame.encode("hex")
438
439             #write frame to specific tap device for testing
440             #os.write(self.tap_fd, payload)#frame.raw_frame)
```

```
441
442         #not running anymore, close this file descriptor
443         self.close()
444
445     def device_to_radio_id(self,device):
446         if(device=="tap0"):
447             return "usrp0A"
448         elif(device=="tap1"):
449             return "usrp0B"
450         elif(device=="tap2"):
451             return "usrp1A"
452         elif(device=="tap3"):
453             return "usrp1B"
454         #virtual nodes
455         elif(device=="tap4"):
456             return "virt1"
457         elif(device=="tap5"):
458             return "virt2"
459         elif(device=="tap6"):
460             return "virt3"
461         elif(device=="tap7"):
462             return "virt4"
463         elif(device=="tap8"):
464             return "virt5"
465         elif(device=="tap9"):
466             return "virt6"
467         else: return "" #should never occur
468
469
470     def stop(self):
471         self.running = 0
472         #self.close()
473
474     def close(self):
475         os.close(self.tap_fd)
476
477
478 ######################
479 ## BEGIN MAIN FRAME ##
480 ######################
481
482 class Frame(wx.Frame):
483     def __init__(self, title):
484         wx.Frame.__init__(self, None, title=title, pos=(20,20), size=(1400,900))
485
```

```
486         self.experiment_path = "/Users/marker/Desktop/sdr/"
487         #self.experiment_path = "/home/mark/Desktop/sdr/"
488         #this is not sychronized with in-range nodes
489         self.recipients=['All','usrp0A','usrp0B','usrp1A','usrp1B','virt1','virt2','virt3','virt4'
                ,'virt5','virt6']
490         self.transmission_mode = ['TX','RX']
491         self.Bind(wx.EVT_CLOSE, self.OnClose)
492         self.rssi = 0
493
494         #10 tap devices for bbc-olsr daemons
495         self.tap0 = None  #for real node usrp0A
496         self.tap1 = None  #for real node usrp0B
497         self.tap2 = None  #for real node usrp1A
498         self.tap3 = None  #for real node usrp1B
499         self.tap4 = None  #virtual node 1
500         self.tap5 = None  #virtual node 2
501         self.tap6 = None  #virtual node 3
502         self.tap7 = None  #virtual node 4
503         self.tap8 = None  #virtual node 5
504         self.tap9 = None  #virtual node 6
505
506         self.txt_cmd = ""
507
508         self.nat_table = None #table for Layer3-to-Layer2 address mappings
509
510         #current abridged route tables (addr : rssi)
511         #used for testing
512         self.usrp0A_table = None
513         self.usrp0B_table = None
514         self.usrp1A_table = None
515         self.usrp1B_table = None
516         self.virt1_table = None
517         self.virt2_table = None
518         self.virt3_table = None
519         self.virt4_table = None
520         self.virt5_table = None
521         self.virt6_table = None
522
523         #has node been killed by action?
524         self.kill_0A=False
525         self.kill_0B=False
526         self.kill_1A=False
527         self.kill_1B=False
528         self.kill_virt1=False
529         self.kill_virt2=False
```

```
530             self.kill_virt3=False
531             self.kill_virt4=False
532             self.kill_virt5=False
533             self.kill_virt6=False
534
535             #must interleave
536             self.dual_rx_0 = False
537             self.dual_rx_1 = False
538
539             #virtual node modes (rx or tx)
540             self.rxtx_virt1 = "RX" #initial mode
541             self.rxtx_virt2 = "RX"
542             self.rxtx_virt3 = "RX"
543             self.rxtx_virt4 = "RX"
544             self.rxtx_virt5 = "RX"
545             self.rxtx_virt6 = "RX"
546
547             #define processes for each daughterboard
548             self.process_usb0A = None
549             self.process_usb0B = None
550             self.process_usb1A = None
551             self.process_usb1B = None
552             #define processes for each virtual node
553             self.process_virt1 = None
554             self.process_virt2 = None
555             self.process_virt3 = None
556             self.process_virt4 = None
557             self.process_virt5 = None
558             self.process_virt6 = None
559             #define process for encoding for each daughterboard
560             self.encode_process_usb0A = None
561             self.encode_process_usb0B = None
562             self.encode_process_usb1A = None
563             self.encode_process_usb1B = None
564             #define process for encoding for each virtual node
565             self.encode_process_virt1 = None
566             self.encode_process_virt2 = None
567             self.encode_process_virt3 = None
568             self.encode_process_virt4 = None
569             self.encode_process_virt5 = None
570             self.encode_process_virt6 = None
571             #encode cmd var
572             self.encode_cmd = ""
573             #define process for decoding for each daughterboard
574             self.decode_process_usb0A = None
```

```
575             self.decode_process_usb0B = None
576             self.decode_process_usb1A = None
577             self.decode_process_usb1B = None
578             #define process for decoding for each virtual node
579             self.decode_process_virt1 = None
580             self.decode_process_virt2 = None
581             self.decode_process_virt3 = None
582             self.decode_process_virt4 = None
583             self.decode_process_virt5 = None
584             self.decode_process_virt6 = None
585             #decode cmd var
586             self.decode_cmd = ""
587             #define bbc-olsr processes for each daughterboard
588             self.olsr_process_usb0A = None
589             self.olsr_process_usb0B = None
590             self.olsr_process_usb1A = None
591             self.olsr_process_usb1B = None
592             #define virtual bbc-olsr processes for each non-hardware node
593             self.olsr_process_virt1 = None
594             self.olsr_process_virt2 = None
595             self.olsr_process_virt3 = None
596             self.olsr_process_virt4 = None
597             self.olsr_process_virt5 = None
598             self.olsr_process_virt6 = None
599
600             #define variables to store output of bbc-olsr nodes not shown in Log Windows
601             self.olsr_v1_output = ""
602             self.olsr_v2_output = ""
603             self.olsr_v3_output = ""
604             self.olsr_v4_output = ""
605             self.olsr_v5_output = ""
606             self.olsr_v6_output = ""
607             #olsr cmd var
608             self.olsr_cmd = ""
609
610             #define process id's
611             self.process_id_usb0A=0
612             self.process_id_usb0B=0
613             self.process_id_usb1A=0
614             self.process_id_usb1B=0
615             self.process_id_virt1=0
616             self.process_id_virt2=0
617             self.process_id_virt3=0
618             self.process_id_virt4=0
619             self.process_id_virt5=0
```

```
620             self.process_id_virt6=0
621
622             #define encode process id's
623             self.encode_process_id_usb0A=0
624             self.encode_process_id_usb0B=0
625             self.encode_process_id_usb1A=0
626             self.encode_process_id_usb1B=0
627             self.encode_process_id_virt1=0
628             self.encode_process_id_virt2=0
629             self.encode_process_id_virt3=0
630             self.encode_process_id_virt4=0
631             self.encode_process_id_virt5=0
632             self.encode_process_id_virt6=0
633             #define decode process id's
634             self.decode_process_id_usb0A=0
635             self.decode_process_id_usb0B=0
636             self.decode_process_id_usb1A=0
637             self.decode_process_id_usb1B=0
638             self.decode_process_id_virt1=0
639             self.decode_process_id_virt2=0
640             self.decode_process_id_virt3=0
641             self.decode_process_id_virt4=0
642             self.decode_process_id_virt5=0
643             self.decode_process_id_virt6=0
644
645             #define process id for bbc-olsr for each daughterboard
646             self.olsr_process_id_usb0A=0
647             self.olsr_process_id_usb0B=0
648             self.olsr_process_id_usb1A=0
649             self.olsr_process_id_usb1B=0
650             #define process id for bbc-olsrd virtual processes
651             self.olsr_process_id_v1=0
652             self.olsr_process_id_v2=0
653             self.olsr_process_id_v3=0
654             self.olsr_process_id_v4=0
655             self.olsr_process_id_v5=0
656             self.olsr_process_id_v6=0
657
658             #define pkt queues for each radio
659             #define rx packet queue for each radio
660             self.rx_pkt_queue_usrp0A=[]
661             self.rx_pkt_queue_usrp0B=[]
662             self.rx_pkt_queue_usrp1A=[]
663             self.rx_pkt_queue_usrp1B=[]
664             self.rx_pkt_queue_virt1=[]
```

```
665             self.rx_pkt_queue_virt2=[]
666             self.rx_pkt_queue_virt3=[]
667             self.rx_pkt_queue_virt4=[]
668             self.rx_pkt_queue_virt5=[]
669             self.rx_pkt_queue_virt6=[]
670
671             #define tx packet queue for each radio
672             self.tx_pkt_queue_usrp0A=[]
673             self.tx_pkt_queue_usrp0B=[]
674             self.tx_pkt_queue_usrp1A=[]
675             self.tx_pkt_queue_usrp1B=[]
676             self.tx_pkt_queue_virt1=[]
677             self.tx_pkt_queue_virt2=[]
678             self.tx_pkt_queue_virt3=[]
679             self.tx_pkt_queue_virt4=[]
680             self.tx_pkt_queue_virt5=[]
681             self.tx_pkt_queue_virt6=[]
682             #define historical pkt queues for each radio (to be used for analysis)
683             #define historical rx queues for each radio
684             self.rx_pkt_queue_historical_usrp0A=[]
685             self.rx_pkt_queue_historical_usrp0B=[]
686             self.rx_pkt_queue_historical_usrp1A=[]
687             self.rx_pkt_queue_historical_usrp1B=[]
688             self.rx_pkt_queue_historical_virt1=[]
689             self.rx_pkt_queue_historical_virt2=[]
690             self.rx_pkt_queue_historical_virt3=[]
691             self.rx_pkt_queue_historical_virt4=[]
692             self.rx_pkt_queue_historical_virt5=[]
693             self.rx_pkt_queue_historical_virt6=[]
694             #define historical tx queues for each radio
695             self.tx_pkt_queue_historical_usrp0A=[]
696             self.tx_pkt_queue_historical_usrp0B=[]
697             self.tx_pkt_queue_historical_usrp1A=[]
698             self.tx_pkt_queue_historical_usrp1B=[]
699             self.tx_pkt_queue_historical_virt1=[]
700             self.tx_pkt_queue_historical_virt2=[]
701             self.tx_pkt_queue_historical_virt3=[]
702             self.tx_pkt_queue_historical_virt4=[]
703             self.tx_pkt_queue_historical_virt5=[]
704             self.tx_pkt_queue_historical_virt6=[]
705
706
707             menuBar = wx.MenuBar()
708             menu = wx.Menu()
709             m_clear_win1 = menu.Append(ID_CLEAR_LOGWIN_1, "Clear Log 1", "Clear Log Window 1")
```

```
710         m_clear_win2 = menu.Append(ID_CLEAR_LOGWIN_2, "Clear Log 2", "Clear Log Window 2")
711         m_clear_win3 = menu.Append(ID_CLEAR_LOGWIN_3, "Clear Log 3", "Clear Log Window 3")
712         m_clear_win4 = menu.Append(ID_CLEAR_LOGWIN_4, "Clear Log 4", "Clear Log Window 4")
713         m_clear_win5 = menu.Append(ID_CLEAR_LOGWIN_5, "Clear Log 5", "Clear Log Window 5")
714         m_clear_win6 = menu.Append(ID_CLEAR_LOGWIN_6, "Clear Log 6", "Clear Log Window 6")
715         m_clear_win7 = menu.Append(ID_CLEAR_LOGWIN_7, "Clear Log 7", "Clear Log Window 7")
716         m_clear_win8 = menu.Append(ID_CLEAR_LOGWIN_8, "Clear Log 8", "Clear Log Window 8")
717         m_clear_win9 = menu.Append(ID_CLEAR_LOGWIN_9, "Clear Log 9", "Clear Log Window 9")
718         m_clear_win10 = menu.Append(ID_CLEAR_LOGWIN_10, "Clear Log 10", "Clear Log Window 10")
719         m_clear_winAll = menu.Append(ID_CLEAR_LOGWIN_ALL, "Clear All Logs", "Clear All Log Windows
                ")
720
721         m_reset = menu.Append(ID_RESET, "Reset Queues", "Reset All TX/RX Queues.")
722         m_about = menu.Append(ID_ABOUT, "About", "About this application.")
723         m_exit = menu.Append(ID_EXIT, "E&xit\tAlt-X", "Close window and exit program.")
724
725         menu2 = wx.Menu()
726         m_rx0A = menu2.Append(ID_PRINT_RX_QUEUE_USB0A,"Print RX Packet Queue 0A", "Print USRP0A RX
                Queue")
727         m_rx0B = menu2.Append(ID_PRINT_RX_QUEUE_USB0B,"Print RX Packet Queue 0B", "Print USRP0B RX
                Queue")
728         m_rx1A = menu2.Append(ID_PRINT_RX_QUEUE_USB1A,"Print RX Packet Queue 1A", "Print USRP1A RX
                Queue")
729         m_rx1B = menu2.Append(ID_PRINT_RX_QUEUE_USB1B,"Print RX Packet Queue 1B", "Print USRP1B RX
                Queue")
730         m_rxv1 = menu2.Append(ID_PRINT_RX_QUEUE_VIRT1,"Print RX Packet Queue V1", "Print VIRT1 RX
                Queue")
731         m_rxv2 = menu2.Append(ID_PRINT_RX_QUEUE_VIRT2,"Print RX Packet Queue V2", "Print VIRT2 RX
                Queue")
732         m_rxv3 = menu2.Append(ID_PRINT_RX_QUEUE_VIRT3,"Print RX Packet Queue V3", "Print VIRT3 RX
                Queue")
733         m_rxv4 = menu2.Append(ID_PRINT_RX_QUEUE_VIRT4,"Print RX Packet Queue V4", "Print VIRT4 RX
                Queue")
734         m_rxv5 = menu2.Append(ID_PRINT_RX_QUEUE_VIRT5,"Print RX Packet Queue V5", "Print VIRT5 RX
                Queue")
735         m_rxv6 = menu2.Append(ID_PRINT_RX_QUEUE_VIRT6,"Print RX Packet Queue V6", "Print VIRT6 RX
                Queue")
736
737         menu3 = wx.Menu()
738         m_tx0A = menu3.Append(ID_PRINT_TX_QUEUE_USB0A,"Print TX Packet Queue 0A", "Print USRP0A TX
                Queue")
739         m_tx0B = menu3.Append(ID_PRINT_TX_QUEUE_USB0B,"Print TX Packet Queue 0B", "Print USRP0B TX
                Queue")
740         m_tx1A = menu3.Append(ID_PRINT_TX_QUEUE_USB1A,"Print TX Packet Queue 1A", "Print USRP1A TX
                Queue")
```

```
741         m_tx1B = menu3.Append(ID_PRINT_TX_QUEUE_USB1B,"Print TX Packet Queue 1B", "Print USRP1B TX
                Queue")
742         m_txv1 = menu3.Append(ID_PRINT_TX_QUEUE_VIRT1,"Print TX Packet Queue V1", "Print VIRT1 TX
                Queue")
743         m_txv2 = menu3.Append(ID_PRINT_TX_QUEUE_VIRT2,"Print TX Packet Queue V2", "Print VIRT2 TX
                Queue")
744         m_txv3 = menu3.Append(ID_PRINT_TX_QUEUE_VIRT3,"Print TX Packet Queue V3", "Print VIRT3 TX
                Queue")
745         m_txv4 = menu3.Append(ID_PRINT_TX_QUEUE_VIRT4,"Print TX Packet Queue V4", "Print VIRT4 TX
                Queue")
746         m_txv5 = menu3.Append(ID_PRINT_TX_QUEUE_VIRT5,"Print TX Packet Queue V5", "Print VIRT5 TX
                Queue")
747         m_txv6 = menu3.Append(ID_PRINT_TX_QUEUE_VIRT6,"Print TX Packet Queue V6", "Print VIRT6 TX
                Queue")
748
749         menu4 = wx.Menu()
750         m_rx_hist_0A = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_USB0A,"Print Historical RX Packet
                Queue 0A", "Print USRP0A Historical RX Queue")
751         m_rx_hist_0B = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_USB0B,"Print Historical RX Packet
                Queue 0B", "Print USRP0B Historical RX Queue")
752         m_rx_hist_1A = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_USB1A,"Print Historical RX Packet
                Queue 1A", "Print USRP1A Historical RX Queue")
753         m_rx_hist_1B = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_USB1B,"Print Historical RX Packet
                Queue 1B", "Print USRP1B Historical RX Queue")
754         m_rx_hist_v1 = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_VIRT1,"Print Historical RX Packet
                Queue V1", "Print VIRT1 Historical RX Queue")
755         m_rx_hist_v2 = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_VIRT2,"Print Historical RX Packet
                Queue V2", "Print VIRT2 Historical RX Queue")
756         m_rx_hist_v3 = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_VIRT3,"Print Historical RX Packet
                Queue V3", "Print VIRT3 Historical RX Queue")
757         m_rx_hist_v4 = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_VIRT4,"Print Historical RX Packet
                Queue V4", "Print VIRT4 Historical RX Queue")
758         m_rx_hist_v5 = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_VIRT5,"Print Historical RX Packet
                Queue V5", "Print VIRT5 Historical RX Queue")
759         m_rx_hist_v6 = menu4.Append(ID_PRINT_RX_QUEUE_HISTORICAL_VIRT6,"Print Historical RX Packet
                Queue V6", "Print VIRT6 Historical RX Queue")
760
761         menu5 = wx.Menu()
762         m_tx_hist_0A = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_USB0A,"Print Historical TX Packet
                Queue 0A", "Print USRP0A Historical TX Queue")
763         m_tx_hist_0B = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_USB0B,"Print Historical TX Packet
                Queue 0B", "Print USRP0B Historical TX Queue")
764         m_tx_hist_1A = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_USB1A,"Print Historical TX Packet
                Queue 1A", "Print USRP1A Historical TX Queue")
```

```
765         m_tx_hist_1B = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_USB1B,"Print Historical TX Packet
                Queue 1B", "Print USRP1B Historical TX Queue")
766         m_tx_hist_v1 = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_VIRT1,"Print Historical TX Packet
                Queue V1", "Print VIRT1 Historical TX Queue")
767         m_tx_hist_v2 = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_VIRT2,"Print Historical TX Packet
                Queue V2", "Print VIRT2 Historical TX Queue")
768         m_tx_hist_v3 = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_VIRT3,"Print Historical TX Packet
                Queue V3", "Print VIRT3 Historical TX Queue")
769         m_tx_hist_v4 = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_VIRT4,"Print Historical TX Packet
                Queue V4", "Print VIRT4 Historical TX Queue")
770         m_tx_hist_v5 = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_VIRT5,"Print Historical TX Packet
                Queue V5", "Print VIRT5 Historical TX Queue")
771         m_tx_hist_v6 = menu5.Append(ID_PRINT_TX_QUEUE_HISTORICAL_VIRT6,"Print Historical TX Packet
                Queue V6", "Print VIRT6 Historical TX Queue")
772
773         menu6 = wx.Menu()
774         m_olsr_v1 = menu6.Append(ID_VIEW_OLSR_1, "View BBC-OLSR VIRT1", "View BBC-OLSR VIRT1")
775         m_olsr_v2 = menu6.Append(ID_VIEW_OLSR_2, "View BBC-OLSR VIRT2", "View BBC-OLSR VIRT2")
776         m_olsr_v3 = menu6.Append(ID_VIEW_OLSR_3, "View BBC-OLSR VIRT3", "View BBC-OLSR VIRT3")
777         m_olsr_v4 = menu6.Append(ID_VIEW_OLSR_4, "View BBC-OLSR VIRT4", "View BBC-OLSR VIRT4")
778         m_olsr_v5 = menu6.Append(ID_VIEW_OLSR_5, "View BBC-OLSR VIRT5", "View BBC-OLSR VIRT5")
779         m_olsr_v6 = menu6.Append(ID_VIEW_OLSR_6, "View BBC-OLSR VIRT6", "View BBC-OLSR VIRT6")
780
781         menu7 = wx.Menu()
782         m_usrp0A_table = menu7.Append(ID_VIEW_TABLE_USRP0A, "View Table USRP0A", "View Table
                USRP0A")
783         m_usrp0B_table = menu7.Append(ID_VIEW_TABLE_USRP0B, "View Table USRP0B", "View Table
                USRP0B")
784         m_usrp1A_table = menu7.Append(ID_VIEW_TABLE_USRP1A, "View Table USRP1A", "View Table
                USRP1A")
785         m_usrp1B_table = menu7.Append(ID_VIEW_TABLE_USRP1B, "View Table USRP1B", "View Table
                USRP1B")
786         m_virt1_table = menu7.Append(ID_VIEW_TABLE_VIRT1, "View Table VIRT1", "View Table VIRT1")
787         m_virt2_table = menu7.Append(ID_VIEW_TABLE_VIRT2, "View Table VIRT2", "View Table VIRT2")
788         m_virt3_table = menu7.Append(ID_VIEW_TABLE_VIRT3, "View Table VIRT3", "View Table VIRT3")
789         m_virt4_table = menu7.Append(ID_VIEW_TABLE_VIRT4, "View Table VIRT4", "View Table VIRT4")
790         m_virt5_table = menu7.Append(ID_VIEW_TABLE_VIRT5, "View Table VIRT5", "View Table VIRT5")
791         m_virt6_table = menu7.Append(ID_VIEW_TABLE_VIRT6, "View Table VIRT6", "View Table VIRT6")
792
793         #bind events for File menu
794         self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win1)
795         self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win2)
796         self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win3)
797         self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win4)
798         self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win5)
```

```
799             self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win6)
800             self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win7)
801             self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win8)
802             self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win9)
803             self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_win10)
804             self.Bind(wx.EVT_MENU, self.OnClearWindow, m_clear_winAll)
805             self.Bind(wx.EVT_MENU, self.OnResetQueue, m_reset)
806             self.Bind(wx.EVT_MENU, self.OnAbout, m_about)
807             self.Bind(wx.EVT_MENU, self.OnClose, m_exit)
808             #bind events for RX queue printing
809             self.Bind(wx.EVT_MENU, self.printQueue, m_rx0A)
810             self.Bind(wx.EVT_MENU, self.printQueue, m_rx0B)
811             self.Bind(wx.EVT_MENU, self.printQueue, m_rx1A)
812             self.Bind(wx.EVT_MENU, self.printQueue, m_rx1B)
813             self.Bind(wx.EVT_MENU, self.printQueue, m_rxv1)
814             self.Bind(wx.EVT_MENU, self.printQueue, m_rxv2)
815             self.Bind(wx.EVT_MENU, self.printQueue, m_rxv3)
816             self.Bind(wx.EVT_MENU, self.printQueue, m_rxv4)
817             self.Bind(wx.EVT_MENU, self.printQueue, m_rxv5)
818             self.Bind(wx.EVT_MENU, self.printQueue, m_rxv6)
819             #bind events for TX queue printing
820             self.Bind(wx.EVT_MENU, self.printQueue, m_tx0A)
821             self.Bind(wx.EVT_MENU, self.printQueue, m_tx0B)
822             self.Bind(wx.EVT_MENU, self.printQueue, m_tx1A)
823             self.Bind(wx.EVT_MENU, self.printQueue, m_tx1B)
824             self.Bind(wx.EVT_MENU, self.printQueue, m_txv1)
825             self.Bind(wx.EVT_MENU, self.printQueue, m_txv2)
826             self.Bind(wx.EVT_MENU, self.printQueue, m_txv3)
827             self.Bind(wx.EVT_MENU, self.printQueue, m_txv4)
828             self.Bind(wx.EVT_MENU, self.printQueue, m_txv5)
829             self.Bind(wx.EVT_MENU, self.printQueue, m_txv6)
830             #bind events for historical RX queue printing
831             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_0A)
832             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_0B)
833             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_1A)
834             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_1B)
835             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_v1)
836             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_v2)
837             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_v3)
838             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_v4)
839             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_v5)
840             self.Bind(wx.EVT_MENU, self.printQueue, m_rx_hist_v6)
841             #bind events for historical TX queue printing
842             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_0A)
843             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_0B)
```

```
844             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_1A)
845             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_1B)
846             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_v1)
847             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_v2)
848             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_v3)
849             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_v4)
850             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_v5)
851             self.Bind(wx.EVT_MENU, self.printQueue, m_tx_hist_v6)
852             #bind events for bbc-olsr process viewing in separate window
853             self.Bind(wx.EVT_MENU, self.printOLSRProcess,m_olsr_v1)
854             self.Bind(wx.EVT_MENU, self.printOLSRProcess,m_olsr_v2)
855             self.Bind(wx.EVT_MENU, self.printOLSRProcess,m_olsr_v3)
856             self.Bind(wx.EVT_MENU, self.printOLSRProcess,m_olsr_v4)
857             self.Bind(wx.EVT_MENU, self.printOLSRProcess,m_olsr_v5)
858             self.Bind(wx.EVT_MENU, self.printOLSRProcess,m_olsr_v6)
859             #bind events for testing-level neighbor table viewing in separate window
860             self.Bind(wx.EVT_MENU, self.printTable, m_usrp0A_table)
861             self.Bind(wx.EVT_MENU, self.printTable, m_usrp0B_table)
862             self.Bind(wx.EVT_MENU, self.printTable, m_usrp1A_table)
863             self.Bind(wx.EVT_MENU, self.printTable, m_usrp1B_table)
864             self.Bind(wx.EVT_MENU, self.printTable, m_virt1_table)
865             self.Bind(wx.EVT_MENU, self.printTable, m_virt2_table)
866             self.Bind(wx.EVT_MENU, self.printTable, m_virt3_table)
867             self.Bind(wx.EVT_MENU, self.printTable, m_virt4_table)
868             self.Bind(wx.EVT_MENU, self.printTable, m_virt5_table)
869             self.Bind(wx.EVT_MENU, self.printTable, m_virt6_table)
870
871             #add menus to menuBar
872             menuBar.Append(menu, "&File")
873             menuBar.Append(menu2, "RX Queues")
874             menuBar.Append(menu3, "TX Queues")
875             menuBar.Append(menu4, "RX Queues (H)")
876             menuBar.Append(menu5, "TX Queues (H)")
877             menuBar.Append(menu6, "BBC-OLSR")
878             menuBar.Append(menu7, "Neighbor Tables")
879
880             #activate menu bar
881             self.SetMenuBar(menuBar)
882             self.statusbar = self.CreateStatusBar()
883
884             #initialize offsets for controls in frame
885             self.x_offset = 0
886             self.y_offset = 0
887
888             #create controls, bind events, and do the layout
```

```
889             for i in range(4):
890                 self.createParameterControls()
891                 self.bindEvents()
892                 self.doLayout()
893                 self.y_offset = self.y_offset+200
894                 self.x_offset = self.x_offset + 250
895 
896             #initialize bbc-olsr processes
897             self.olsr_init()
898 
899             ################
900             #END OF __INIT__
901             ################
902 
903 
904         def olsr_init(self):
905             #print "Starting OLSR"
906             # open the tap interfaces, and bring them up with ifconfig
907             self.tap0 = tap_device("tap0", "192.168.2.21")
908             self.tap1 = tap_device("tap1", "192.168.3.22")
909             self.tap2 = tap_device("tap2", "192.168.4.23")
910             self.tap3 = tap_device("tap3", "192.168.5.24")
911 
912             #open tap interfaces for virtual bbc-olsr nodes
913             self.tap4 = tap_device("tap4", "192.168.6.25")
914             self.tap5 = tap_device("tap5", "192.168.7.26")
915             self.tap6 = tap_device("tap6", "192.168.8.27")
916             self.tap7 = tap_device("tap7", "192.168.9.28")
917             self.tap8 = tap_device("tap8", "192.168.10.29")
918             self.tap9 = tap_device("tap9", "192.168.11.30")
919 
920             time.sleep(1)
921 
922             #setup Network Address Translation table
923             self.setup_nat()
924 
925             #start interface reads and writes in tap_device threads
926             self.olsr_bring_up_interfaces()
927 
928             #start olsr daemons on each tap device
929             self.olsr_start_process()
930 
931 
932         def setup_nat(self):
933             #generate addresses for nodes
```

```
934             addr_list = []
935             for i in range(0,10):
936                 addr_list.append(random.randint(1,9999))
937 
938             self.nat_table = {"192.168.2.21": str(addr_list.pop()),
939                               "192.168.3.22": str(addr_list.pop()),
940                               "192.168.4.23": str(addr_list.pop()),
941                               "192.168.5.24": str(addr_list.pop()),
942                               #virtual interfaces
943                               "192.168.6.25": str(addr_list.pop()),
944                               "192.168.7.26": str(addr_list.pop()),
945                               "192.168.8.27": str(addr_list.pop()),
946                               "192.168.9.28": str(addr_list.pop()),
947                               "192.168.10.29": str(addr_list.pop()),
948                               "192.168.11.30": str(addr_list.pop())}
949 
950         def olsr_bring_up_interfaces(self):
951             #print "interfaces coming up"
952             self.tap0.start()
953             self.tap1.start()
954             self.tap2.start()
955             self.tap3.start()
956 
957             #virtual interfaces
958             self.tap4.start()
959             self.tap5.start()
960             self.tap6.start()
961             self.tap7.start()
962             self.tap8.start()
963             self.tap9.start()
964 
965 
966         def olsr_bring_down_interfaces(self):
967             #print "interfaces going down"
968             self.tap0.stop()
969             self.tap1.stop()
970             self.tap2.stop()
971             self.tap3.stop()
972 
973             #virtual interfaces
974             self.tap4.stop()
975             self.tap5.stop()
976             self.tap6.stop()
977             self.tap7.stop()
978             self.tap8.stop()
```

```
979             self.tap9.stop()
980
981         def olsr_stop_tap0(self):
982             self.tap0.stop()
983         def olsr_stop_tap1(self):
984             self.tap1.stop()
985         def olsr_stop_tap2(self):
986             self.tap2.stop()
987         def olsr_stop_tap3(self):
988             self.tap3.stop()
989         def olsr_stop_tap4(self):  #virtual node
990             self.tap4.stop()
991         def olsr_stop_tap5(self):  #virtual node
992             self.tap5.stop()
993         def olsr_stop_tap6(self):  #virtual node
994             self.tap6.stop()
995         def olsr_stop_tap7(self):  #virtual node
996             self.tap7.stop()
997         def olsr_stop_tap8(self):  #virtual node
998             self.tap8.stop()
999         def olsr_stop_tap9(self):  #virtual node
1000            self.tap9.stop()
1001
1002        def olsr_start_process(self):
1003            if (self.olsr_process_usb0A is None):
1004                #sudo olsrd -f /usr/local/etc/olsrd.conf -i tap0 -d 4
1005                self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd0.conf -i tap0 -d 4 -hint 5.0 -tcint 7.0
                        "
1006                self.olsr_process_usb0A = wx.Process(self, id=ID_PROCESS_OLSR_1)
1007                self.olsr_process_usb0A.Redirect()
1008                self.olsr_process_id_usb0A = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                        olsr_process_usb0A)
1009                self.display0A(">Starting BBC-OLSR Daemon\n")
1010                self.display0A(">Executing: "+self.olsr_cmd+"\n")
1011            if(self.olsr_process_usb0B is None):
1012                self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd1.conf -i tap1 -d 4 -hint 5.0 -tcint 7.0
                        "
1013                self.olsr_process_usb0B = wx.Process(self, id=ID_PROCESS_OLSR_2)
1014                self.olsr_process_usb0B.Redirect()
1015                self.olsr_process_id_usb0B = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                        olsr_process_usb0B)
1016                self.display0B(">Starting BBC-OLSR Daemon\n")
1017                self.display0B(">Executing: "+self.olsr_cmd+"\n")
1018            if(self.olsr_process_usb1A is None):
```

```
1019                self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd2.conf -i tap2 -d 4 -hint 5.0 -tcint 7.0
                        "
1020                self.olsr_process_usb1A = wx.Process(self, id=ID_PROCESS_OLSR_3)
1021                self.olsr_process_usb1A.Redirect()
1022                self.olsr_process_id_usb1A = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                        olsr_process_usb1A)
1023                self.display1A(">Starting BBC-OLSR Daemon\n")
1024                self.display1A(">Executing: "+self.olsr_cmd+"\n")
1025            if(self.olsr_process_usb1B is None):
1026                self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd3.conf -i tap3 -d 4 -hint 5.0 -tcint 7.0
                        "
1027                self.olsr_process_usb1B = wx.Process(self, id=ID_PROCESS_OLSR_4)
1028                self.olsr_process_usb1B.Redirect()
1029                self.olsr_process_id_usb1B = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                        olsr_process_usb1B)
1030                self.display1B(">Starting BBC-OLSR Daemon\n")
1031                self.display1B(">Executing: "+self.olsr_cmd+"\n")
1032
1033            #start virtual bbc-olsr processes
1034            if(self.olsr_process_virt1 is None):
1035                self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd4.conf -i tap4 -d 4 -hint 5.0 -tcint 7.0
                        "
1036                self.olsr_process_virt1 = wx.Process(self, id=ID_PROCESS_OLSR_5)
1037                self.olsr_process_virt1.Redirect()
1038                self.olsr_process_id_v1 = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                        olsr_process_virt1)
1039                self.olsr_v1_output+=">Starting BBC-OLSR Daemon\n"
1040                self.olsr_v1_output+=">Executing: "+self.olsr_cmd+"\n"
1041            if(self.olsr_process_virt2 is None):
1042                self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd5.conf -i tap5 -d 4 -hint 5.0 -tcint 7.0
                        "
1043                self.olsr_process_virt2 = wx.Process(self, id=ID_PROCESS_OLSR_6)
1044                self.olsr_process_virt2.Redirect()
1045                self.olsr_process_id_v2 = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                        olsr_process_virt2)
1046                self.olsr_v2_output+=">Starting BBC-OLSR Daemon\n"
1047                self.olsr_v2_output+=">Executing: "+self.olsr_cmd+"\n"
1048            if(self.olsr_process_virt3 is None):
1049                self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd6.conf -i tap6 -d 4 -hint 5.0 -tcint 7.0
                        "
1050                self.olsr_process_virt3 = wx.Process(self, id=ID_PROCESS_OLSR_7)
1051                self.olsr_process_virt3.Redirect()
1052                self.olsr_process_id_v3 = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                        olsr_process_virt3)
1053                self.olsr_v3_output+=">Starting BBC-OLSR Daemon\n"
```

```
1054            self.olsr_v3_output+=">Executing: "+self.olsr_cmd+"\n"
1055        if(self.olsr_process_virt4 is None):
1056            self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd7.conf -i tap7 -d 4 -hint 5.0 -tcint 7.0
                "
1057            self.olsr_process_virt4 = wx.Process(self, id=ID_PROCESS_OLSR_8)
1058            self.olsr_process_virt4.Redirect()
1059            self.olsr_process_id_v4 = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                olsr_process_virt4)
1060            self.olsr_v4_output+=">Starting BBC-OLSR Daemon\n"
1061            self.olsr_v4_output+=">Executing: "+self.olsr_cmd+"\n"
1062        if(self.olsr_process_virt5 is None):
1063            self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd8.conf -i tap8 -d 4 -hint 5.0 -tcint 7.0
                "
1064            self.olsr_process_virt5 = wx.Process(self, id=ID_PROCESS_OLSR_9)
1065            self.olsr_process_virt5.Redirect()
1066            self.olsr_process_id_v5 = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                olsr_process_virt5)
1067            self.olsr_v5_output+=">Starting BBC-OLSR Daemon\n"
1068            self.olsr_v5_output+=">Executing: "+self.olsr_cmd+"\n"
1069        if(self.olsr_process_virt6 is None):
1070            self.olsr_cmd = "olsrd -f /usr/local/etc/olsrd9.conf -i tap9 -d 4 -hint 5.0 -tcint 7.0
                "
1071            self.olsr_process_virt6 = wx.Process(self, id=ID_PROCESS_OLSR_10)
1072            self.olsr_process_virt6.Redirect()
1073            self.olsr_process_id_v6 = wx.Execute(self.olsr_cmd,wx.EXEC_ASYNC,self.
                olsr_process_virt6)
1074            self.olsr_v6_output+=">Starting BBC-OLSR Daemon\n"
1075            self.olsr_v6_output+=">Executing: "+self.olsr_cmd+"\n"
1076
1077
1078    def bindEvents(self):
1079        for control, event, handler in \
1080        [
1081         (self.updateButton, wx.EVT_BUTTON, self.onUpdate),
1082         (self.TxOrRx, wx.EVT_RADIOBOX, self.radioClick),
1083         (self.startButton, wx.EVT_BUTTON, self.startClick),
1084         (self.stopButton, wx.EVT_BUTTON, self.stopClick),
1085         (self, wx.EVT_IDLE, self.OnIdle),
1086         (self, wx.EVT_END_PROCESS,self.OnProcessEnded),
1087
1088         ]:
1089            control.Bind(event=event, handler=handler)
1090
1091
1092    def printQueue(self,event):
```

```
1093            #rx queue
1094            if(event.GetId()==ID_PRINT_RX_QUEUE_USB0A):
1095                queue = self.rx_pkt_queue_usrp0A
1096            elif(event.GetId()==ID_PRINT_RX_QUEUE_USB0B):
1097                queue = self.rx_pkt_queue_usrp0B
1098            elif(event.GetId()==ID_PRINT_RX_QUEUE_USB1A):
1099                queue = self.rx_pkt_queue_usrp1A
1100            elif(event.GetId()==ID_PRINT_RX_QUEUE_USB1B):
1101                queue = self.rx_pkt_queue_usrp1B
1102            elif(event.GetId()==ID_PRINT_RX_QUEUE_VIRT1):
1103                queue = self.rx_pkt_queue_virt1
1104            elif(event.GetId()==ID_PRINT_RX_QUEUE_VIRT2):
1105                queue = self.rx_pkt_queue_virt2
1106            elif(event.GetId()==ID_PRINT_RX_QUEUE_VIRT3):
1107                queue = self.rx_pkt_queue_virt3
1108            elif(event.GetId()==ID_PRINT_RX_QUEUE_VIRT4):
1109                queue = self.rx_pkt_queue_virt4
1110            elif(event.GetId()==ID_PRINT_RX_QUEUE_VIRT5):
1111                queue = self.rx_pkt_queue_virt5
1112            elif(event.GetId()==ID_PRINT_RX_QUEUE_VIRT6):
1113                queue = self.rx_pkt_queue_virt6
1114            #tx queue
1115            elif(event.GetId()==ID_PRINT_TX_QUEUE_USB0A):
1116                queue = self.tx_pkt_queue_usrp0A
1117            elif(event.GetId()==ID_PRINT_TX_QUEUE_USB0B):
1118                queue = self.tx_pkt_queue_usrp0B
1119            elif(event.GetId()==ID_PRINT_TX_QUEUE_USB1A):
1120                queue = self.tx_pkt_queue_usrp1A
1121            elif(event.GetId()==ID_PRINT_TX_QUEUE_USB1B):
1122                queue = self.tx_pkt_queue_usrp1B
1123            elif(event.GetId()==ID_PRINT_TX_QUEUE_VIRT1):
1124                queue = self.tx_pkt_queue_virt1
1125            elif(event.GetId()==ID_PRINT_TX_QUEUE_VIRT2):
1126                queue = self.tx_pkt_queue_virt2
1127            elif(event.GetId()==ID_PRINT_TX_QUEUE_VIRT3):
1128                queue = self.tx_pkt_queue_virt3
1129            elif(event.GetId()==ID_PRINT_TX_QUEUE_VIRT4):
1130                queue = self.tx_pkt_queue_virt4
1131            elif(event.GetId()==ID_PRINT_TX_QUEUE_VIRT5):
1132                queue = self.tx_pkt_queue_virt5
1133            elif(event.GetId()==ID_PRINT_TX_QUEUE_VIRT6):
1134                queue = self.tx_pkt_queue_virt6
1135            #rx queue historical
1136            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_USB0A):
1137                queue = self.rx_pkt_queue_historical_usrp0A
```

```
1138            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_USB0B):
1139                queue = self.rx_pkt_queue_historical_usrp0B
1140            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_USB1A):
1141                queue = self.rx_pkt_queue_historical_usrp1A
1142            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_USB1B):
1143                queue = self.rx_pkt_queue_historical_usrp1B
1144            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_VIRT1):
1145                queue = self.rx_pkt_queue_historical_virt1
1146            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_VIRT2):
1147                queue = self.rx_pkt_queue_historical_virt2
1148            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_VIRT3):
1149                queue = self.rx_pkt_queue_historical_virt3
1150            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_VIRT4):
1151                queue = self.rx_pkt_queue_historical_virt4
1152            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_VIRT5):
1153                queue = self.rx_pkt_queue_historical_virt5
1154            elif(event.GetId()==ID_PRINT_RX_QUEUE_HISTORICAL_VIRT6):
1155                queue = self.rx_pkt_queue_historical_virt6
1156            #tx queue historical
1157            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_USB0A):
1158                queue = self.tx_pkt_queue_historical_usrp0A
1159            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_USB0B):
1160                queue = self.tx_pkt_queue_historical_usrp0B
1161            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_USB1A):
1162                queue = self.tx_pkt_queue_historical_usrp1A
1163            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_USB1B):
1164                queue = self.tx_pkt_queue_historical_usrp1B
1165            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_VIRT1):
1166                queue = self.tx_pkt_queue_historical_virt1
1167            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_VIRT1):
1168                queue = self.tx_pkt_queue_historical_virt1
1169            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_VIRT2):
1170                queue = self.tx_pkt_queue_historical_virt2
1171            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_VIRT3):
1172                queue = self.tx_pkt_queue_historical_virt3
1173            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_VIRT4):
1174                queue = self.tx_pkt_queue_historical_virt4
1175            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_VIRT5):
1176                queue = self.tx_pkt_queue_historical_virt5
1177            elif(event.GetId()==ID_PRINT_TX_QUEUE_HISTORICAL_VIRT6):
1178                queue = self.tx_pkt_queue_historical_virt6
1179            else:
1180                print "Error: Print Queue Event ID Not Found!\n"
1181
1182            if queue is not None:
```

```
1183                for i in queue:
1184                    i.print_pkt()
1185
1186        #from menu, display running bbc-olsr process
1187        def printOLSRProcess(self, event):
1188            if(event.GetId()==ID_VIEW_OLSR_1):
1189                print self.olsr_v1_output
1190            elif(event.GetId()==ID_VIEW_OLSR_2):
1191                print self.olsr_v2_output
1192            elif(event.GetId()==ID_VIEW_OLSR_3):
1193                print self.olsr_v3_output
1194            elif(event.GetId()==ID_VIEW_OLSR_4):
1195                print self.olsr_v4_output
1196            elif(event.GetId()==ID_VIEW_OLSR_5):
1197                print self.olsr_v5_output
1198            elif(event.GetId()==ID_VIEW_OLSR_6):
1199                print self.olsr_v6_output
1200
1201
1202        #from menu, display neighbor table for testing purposes
1203        def printTable(self,event):
1204            if(event.GetId()==ID_VIEW_TABLE_USRP0A):
1205                print self.usrp0A_table.items()
1206            elif(event.GetId()==ID_VIEW_TABLE_USRP0B):
1207                print self.usrp0B_table.items()
1208            elif(event.GetId()==ID_VIEW_TABLE_USRP1A):
1209                print self.usrp1A_table.items()
1210            elif(event.GetId()==ID_VIEW_TABLE_USRP1B):
1211                print self.usrp1B_table.items()
1212            elif(event.GetId()==ID_VIEW_TABLE_VIRT1):
1213                print self.virt1_table.items()
1214            elif(event.GetId()==ID_VIEW_TABLE_VIRT2):
1215                print self.virt2_table.items()
1216            elif(event.GetId()==ID_VIEW_TABLE_VIRT3):
1217                print self.virt3_table.items()
1218            elif(event.GetId()==ID_VIEW_TABLE_VIRT4):
1219                print self.virt4_table.items()
1220            elif(event.GetId()==ID_VIEW_TABLE_VIRT5):
1221                print self.virt5_table.items()
1222            elif(event.GetId()==ID_VIEW_TABLE_VIRT6):
1223                print self.virt6_table.items()
1224
1225        def killOLSRProcesses(self):
1226            if(self.olsr_process_usb0A is not None):
1227                    res=self.olsr_process_usb0A.Kill(self.olsr_process_id_usb0A,2)
```

```
1228            if(self.olsr_process_usb0B is not None):
1229                    res=self.olsr_process_usb0B.Kill(self.olsr_process_id_usb0B,2)
1230            if(self.olsr_process_usb1A is not None):
1231                    res=self.olsr_process_usb1A.Kill(self.olsr_process_id_usb1A,2)
1232            if(self.olsr_process_usb1B is not None):
1233                    res=self.olsr_process_usb1B.Kill(self.olsr_process_id_usb1B,2)
1234            if(self.olsr_process_virt1 is not None):
1235                    res=self.olsr_process_virt1.Kill(self.olsr_process_id_v1,2)
1236            if(self.olsr_process_virt2 is not None):
1237                    res=self.olsr_process_virt2.Kill(self.olsr_process_id_v2,2)
1238            if(self.olsr_process_virt3 is not None):
1239                    res=self.olsr_process_virt3.Kill(self.olsr_process_id_v3,2)
1240            if(self.olsr_process_virt4 is not None):
1241                    res=self.olsr_process_virt4.Kill(self.olsr_process_id_v4,2)
1242            if(self.olsr_process_virt5 is not None):
1243                    res=self.olsr_process_virt5.Kill(self.olsr_process_id_v5,2)
1244            if(self.olsr_process_virt6 is not None):
1245                    res=self.olsr_process_virt6.Kill(self.olsr_process_id_v6,2)
1246
1247        #stop a node's complete operation
1248        def stopClick(self,event):
1249            if(event.GetId()==ID_STOP_USB0A):
1250                self.olsr_stop_tap0()
1251                if (self.process_usb0A is not None):
1252                    self.kill_0A = True
1253                    res=self.process_usb0A.Kill(self.process_id_usb0A,2)
1254                    self.display0A(">Terminated USRP0A Process.\n")
1255                else: self.display0A(">USRP0A process already terminated.\n")
1256                if(self.olsr_process_usb0A is not None):
1257                    res=self.olsr_process_usb0A.Kill(self.olsr_process_id_usb0A,2)
1258                    self.display0A(">Terminated OLSR Process.\n")
1259                else: self.display0A(">OLSR Process already terminated\n")
1260            elif(event.GetId()==ID_STOP_USB0B):
1261                self.olsr_stop_tap1()
1262                if (self.process_usb0B is not None):
1263                    self.kill_0B = True
1264                    res=self.process_usb0B.Kill(self.process_id_usb0B,2)
1265                    self.display0B(">Terminated USRP0B Process.\n")
1266                else: self.display0B(">USRP0B process already terminated.\n")
1267                if(self.olsr_process_usb0B is not None):
1268                    res=self.olsr_process_usb0B.Kill(self.olsr_process_id_usb0B,2)
1269                    self.display0B(">Terminated OLSR Process.\n")
1270                else: self.display0B(">OLSR Process already terminated.\n")
1271            elif(event.GetId()==ID_STOP_USB1A):
1272                self.olsr_stop_tap2()
```

```
1273            if (self.process_usb1A is not None):
1274                self.kill_1A = True
1275                res=self.process_usb1A.Kill(self.process_id_usb1A,2)
1276                self.display1A(">Terminated USRP1A Process.\n")
1277            else: self.display1A(">USRP1A process already terminated.\n")
1278            if(self.olsr_process_usb1A is not None):
1279                res=self.olsr_process_usb1A.Kill(self.olsr_process_id_usb1A,2)
1280                self.display1A(">Terminated OLSR Process.\n")
1281            else: self.display1A(">OLSR Process already terminated.\n")
1282        elif(event.GetId()==ID_STOP_USB1B):
1283            self.olsr_stop_tap3()
1284            if (self.process_usb1B is not None):
1285                self.kill_1B = True
1286                res=self.process_usb1B.Kill(self.process_id_usb1B,2)
1287                self.display1B(">Terminated USRP1B Process.\n")
1288            else: self.display1B(">USRP1B process already terminated.\n")
1289            if(self.olsr_process_usb1B is not None):
1290                res=self.olsr_process_usb1B.Kill(self.olsr_process_id_usb1B,2)
1291                self.display1B(">Terminated OLSR Process.\n")
1292            else: self.display1B(">OLSR Process already terminated.\n")
1293        #virtual nodes
1294        elif(event.GetId()==ID_STOP_VIRT1):
1295            self.olsr_stop_tap4()
1296            if (self.process_virt1 is not None):
1297                #self.kill_1B = True
1298                res=self.process_virt1.Kill(self.process_id_virt1,2)
1299                self.olsr_v1_output+=">Terminated VIRT1 Process.\n"
1300            else: self.olsr_v1_output+=">VIRT1 process already terminated.\n"
1301            if(self.olsr_process_virt1 is not None):
1302                res=self.olsr_process_virt1.Kill(self.olsr_process_id_virt1,2)
1303                self.olsr_v1_output+=">Terminated OLSR Process.\n"
1304            else: self.olsr_v1_output+=">OLSR Process already terminated.\n"
1305        elif(event.GetId()==ID_STOP_VIRT2):
1306            self.olsr_stop_tap5()
1307            if (self.process_virt2 is not None):
1308                #self.kill_2B = True
1309                res=self.process_virt2.Kill(self.process_id_virt2,2)
1310                self.olsr_v2_output+=">Terminated VIRT2 Process.\n"
1311            else: self.olsr_v2_output+=">VIRT2 process already terminated.\n"
1312            if(self.olsr_process_virt2 is not None):
1313                res=self.olsr_process_virt2.Kill(self.olsr_process_id_virt2,2)
1314                self.olsr_v2_output+=">Terminated OLSR Process.\n"
1315            else: self.olsr_v2_output+=">OLSR Process already terminated.\n"
1316        elif(event.GetId()==ID_STOP_VIRT3):
1317            self.olsr_stop_tap6()
```

```
1318                if (self.process_virt3 is not None):
1319                    #self.kill_3B = True
1320                    res=self.process_virt3.Kill(self.process_id_virt3,2)
1321                    self.olsr_v3_output+=">Terminated VIRT3 Process.\n"
1322                else: self.olsr_v3_output+=">VIRT3 process already terminated.\n"
1323                if(self.olsr_process_virt3 is not None):
1324                    res=self.olsr_process_virt3.Kill(self.olsr_process_id_virt3,2)
1325                    self.olsr_v3_output+=">Terminated OLSR Process.\n"
1326                else: self.olsr_v3_output+=">OLSR Process already terminated.\n"
1327            elif(event.GetId()==ID_STOP_VIRT4):
1328                self.olsr_stop_tap7()
1329                if (self.process_virt4 is not None):
1330                    #self.kill_4B = True
1331                    res=self.process_virt4.Kill(self.process_id_virt4,2)
1332                    self.olsr_v4_output+=">Terminated VIRT4 Process.\n"
1333                else: self.olsr_v4_output+=">VIRT4 process already terminated.\n"
1334                if(self.olsr_process_virt4 is not None):
1335                    res=self.olsr_process_virt4.Kill(self.olsr_process_id_virt4,2)
1336                    self.olsr_v4_output+=">Terminated OLSR Process.\n"
1337                else: self.olsr_v4_output+=">OLSR Process already terminated.\n"
1338            elif(event.GetId()==ID_STOP_VIRT5):
1339                self.olsr_stop_tap8()
1340                if (self.process_virt5 is not None):
1341                    #self.kill_5B = True
1342                    res=self.process_virt5.Kill(self.process_id_virt5,2)
1343                    self.olsr_v5_output+=">Terminated VIRT5 Process.\n"
1344                else: self.olsr_v5_output+=">VIRT5 process already terminated.\n"
1345                if(self.olsr_process_virt5 is not None):
1346                    res=self.olsr_process_virt5.Kill(self.olsr_process_id_virt5,2)
1347                    self.olsr_v5_output+=">Terminated OLSR Process.\n"
1348                else: self.olsr_v5_output+=">OLSR Process already terminated.\n"
1349            elif(event.GetId()==ID_STOP_VIRT6):
1350                self.olsr_stop_tap9()
1351                if (self.process_virt6 is not None):
1352                    #self.kill_6B = True
1353                    res=self.process_virt6.Kill(self.process_id_virt6,2)
1354                    self.olsr_v6_output+=">Terminated VIRT6 Process.\n"
1355                else: self.olsr_v6_output+=">VIRT6 process already terminated.\n"
1356                if(self.olsr_process_virt6 is not None):
1357                    res=self.olsr_process_virt6.Kill(self.olsr_process_id_virt6,2)
1358                    self.olsr_v6_output+=">Terminated OLSR Process.\n"
1359                else: self.olsr_v6_output+=">OLSR Process already terminated.\n"
1360
1361
1362        def startClick(self,event):
```

```
1363            if(event.GetId()==ID_START_USB0A):
1364                node_id = self.FindWindowById(ID_RADIO_USB0A).GetLabelText()
1365                if (self.process_usb0A is None):
1366                    if(self.FindWindowById(ID_TXRX_USB0A).GetStringSelection()=="TX"):
1367                        if(self.process_usb0B is not None and self.FindWindowById(ID_TXRX_USB0B).
                                GetStringSelection()=="TX"):
1368                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                    getStopButtonId("usrp0B"))
1369                            self.display0B("\n>Stopping this transmission to restart TX block to TX
                                    on both daughterboards.\n")
1370                            self.txt_cmd = self.experiment_path+ "usrp"+node_id[3]+node_id[4]+"/
                                    bbc_tx.py -U " + node_id[3] + " -T " + node_id[4] + " -C 2 -f 1250M -
                                    i 64" + " -S " + self.experiment_path+"usrp0A/"+self.FindWindowById(
                                    ID_SINK_NAME_USB0A).GetValue() + " -P B -N " + self.experiment_path+
                                    "usrp0B/"+self.FindWindowById(ID_SINK_NAME_USB0B).GetValue()
1371                            self.display0A("\n>Transmitting...\n")
1372                            self.display0B("\n>Transmitting...\n")
1373                        else:
1374                            self.display0A("\nTransmitting...\n")
1375                            self.txt_cmd = self.experiment_path+ "usrp"+node_id[3]+node_id[4]+"/bbc_tx
                                    .py -U " + node_id[3] + " -T " + node_id[4] + " -f 1250M -i 64" + " -S
                                    " + self.experiment_path+"usrp0A/"+self.FindWindowById(
                                    ID_SINK_NAME_USB0A).GetValue()
1376                    else: #0A is in RX mode
1377                        if(self.process_usb0B is not None and self.FindWindowById(ID_TXRX_USB0B).
                                GetStringSelection()=="RX"):
1378                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                    getStopButtonId("usrp0B"))
1379                            self.dual_rx_0 = True
1380                            self.display0B("\n>Stopping this transmission to restart RX block to RX
                                    on both daughterboards.\n")
1381                            self.txt_cmd = self.experiment_path+ "usrp0A/usrp_rx_cfile.py -U " +
                                    node_id[3] + " -R " + node_id[4] + " -C 2 -f 1250M -d 32 " + self.
                                    experiment_path+"usrp0A/"+self.FindWindowById(ID_SRC_NAME_USB0A).
                                    GetValue() + " " + self.experiment_path+"usrp0B/"+self.FindWindowById
                                    (ID_SRC_NAME_USB0B).GetValue()
1382                            self.display0A("\n>Receiving...\n")
1383                            self.display0B("\n>Receiving...\n")
1384                        else:
1385                            self.display0A("\n>Receiving...\n")
1386                            self.txt_cmd = self.experiment_path+ "usrp0A/usrp_rx_cfile.py -U " +
                                    node_id[3] + " -R " + node_id[4] + " -f 1250M -d 32 " + self.
                                    experiment_path+"usrp0A/"+self.FindWindowById(ID_SRC_NAME_USB0A).
                                    GetValue()
1387                    self.process_usb0A = wx.Process(self,id=ID_PROCESS_1)
```

```
1388                    self.process_usb0A.Redirect()
1389                    self.process_id_usb0A = wx.Execute(self.txt_cmd,wx.EXEC_ASYNC,self.process_usb0A)
1390                    self.display0A("\n>Executing: "+self.txt_cmd+"\n")
1391
1392            elif(event.GetId()==ID_START_USB0B):
1393                node_id = self.FindWindowById(ID_RADIO_USB0B).GetLabelText()
1394                if (self.process_usb0B is None):
1395                    if(self.FindWindowById(ID_TXRX_USB0B).GetStringSelection()=="TX"):
1396                        if(self.process_usb0A is not None and self.FindWindowById(ID_TXRX_USB0B).
                                GetStringSelection()=="TX"):
1397                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                    getStopButtonId("usrp0A"))
1398                            self.display0A("\n>Stopping this transmission to restart TX block to TX
                                    on both daughterboards.\n")
1399                            self.txt_cmd = self.experiment_path+ "usrp"+node_id[3]+node_id[4]+"/
                                    bbc_tx.py -U " + node_id[3] + " -T " + node_id[4] + " -C 2 -f 1250M -
                                    i 64" + " -S " + self.experiment_path+"usrp0B/"+self.FindWindowById(
                                    ID_SINK_NAME_USB0B).GetValue()  + " -P A -N " + self.experiment_path+
                                    "usrp0A/"+self.FindWindowById(ID_SINK_NAME_USB0A).GetValue()
1400                            self.display0A("\n>Transmitting...\n")
1401                            self.display0B("\n>Transmitting...\n")
1402                        else:
1403                            self.display0B("\nTransmitting...\n")
1404                            self.txt_cmd = self.experiment_path+ "usrp0B/bbc_tx.py -U " + node_id[3]
                                    + " -T " + node_id[4] + " -f 1250M -i 64" + " -S " + self.
                                    experiment_path+"usrp0B/"+self.FindWindowById(ID_SINK_NAME_USB0B).
                                    GetValue()
1405                    else:
1406                        if(self.process_usb0A is not None and self.FindWindowById(ID_TXRX_USB0A).
                                GetStringSelection()=="RX"):
1407                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                    getStopButtonId("usrp0A"))
1408                            self.dual_rx_0 = True
1409                            self.display0A("\n>Stopping this transmission to restart RX block to RX
                                    on both daughterboards.\n")
1410                            self.txt_cmd = self.experiment_path+ "usrp0B/usrp_rx_cfile.py -U " +
                                    node_id[3] + " -R " + node_id[4] + " -C 2 -f 1250M -d 32 " + self.
                                    experiment_path+"usrp0A/"+self.FindWindowById(ID_SRC_NAME_USB0A).
                                    GetValue() + " " + self.experiment_path+"usrp0B/"+self.FindWindowById
                                    (ID_SRC_NAME_USB0B).GetValue()
1411                            self.display0A("\n>Receiving...\n")
1412                            self.display0B("\n>Receiving...\n")
1413                        else:
1414                            self.display0B("\nReceiving...\n")
```

```
1415                            self.txt_cmd = self.experiment_path+ "usrp0B/usrp_rx_cfile.py -U " +
                                    node_id[3] + " -R " + node_id[4] + " -f 1250M -d 32 " + self.
                                    experiment_path+"usrp0B/"+self.FindWindowById(ID_SRC_NAME_USB0B).
                                    GetValue()
1416                    self.process_usb0B = wx.Process(self,id=ID_PROCESS_2)
1417                    self.process_usb0B.Redirect()
1418                    self.process_id_usb0B = wx.Execute(self.txt_cmd,wx.EXEC_ASYNC,self.process_usb0B)
1419                    self.display0B("\n>Executing: "+self.txt_cmd+"\n")
1420            elif(event.GetId()==ID_START_USB1A):
1421                node_id = self.FindWindowById(ID_RADIO_USB1A).GetLabelText()
1422                if (self.process_usb1A is None):
1423                    if(self.FindWindowById(ID_TXRX_USB1A).GetStringSelection()=="TX"):
1424                        if(self.process_usb1B is not None and self.FindWindowById(ID_TXRX_USB1B).
                                GetStringSelection()=="TX"):
1425                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                    getStopButtonId("usrp1B"))
1426                            self.display1B("\n>Stopping this transmission to restart TX block to TX on
                                    both daughterboards.\n")
1427                            self.txt_cmd = self.experiment_path+ "usrp"+node_id[3]+node_id[4]+"/bbc_tx
                                    .py -U " + node_id[3] + " -T " + node_id[4] + " -C 2 -f 1250M -i 64" +
                                    " -S " + self.experiment_path+"usrp1A/"+self.FindWindowById(
                                    ID_SINK_NAME_USB1A).GetValue()  + " -P B -N " + self.experiment_path+"
                                    usrp1B/"+self.FindWindowById(ID_SINK_NAME_USB1B).GetValue()
1428                            self.display1A("\n>Transmitting...\n")
1429                            self.display1B("\n>Transmitting...\n")
1430                        else:
1431                            self.display1A("\nTransmitting...\n")
1432                            self.txt_cmd = self.experiment_path+ "usrp1A/bbc_tx.py -U " + node_id[3] +
                                    " -T " + node_id[4] + " -f 1250M -i 64" + " -S " + self.
                                    experiment_path+"usrp1A/"+self.FindWindowById(ID_SINK_NAME_USB1A).
                                    GetValue()
1433                    else:
1434                        if(self.process_usb1B is not None and self.FindWindowById(ID_TXRX_USB1B).
                                GetStringSelection()=="RX"):
1435                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                    getStopButtonId("usrp1B"))
1436                            self.dual_rx_1 = True
1437                            self.display1B("\n>Stopping this transmission to restart RX block to RX
                                    on both daughterboards.\n")
1438                            self.txt_cmd = self.experiment_path+ "usrp1A/usrp_rx_cfile.py -U " +
                                    node_id[3] + " -R " + node_id[4] + " -C 2 -f 1250M -d 32 " + self.
                                    experiment_path+"usrp1A/"+self.FindWindowById(ID_SRC_NAME_USB1A).
                                    GetValue() + " " + self.experiment_path+"usrp1B/"+self.FindWindowById
                                    (ID_SRC_NAME_USB1B).GetValue()
1439                            self.display1A("\n>Receiving...\n")
```

```
1440                        self.display1B("\n>Receiving...\n")
1441                    else:
1442                        self.display1A("\nReceiving...\n")
1443                        self.txt_cmd = self.experiment_path+ "usrp1A/usrp_rx_cfile.py -U " +
                                node_id[3] + " -R " + node_id[4] + " -f 1250M -d 32 " + self.
                                experiment_path+"usrp1A/"+self.FindWindowById(ID_SRC_NAME_USB1A).
                                GetValue()
1444                self.process_usb1A = wx.Process(self,id=ID_PROCESS_3)
1445                self.process_usb1A.Redirect()
1446                self.process_id_usb1A = wx.Execute(self.txt_cmd,wx.EXEC_ASYNC,self.process_usb1A)
1447                self.display1A("\n>Executing: "+self.txt_cmd+"\n")
1448            elif(event.GetId()==ID_START_USB1B):
1449                node_id = self.FindWindowById(ID_RADIO_USB1B).GetLabelText()
1450                if (self.process_usb1B is None):
1451                    if(self.FindWindowById(ID_TXRX_USB1B).GetStringSelection()=="TX"):
1452                        if(self.process_usb1A is not None and self.FindWindowById(ID_TXRX_USB1A).
                                GetStringSelection()=="TX"):
1453                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                getStopButtonId("usrp1A"))
1454                            self.display1A("\n>Stopping this transmission to restart TX block to TX
                                on both daughterboards.\n")
1455                            self.txt_cmd = self.experiment_path+ "usrp"+node_id[3]+node_id[4]+"/
                                bbc_tx.py -U " + node_id[3] + " -T " + node_id[4] + " -C 2 -f 1250M -
                                i 64" + " -S " + self.experiment_path+"usrp1B/"+self.FindWindowById(
                                ID_SINK_NAME_USB1B).GetValue() + " -P A -N " + self.experiment_path+
                                "usrp1A/"+self.FindWindowById(ID_SINK_NAME_USB1A).GetValue()
1456                            self.display1A("\n>Transmitting...\n")
1457                            self.display1B("\n>Transmitting...\n")
1458                        else:
1459                            self.display1B("\nTransmitting...\n")
1460                            self.txt_cmd = self.experiment_path+ "usrp1B/bbc_tx.py -U " + node_id[3]
                                + " -T " + node_id[4] + " -f 1250M -i 64" + " -S " + self.
                                experiment_path+"usrp1B/"+self.FindWindowById(ID_SINK_NAME_USB1B).
                                GetValue()
1461                    else:
1462                        if(self.process_usb1A is not None and self.FindWindowById(ID_TXRX_USB1A).
                                GetStringSelection()=="RX"):
1463                            self.SendEvent(mycontrol=self.stopButton, event=wx.EVT_BUTTON, id=self.
                                getStopButtonId("usrp1A"))
1464                            self.dual_rx_1 = True
1465                            self.display1A("\n>Stopping this transmission to restart RX block to RX
                                on both daughterboards.\n")
```

```
1466                            self.txt_cmd = self.experiment_path+ "usrp1B/usrp_rx_cfile.py -U " +
                                    node_id[3] + " -R " + node_id[4] + " -C 2 -f 1250M -d 32 " + self.
                                    experiment_path+"usrp1B/"+self.FindWindowById(ID_SRC_NAME_USB1B).
                                    GetValue() + " " + self.experiment_path+"usrp1A/"+self.FindWindowById
                                    (ID_SRC_NAME_USB1A).GetValue()
1467                            self.display1A("\n>Receiving...\n")
1468                            self.display1B("\n>Receiving...\n")
1469                        else:
1470                            self.display1B("\nReceiving...\n")
1471                            self.txt_cmd = self.experiment_path+ "usrp1B/usrp_rx_cfile.py -U " +
                                    node_id[3] + " -R " + node_id[4] + " -f 1250M -d 32 " + self.
                                    experiment_path+"usrp1B/"+self.FindWindowById(ID_SRC_NAME_USB1B).
                                    GetValue()
1472                    self.process_usb1B = wx.Process(self,id=ID_PROCESS_4)
1473                    self.process_usb1B.Redirect()
1474                    self.process_id_usb1B = wx.Execute(self.txt_cmd,wx.EXEC_ASYNC,self.process_usb1B)
1475                    self.display1B("\n>Executing: "+self.txt_cmd+"\n")
1476
1477
1478        def CheckTxQueue(self):
1479
1480            if(len(self.tx_pkt_queue_usrp0A)>0):
1481                #only called after the end of a TX, so no need to stop the current TX
1482                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("usrp0A"))
1483                #receivers are not turned on at this point
1484                #use transmitToFrom instead?
1485                #or, make sure the other nodes are put into a receive mode by default?
1486            #else put in RX mode?
1487                #stop the TX, set RX radio box, update, and do the start button
1488            #else:
1489                #if the RSSI is above threshold, start RX cycle.
1490            if(len(self.tx_pkt_queue_usrp0B)>0):
1491                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("usrp0B"))
1492            if(len(self.tx_pkt_queue_usrp1A)>0):
1493                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("usrp1A"))
1494            if(len(self.tx_pkt_queue_usrp1B)>0):
1495                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("usrp1B"))
1496            #virtual nodes
1497            if(len(self.tx_pkt_queue_virt1)>0):
1498                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("virt1"))
```

```
1499            if(len(self.tx_pkt_queue_virt2)>0):
1500                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("virt2"))
1501            if(len(self.tx_pkt_queue_virt3)>0):
1502                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("virt3"))
1503            if(len(self.tx_pkt_queue_virt4)>0):
1504                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("virt4"))
1505            if(len(self.tx_pkt_queue_virt5)>0):
1506                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("virt5"))
1507            if(len(self.tx_pkt_queue_virt6)>0):
1508                self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.
                        getStartButtonId("virt6"))
1509
1510        #check idle events for the many many processes
1511        def OnIdle( self, event):
1512
1513            #processing signals physical nodes
1514            if (self.process_usb0A is not None):
1515                stream_usb0A = self.process_usb0A.GetInputStream()
1516                if(stream_usb0A.CanRead()):
1517                    text_usb0A = stream_usb0A.read()
1518                    self.display0A(text_usb0A)
1519            if (self.process_usb0B is not None):
1520                stream_usb0B = self.process_usb0B.GetInputStream()
1521                if(stream_usb0B.CanRead()):
1522                    text_usb0B = stream_usb0B.read()
1523                    self.display0B(text_usb0B)
1524            if (self.process_usb1A is not None):
1525                stream_usb1A = self.process_usb1A.GetInputStream()
1526                if(stream_usb1A.CanRead()):
1527                    text_usb1A = stream_usb1A.read()
1528                    self.display1A(text_usb1A)
1529            if (self.process_usb1B is not None):
1530                stream_usb1B = self.process_usb1B.GetInputStream()
1531                if(stream_usb1B.CanRead()):
1532                    text_usb1B = stream_usb1B.read()
1533                    self.display1B(text_usb1B)
1534
1535            #processing signals virtual nodes
1536            if (self.process_virt1 is not None):
1537                stream_virt1 = self.process_virt1.GetInputStream()
1538                if(stream_virt1.CanRead()):
```

```
                text_virt1 = stream_virt1.read()
                self.olsr_v1_output+=text_virt1
        if (self.process_virt2 is not None):
            stream_virt2 = self.process_virt2.GetInputStream()
            if(stream_virt2.CanRead()):
                text_virt2 = stream_virt2.read()
                self.olsr_v2_output+=text_virt2
        if (self.process_virt3 is not None):
            stream_virt3 = self.process_virt3.GetInputStream()
            if(stream_virt3.CanRead()):
                text_virt3 = stream_virt3.read()
                self.olsr_v3_output+=text_virt3
        if (self.process_virt4 is not None):
            stream_virt4 = self.process_virt4.GetInputStream()
            if(stream_virt4.CanRead()):
                text_virt4 = stream_virt4.read()
                self.olsr_v4_output+=text_virt4
        if (self.process_virt5 is not None):
            stream_virt5 = self.process_virt5.GetInputStream()
            if(stream_virt5.CanRead()):
                text_virt5 = stream_virt5.read()
                self.olsr_v5_output+=text_virt5
        if (self.process_virt6 is not None):
            stream_virt6 = self.process_virt6.GetInputStream()
            if(stream_virt6.CanRead()):
                text_virt6 = stream_virt6.read()
                self.olsr_v6_output+=text_virt6

        #encoding physical nodes
        if(self.encode_process_usb0A is not None):
            stream_encode_usb0A = self.encode_process_usb0A.GetInputStream()
            if(stream_encode_usb0A.CanRead()):
                text_encode_usb0A = stream_encode_usb0A.read()
                self.display0A(text_encode_usb0A)
        if(self.encode_process_usb0B is not None):
            stream_encode_usb0B = self.encode_process_usb0B.GetInputStream()
            if(stream_encode_usb0B.CanRead()):
                text_encode_usb0B = stream_encode_usb0B.read()
                self.display0B(text_encode_usb0B)
        if(self.encode_process_usb1A is not None):
            stream_encode_usb1A = self.encode_process_usb1A.GetInputStream()
            if(stream_encode_usb1A.CanRead()):
                text_encode_usb1A = stream_encode_usb1A.read()
                self.display1A(text_encode_usb1A)
        if(self.encode_process_usb1B is not None):
```

```
1584                stream_encode_usb1B = self.encode_process_usb1B.GetInputStream()
1585                if(stream_encode_usb1B.CanRead()):
1586                    text_encode_usb1B = stream_encode_usb1B.read()
1587                    self.display1B(text_encode_usb1B)
1588
1589            #encoding virtual nodes
1590            if(self.encode_process_virt1 is not None):
1591                stream_encode_virt1 = self.encode_process_virt1.GetInputStream()
1592                if(stream_encode_virt1.CanRead()):
1593                    text_encode_virt1 = stream_encode_virt1.read()
1594                    self.olsr_v1_output+=text_encode_virt1
1595            if(self.encode_process_virt2 is not None):
1596                stream_encode_virt2 = self.encode_process_virt2.GetInputStream()
1597                if(stream_encode_virt2.CanRead()):
1598                    text_encode_virt2 = stream_encode_virt2.read()
1599                    self.olsr_v2_output+=text_encode_virt2
1600            if(self.encode_process_virt3 is not None):
1601                stream_encode_virt3 = self.encode_process_virt3.GetInputStream()
1602                if(stream_encode_virt3.CanRead()):
1603                    text_encode_virt3 = stream_encode_virt3.read()
1604                    self.olsr_v3_output+=text_encode_virt3
1605            if(self.encode_process_virt4 is not None):
1606                stream_encode_virt4 = self.encode_process_virt4.GetInputStream()
1607                if(stream_encode_virt4.CanRead()):
1608                    text_encode_virt4 = stream_encode_virt4.read()
1609                    self.olsr_v4_output+=text_encode_virt4
1610            if(self.encode_process_virt5 is not None):
1611                stream_encode_virt5 = self.encode_process_virt5.GetInputStream()
1612                if(stream_encode_virt5.CanRead()):
1613                    text_encode_virt5 = stream_encode_virt5.read()
1614                    self.olsr_v5_output+=text_encode_virt5
1615            if(self.encode_process_virt6 is not None):
1616                stream_encode_virt6 = self.encode_process_virt6.GetInputStream()
1617                if(stream_encode_virt6.CanRead()):
1618                    text_encode_virt6 = stream_encode_virt6.read()
1619                    self.olsr_v6_output+=text_encode_virt6
1620
1621            #decoding physical nodes
1622            if(self.decode_process_usb0A is not None):
1623                stream_decode_usb0A = self.decode_process_usb0A.GetInputStream()
1624                if(stream_decode_usb0A.CanRead()):
1625                    text_decode_usb0A = stream_decode_usb0A.read()
1626                    self.display0A(text_decode_usb0A)
1627            if(self.decode_process_usb0B is not None):
1628                stream_decode_usb0B = self.decode_process_usb0B.GetInputStream()
```

```
1629                 if(stream_decode_usb0B.CanRead()):
1630                     text_decode_usb0B = stream_decode_usb0B.read()
1631                     self.display0B(text_decode_usb0B)
1632             if(self.decode_process_usb1A is not None):
1633                 stream_decode_usb1A = self.decode_process_usb1A.GetInputStream()
1634                 if(stream_decode_usb1A.CanRead()):
1635                     text_decode_usb1A = stream_decode_usb1A.read()
1636                     self.display1A(text_decode_usb1A)
1637             if(self.decode_process_usb1B is not None):
1638                 stream_decode_usb1B = self.decode_process_usb1B.GetInputStream()
1639                 if(stream_decode_usb1B.CanRead()):
1640                     text_decode_usb1B = stream_decode_usb1B.read()
1641                     self.display1B(text_decode_usb1B)
1642 
1643             #decoding virtual nodes
1644             if(self.decode_process_virt1 is not None):
1645                 stream_decode_virt1 = self.decode_process_virt1.GetInputStream()
1646                 if(stream_decode_virt1.CanRead()):
1647                     text_decode_virt1 = stream_decode_virt1.read()
1648                     self.olsr_v1_output+=text_decode_virt1
1649             if(self.decode_process_virt2 is not None):
1650                 stream_decode_virt2 = self.decode_process_virt2.GetInputStream()
1651                 if(stream_decode_virt2.CanRead()):
1652                     text_decode_virt2 = stream_decode_virt2.read()
1653                     self.olsr_v2_output+=text_decode_virt2
1654             if(self.decode_process_virt3 is not None):
1655                 stream_decode_virt3 = self.decode_process_virt3.GetInputStream()
1656                 if(stream_decode_virt3.CanRead()):
1657                     text_decode_virt3 = stream_decode_virt3.read()
1658                     self.olsr_v3_output+=text_decode_virt3
1659             if(self.decode_process_virt4 is not None):
1660                 stream_decode_virt4 = self.decode_process_virt4.GetInputStream()
1661                 if(stream_decode_virt4.CanRead()):
1662                     text_decode_virt4 = stream_decode_virt4.read()
1663                     self.olsr_v4_output+=text_decode_virt4
1664             if(self.decode_process_virt5 is not None):
1665                 stream_decode_virt5 = self.decode_process_virt5.GetInputStream()
1666                 if(stream_decode_virt5.CanRead()):
1667                     text_decode_virt5 = stream_decode_virt5.read()
1668                     self.olsr_v5_output+=text_decode_virt5
1669             if(self.decode_process_virt6 is not None):
1670                 stream_decode_virt6 = self.decode_process_virt6.GetInputStream()
1671                 if(stream_decode_virt6.CanRead()):
1672                     text_decode_virt6 = stream_decode_virt6.read()
1673                     self.olsr_v6_output+=text_decode_virt6
```

```
1674
1675
1676            #bbc-olsr agent processes
1677            if(self.olsr_process_usb0A is not None):
1678                stream_olsr_usb0A = self.olsr_process_usb0A.GetInputStream()
1679                if(stream_olsr_usb0A.CanRead()):
1680                    text_olsr_usb0A = stream_olsr_usb0A.read()
1681                    if(re.search("received",text_olsr_usb0A)):  #parse feedback
1682                        node_output = text_olsr_usb0A.partition(",")[0]
1683                        bytes = re.split(" ", node_output)[0]
1684                        ip = re.split(" ", node_output)[3]
1685                        addr = self.nat_table.get(ip)
1686                        #self.display0A(node_output)  #keep only necessary output
1687                        self.display0A("Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")")
1688                        self.display0A("\n")
1689            if(self.olsr_process_usb0B is not None):
1690                stream_olsr_usb0B = self.olsr_process_usb0B.GetInputStream()
1691                if(stream_olsr_usb0B.CanRead()):
1692                    text_olsr_usb0B = stream_olsr_usb0B.read()
1693                    if(re.search("received",text_olsr_usb0B)):  #parse feedback
1694                        node_output = text_olsr_usb0B.partition(",")[0]
1695                        bytes = re.split(" ", node_output)[0]
1696                        ip = re.split(" ", node_output)[3]
1697                        addr = self.nat_table.get(ip)
1698                        #self.display0A(node_output)  #keep only necessary output
1699                        self.display0B("Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")")
1700                        self.display0B("\n")
1701            if(self.olsr_process_usb1A is not None):
1702                stream_olsr_usb1A = self.olsr_process_usb1A.GetInputStream()
1703                if(stream_olsr_usb1A.CanRead()):
1704                    text_olsr_usb1A = stream_olsr_usb1A.read()
1705                    if(re.search("received",text_olsr_usb1A)):  #parse feedback
1706                        node_output = text_olsr_usb1A.partition(",")[0]
1707                        bytes = re.split(" ", node_output)[0]
1708                        ip = re.split(" ", node_output)[3]
1709                        addr = self.nat_table.get(ip)
1710                        #self.display0A(node_output)  #keep only necessary output
1711                        self.display1A("Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")")
1712                        self.display1A("\n")
1713            if(self.olsr_process_usb1B is not None):
1714                stream_olsr_usb1B = self.olsr_process_usb1B.GetInputStream()
1715                if(stream_olsr_usb1B.CanRead()):
1716                    text_olsr_usb1B = stream_olsr_usb1B.read()
1717                    if(re.search("received",text_olsr_usb1B)):  #parse feedback
1718                        node_output = text_olsr_usb1B.partition(",")[0]
```

```
1719                    bytes = re.split(" ", node_output)[0]
1720                    ip = re.split(" ", node_output)[3]
1721                    addr = self.nat_table.get(ip)
1722                    #self.display0A(node_output)  #keep only necessary output
1723                    self.display1B("Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")")
1724                    self.display1B("\n")
1725
1726            #bbc-olsr virtual nodes
1727            if(self.olsr_process_id_v1 is not None):
1728                stream_olsr_v1 = self.olsr_process_virt1.GetInputStream()
1729                if(stream_olsr_v1.CanRead()):
1730                    text_olsr_v1 = stream_olsr_v1.read()
1731                    if(re.search("received",text_olsr_v1)):  #parse feedback
1732                        node_output = text_olsr_v1.partition(",")[0]
1733                        bytes = re.split(" ", node_output)[0]
1734                        ip = re.split(" ", node_output)[3]
1735                        addr = self.nat_table.get(ip)
1736                        #self.display0A(node_output)  #keep only necessary output
1737                        self.olsr_v1_output+="Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")
                                "
1738                        self.olsr_v1_output+="\n"
1739
1740
1741            if(self.olsr_process_id_v2 is not None):
1742                stream_olsr_v2 = self.olsr_process_virt2.GetInputStream()
1743                if(stream_olsr_v2.CanRead()):
1744                    text_olsr_v2 = stream_olsr_v2.read()
1745                    if(re.search("received",text_olsr_v2)):  #parse feedback
1746                        node_output = text_olsr_v2.partition(",")[0]
1747                        bytes = re.split(" ", node_output)[0]
1748                        ip = re.split(" ", node_output)[3]
1749                        addr = self.nat_table.get(ip)
1750                        #self.display0A(node_output)  #keep only necessary output
1751                        self.olsr_v2_output+="Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")
                                "
1752                        self.olsr_v2_output+="\n"
1753
1754            if(self.olsr_process_id_v3 is not None):
1755                stream_olsr_v3 = self.olsr_process_virt3.GetInputStream()
1756                if(stream_olsr_v3.CanRead()):
1757                    text_olsr_v3 = stream_olsr_v3.read()
1758                    if(re.search("received",text_olsr_v3)):  #parse feedback
1759                        node_output = text_olsr_v3.partition(",")[0]
1760                        bytes = re.split(" ", node_output)[0]
1761                        ip = re.split(" ", node_output)[3]
```

```
1762                    addr = self.nat_table.get(ip)
1763                    #self.display0A(node_output)  #keep only necessary output
1764                    self.olsr_v3_output+="Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")
                            "
1765                    self.olsr_v3_output+="\n"
1766
1767            if(self.olsr_process_id_v4 is not None):
1768                stream_olsr_v4 = self.olsr_process_virt4.GetInputStream()
1769                if(stream_olsr_v4.CanRead()):
1770                    text_olsr_v4 = stream_olsr_v4.read()
1771                    if(re.search("received",text_olsr_v4)):  #parse feedback
1772                        node_output = text_olsr_v4.partition(",")[0]
1773                        bytes = re.split(" ", node_output)[0]
1774                        ip = re.split(" ", node_output)[3]
1775                        addr = self.nat_table.get(ip)
1776                        #self.display0A(node_output)  #keep only necessary output
1777                        self.olsr_v4_output+="Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")
                                "
1778                        self.olsr_v4_output+="\n"
1779
1780            if(self.olsr_process_id_v5 is not None):
1781                stream_olsr_v5 = self.olsr_process_virt5.GetInputStream()
1782                if(stream_olsr_v5.CanRead()):
1783                    text_olsr_v5 = stream_olsr_v5.read()
1784                    if(re.search("received",text_olsr_v5)):  #parse feedback
1785                        node_output = text_olsr_v5.partition(",")[0]
1786                        bytes = re.split(" ", node_output)[0]
1787                        ip = re.split(" ", node_output)[3]
1788                        addr = self.nat_table.get(ip)
1789                        #self.display0A(node_output)  #keep only necessary output
1790                        self.olsr_v5_output+="Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")
                                "
1791                        self.olsr_v5_output+="\n"
1792
1793            if(self.olsr_process_id_v6 is not None):
1794                stream_olsr_v6 = self.olsr_process_virt6.GetInputStream()
1795                if(stream_olsr_v6.CanRead()):
1796                    text_olsr_v6 = stream_olsr_v6.read()
1797                    if(re.search("received",text_olsr_v6)):  #parse feedback
1798                        node_output = text_olsr_v6.partition(",")[0]
1799                        bytes = re.split(" ", node_output)[0]
1800                        ip = re.split(" ", node_output)[3]
1801                        addr = self.nat_table.get(ip)
1802                        #self.display0A(node_output)  #keep only necessary output
```

```
1803                    self.olsr_v6_output+="Received packet ("+bytes+" bytes) from "+addr+" ("+ip+")
                            "
1804                    self.olsr_v6_output+="\n"
1805
1806        def doWait(self,time):
1807            wait_process = wx.Process(self, ID_WAIT_PROCESS)
1808            wait_process.Redirect()
1809            cmd = "sleep "+time
1810            wait_process_id = wx.Execute(cmd,wx.EXEC_ASYNC, wait_process)
1811
1812        def OnProcessEnded(self, event):
1813
1814            if(event.GetId() == ID_WAIT_PROCESS):
1815                self.CheckTxQueue()
1816
1817            #TX/RX processes
1818            if(event.GetId() == ID_PROCESS_1):
1819                if(self.kill_0A == False):
1820                    stream_usb0A = self.process_usb0A.GetInputStream()
1821                    if(stream_usb0A.CanRead()):
1822                        text = stream_usb0A.read()
1823                        self.display0A(text)
1824                    if(self.FindWindowById(ID_TXRX_USB0A).GetStringSelection()=="RX"):
1825                        self.decodeRxData(ID_PROCESS_1)
1826                        #need to decode for the stopped process as well
1827                        if(self.dual_rx_0 == True):
1828                            self.decodeRxData(ID_PROCESS_2)
1829                self.process_usb0A.Destroy()
1830                self.process_usb0A = None
1831                self.kill_0A = False
1832                self.doWait("120")
1833
1834            if(event.GetId() == ID_PROCESS_2):
1835                if (self.kill_0B == False):
1836                    stream_usb0B = self.process_usb0B.GetInputStream()
1837                    if(stream_usb0B.CanRead()):
1838                        text = stream_usb0B.read()
1839                        self.display0B(text)
1840                    if(self.FindWindowById(ID_TXRX_USB0B).GetStringSelection()=="RX"):
1841                        self.decodeRxData(ID_PROCESS_2)
1842                        #need to decode for the stopped process as well
1843                        if(self.dual_rx_0 == True):
1844                            self.decodeRxData(ID_PROCESS_1)
1845                self.process_usb0B.Destroy()
1846                self.process_usb0B = None
```

```
1847                self.kill_0B = False
1848                self.doWait("120")
1849
1850            if(event.GetId() == ID_PROCESS_3):
1851                if (self.kill_1A == False):
1852                    stream_usb1A = self.process_usb1A.GetInputStream()
1853                    if(stream_usb1A.CanRead()):
1854                        text = stream_usb1A.read()
1855                        self.display1A(text)
1856                    if(self.FindWindowById(ID_TXRX_USB1A).GetStringSelection()=="RX"):
1857                        self.decodeRxData(ID_PROCESS_3)
1858                        if(self.dual_rx_1==True):
1859                            self.decodeRxData(ID_PROCESS_4)
1860                self.process_usb1A.Destroy()
1861                self.process_usb1A = None
1862                self.kill_1A = False
1863                self.doWait("120")
1864
1865            if(event.GetId() == ID_PROCESS_4):
1866                if (self.kill_1B == False):
1867                    stream_usb1B = self.process_usb1B.GetInputStream()
1868                    if(stream_usb1B.CanRead()):
1869                        text = stream_usb1B.read()
1870                        self.display1B(text)
1871                    if(self.FindWindowById(ID_TXRX_USB1B).GetStringSelection()=="RX"):
1872                        self.decodeRxData(ID_PROCESS_4)
1873                        if(self.dual_rx_1==True):
1874                            self.decodeRxData(ID_PROCESS_3)
1875                self.process_usb1B.Destroy()
1876                self.process_usb1B = None
1877                self.kill_1B = False
1878                self.doWait("120")
1879
1880    #          if(event.GetId() == ID_PROCESS_5):
1881    #              if(self.kill_virt1 == False):
1882    #                  stream_virt1 = self.process_virt1.GetInputStream()
1883    #                  if(stream_virt1.CanRead()):
1884    #                      text = stream_virt1.read()
1885    #                      self.olsr_v1_output+=text
1886    #                  if(self.FindWindowById(ID_TXRX_USB0A).GetStringSelection()=="RX"):
1887    #                      self.decodeRxData(ID_PROCESS_5)
1888    #              self.process_virt1.Destroy()
1889    #              self.process_virt1 = None
1890    #              self.kill_virt1 = False
1891    #              self.doWait("120")
```

```
1892
1893
1894            #encoding
1895            if(event.GetId() == ID_PROCESS_ENC_1):
1896                stream_encode_usb0A = self.encode_process_usb0A.GetInputStream()
1897                if(stream_encode_usb0A.CanRead()):
1898                    text_encode_usb0A = stream_encode_usb0A.read()
1899                    self.display0A(text_encode_usb0A)
1900                    self.display0A(">Encoding Complete.\n")
1901                    tx_packet=self.makePacket("usrp0A")
1902                    #add to TX ready queue
1903                    self.display0A(">Adding outgoing packet to TX queue.\n")
1904                    self.tx_pkt_queue_usrp0A.append(tx_packet)
1905                    if(tx_packet.IsAckPacket()==True):
1906                        self.slideTxWindow("usrp0A", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
1907                        self.display0A(">Moving TX'd ACK to TX History queue\n")
1908                self.encode_process_usb0A.Destroy()
1909                self.encode_process_usb0A = None
1910            if(event.GetId() == ID_PROCESS_ENC_2):
1911                stream_encode_usb0B = self.encode_process_usb0B.GetInputStream()
1912                if(stream_encode_usb0B.CanRead()):
1913                    text_encode_usb0B = stream_encode_usb0B.read()
1914                    self.display0B(text_encode_usb0B)
1915                    self.display0B(">Encoding Complete.\n")
1916                    tx_packet=self.makePacket("usrp0B")
1917                    #add to TX ready queue
1918                    self.display0B(">Adding outgoing packet to TX queue.\n")
1919                    self.tx_pkt_queue_usrp0B.append(tx_packet)
1920                    if(tx_packet.IsAckPacket()==True):
1921                        self.slideTxWindow("usrp0B", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
1922                        self.display0B(">Moving TX'd ACK to TX History queue\n")
1923                self.encode_process_usb0B.Destroy()
1924                self.encode_process_usb0B = None
1925            if(event.GetId() == ID_PROCESS_ENC_3):
1926                stream_encode_usb1A = self.encode_process_usb1A.GetInputStream()
1927                if(stream_encode_usb1A.CanRead()):
1928                    text_encode_usb1A = stream_encode_usb1A.read()
1929                    self.display1A(text_encode_usb1A)
1930                    self.display1A(">Encoding Complete.\n")
1931                    tx_packet=self.makePacket("usrp1A")
1932                    #add to TX ready queue
1933                    self.display1A(">Adding outgoing packet to TX queue.\n")
1934                    self.tx_pkt_queue_usrp1A.append(tx_packet)
```

```
1935                    if(tx_packet.IsAckPacket()==True):
1936                        self.slideTxWindow("usrp1A", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
1937                        self.display1A(">Moving TX'd ACK to TX History queue\n")
1938                self.encode_process_usb1A.Destroy()
1939                self.encode_process_usb1A = None
1940            if(event.GetId() == ID_PROCESS_ENC_4):
1941                stream_encode_usb1B = self.encode_process_usb1B.GetInputStream()
1942                if(stream_encode_usb1B.CanRead()):
1943                    text_encode_usb1B = stream_encode_usb1B.read()
1944                    self.display1B(text_encode_usb1B)
1945                    self.display1B(">Encoding Complete.\n")
1946                    tx_packet=self.makePacket("usrp1B")
1947                    #tx_packet.print_pkt()
1948                    self.display1B(">Adding outgoing packet to TX queue.\n")
1949                    self.tx_pkt_queue_usrp1B.append(tx_packet)
1950                    if(tx_packet.IsAckPacket()==True):
1951                        self.slideTxWindow("usrp1B", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
1952                        self.display1B(">Moving TX'd ACK to TX History queue\n")
1953                self.encode_process_usb1B.Destroy()
1954                self.encode_process_usb1B = None
1955
1956            if(event.GetId() == ID_PROCESS_ENC_5):
1957                stream_encode_virt1 = self.encode_process_virt1.GetInputStream()
1958                if(stream_encode_virt1.CanRead()):
1959                    text_encode_virt1 = stream_encode_virt1.read()
1960                    self.olsr_v1_output+=text_encode_virt1
1961                    self.olsr_v1_output+=">Encoding Complete.\n"
1962                    tx_packet=self.makePacket("virt1")
1963                    #tx_packet.print_pkt()
1964                    self.olsr_v1_output+=">Adding outgoing packet to TX queue.\n"
1965                    self.tx_pkt_queue_virt1.append(tx_packet)
1966                    if(tx_packet.IsAckPacket()==True):
1967                        self.slideTxWindow("virt1", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
1968                        self.olsr_v1_output+=">Moving TX'd ACK to TX History queue\n"
1969                self.encode_process_virt1.Destroy()
1970                self.encode_process_virt1 = None
1971            if(event.GetId() == ID_PROCESS_ENC_6):
1972                stream_encode_virt2 = self.encode_process_virt2.GetInputStream()
1973                if(stream_encode_virt2.CanRead()):
1974                    text_encode_virt2 = stream_encode_virt2.read()
1975                    self.olsr_v2_output+=text_encode_virt2
1976                    self.olsr_v2_output+=">Encoding Complete.\n"
```

```
1977                    tx_packet=self.makePacket("virt2")
1978                    #tx_packet.print_pkt()
1979                    self.olsr_v2_output+=">Adding outgoing packet to TX queue.\n"
1980                    self.tx_pkt_queue_virt2.append(tx_packet)
1981                    if(tx_packet.IsAckPacket()==True):
1982                        self.slideTxWindow("virt2", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
1983                        self.olsr_v2_output+=">Moving TX'd ACK to TX History queue\n"
1984                self.encode_process_virt2.Destroy()
1985                self.encode_process_virt2 = None
1986            if(event.GetId() == ID_PROCESS_ENC_7):
1987                stream_encode_virt3 = self.encode_process_virt3.GetInputStream()
1988                if(stream_encode_virt3.CanRead()):
1989                    text_encode_virt3 = stream_encode_virt3.read()
1990                    self.olsr_v3_output+=text_encode_virt3
1991                    self.olsr_v3_output+=">Encoding Complete.\n"
1992                    tx_packet=self.makePacket("virt3")
1993                    #tx_packet.print_pkt()
1994                    self.olsr_v3_output+=">Adding outgoing packet to TX queue.\n"
1995                    self.tx_pkt_queue_virt3.append(tx_packet)
1996                    if(tx_packet.IsAckPacket()==True):
1997                        self.slideTxWindow("virt3", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
1998                        self.olsr_v3_output+=">Moving TX'd ACK to TX History queue\n"
1999                self.encode_process_virt3.Destroy()
2000                self.encode_process_virt3 = None
2001            if(event.GetId() == ID_PROCESS_ENC_8):
2002                stream_encode_virt4 = self.encode_process_virt4.GetInputStream()
2003                if(stream_encode_virt4.CanRead()):
2004                    text_encode_virt4 = stream_encode_virt4.read()
2005                    self.olsr_v4_output+=text_encode_virt4
2006                    self.olsr_v4_output+=">Encoding Complete.\n"
2007                    tx_packet=self.makePacket("virt4")
2008                    #tx_packet.print_pkt()
2009                    self.olsr_v4_output+=">Adding outgoing packet to TX queue.\n"
2010                    self.tx_pkt_queue_virt4.append(tx_packet)
2011                    if(tx_packet.IsAckPacket()==True):
2012                        self.slideTxWindow("virt4", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
2013                        self.olsr_v4_output+=">Moving TX'd ACK to TX History queue\n"
2014                self.encode_process_virt4.Destroy()
2015                self.encode_process_virt4 = None
2016            if(event.GetId() == ID_PROCESS_ENC_9):
2017                stream_encode_virt5 = self.encode_process_virt5.GetInputStream()
2018                if(stream_encode_virt5.CanRead()):
```

```
2019                    text_encode_virt5 = stream_encode_virt5.read()
2020                    self.olsr_v5_output+=text_encode_virt5
2021                    self.olsr_v5_output+=">Encoding Complete.\n"
2022                    tx_packet=self.makePacket("virt5")
2023                    #tx_packet.print_pkt()
2024                    self.olsr_v5_output+=">Adding outgoing packet to TX queue.\n"
2025                    self.tx_pkt_queue_virt5.append(tx_packet)
2026                    if(tx_packet.IsAckPacket()==True):
2027                        self.slideTxWindow("virt5", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
2028                        self.olsr_v5_output+=">Moving TX'd ACK to TX History queue\n"
2029                self.encode_process_virt5.Destroy()
2030                self.encode_process_virt5 = None
2031            if(event.GetId() == ID_PROCESS_ENC_10):
2032                stream_encode_virt6 = self.encode_process_virt6.GetInputStream()
2033                if(stream_encode_virt6.CanRead()):
2034                    text_encode_virt6 = stream_encode_virt6.read()
2035                    self.olsr_v6_output+=text_encode_virt6
2036                    self.olsr_v6_output+=">Encoding Complete.\n"
2037                    tx_packet=self.makePacket("virt6")
2038                    #tx_packet.print_pkt()
2039                    self.olsr_v6_output+=">Adding outgoing packet to TX queue.\n"
2040                    self.tx_pkt_queue_virt6.append(tx_packet)
2041                    if(tx_packet.IsAckPacket()==True):
2042                        self.slideTxWindow("virt6", tx_packet)  #move newly transmitted ACK packet
                                from TX to TX History
2043                        self.olsr_v6_output+=">Moving TX'd ACK to TX History queue\n"
2044                self.encode_process_virt6.Destroy()
2045                self.encode_process_virt6 = None
2046
2047            #decoding
2048            if(event.GetId() == ID_PROCESS_DEC_1):
2049                stream_decode_usb0A = self.decode_process_usb0A.GetInputStream()
2050                if(stream_decode_usb0A.CanRead()):
2051                    text_decode_usb0A = stream_decode_usb0A.read()
2052                    self.display0A(text_decode_usb0A)
2053                    self.display0A(">Decoding Complete.  Starting Analysis.\n")
2054                    self.analyzeReceivedData("usrp0A")
2055                self.decode_process_usb0A.Destroy()
2056                self.decode_process_usb0A = None
2057            if(event.GetId() == ID_PROCESS_DEC_2):
2058                stream_decode_usb0B = self.decode_process_usb0B.GetInputStream()
2059                if(stream_decode_usb0B.CanRead()):
2060                    text_decode_usb0B = stream_decode_usb0B.read()
2061                    self.display0B(text_decode_usb0B)
```

```
2062                 self.display0B(">Decoding Complete.  Starting Analysis.\n")
2063                 self.analyzeReceivedData("usrp0B")
2064             self.decode_process_usb0B.Destroy()
2065             self.decode_process_usb0B = None
2066         if(event.GetId() == ID_PROCESS_DEC_3):
2067             stream_decode_usb1A = self.decode_process_usb1A.GetInputStream()
2068             if(stream_decode_usb1A.CanRead()):
2069                 text_decode_usb1A = stream_decode_usb1A.read()
2070                 self.display1A(text_decode_usb1A)
2071                 self.display1A(">Decoding Complete.  Starting Analysis.\n")
2072                 self.analyzeReceivedData("usrp1A")
2073             self.decode_process_usb1A.Destroy()
2074             self.decode_process_usb1A = None
2075         if(event.GetId() == ID_PROCESS_DEC_4):
2076             stream_decode_usb1B = self.decode_process_usb1B.GetInputStream()
2077             if(stream_decode_usb1B.CanRead()):
2078                 text_decode_usb1B = stream_decode_usb1B.read()
2079                 self.display1B(text_decode_usb1B)
2080                 self.display1B(">Decoding Complete.  Starting Analysis.\n")
2081                 #print text_decode_usb1B
2082                 self.analyzeReceivedData("usrp1B")
2083             self.decode_process_usb1B.Destroy()
2084             self.decode_process_usb1B = None
2085         if(event.GetId() == ID_PROCESS_DEC_5):
2086             stream_decode_virt1 = self.decode_process_virt1.GetInputStream()
2087             if(stream_decode_virt1.CanRead()):
2088                 text_decode_virt1 = stream_decode_virt1.read()
2089                 self.olsr_v1_output+=text_decode_virt1
2090                 self.olsr_v1_output+=">Decoding Complete.  Starting Analysis.\n"
2091                 self.analyzeReceivedData("virt1")
2092             self.decode_process_virt1.Destroy()
2093             self.decode_process_virt1 = None
2094         if(event.GetId() == ID_PROCESS_DEC_6):
2095             stream_decode_virt2 = self.decode_process_virt2.GetInputStream()
2096             if(stream_decode_virt2.CanRead()):
2097                 text_decode_virt2 = stream_decode_virt2.read()
2098                 self.olsr_v2_output+=text_decode_virt2
2099                 self.olsr_v2_output+=">Decoding Complete.  Starting Analysis.\n"
2100                 self.analyzeReceivedData("virt2")
2101             self.decode_process_virt2.Destroy()
2102             self.decode_process_virt2 = None
2103         if(event.GetId() == ID_PROCESS_DEC_7):
2104             stream_decode_virt3 = self.decode_process_virt3.GetInputStream()
2105             if(stream_decode_virt3.CanRead()):
2106                 text_decode_virt3 = stream_decode_virt3.read()
```

```
2107                 self.olsr_v3_output+=text_decode_virt3
2108                 self.olsr_v3_output+=">Decoding Complete.  Starting Analysis.\n"
2109                 self.analyzeReceivedData("virt3")
2110             self.decode_process_virt3.Destroy()
2111             self.decode_process_virt3 = None
2112         if(event.GetId() == ID_PROCESS_DEC_8):
2113             stream_decode_virt4 = self.decode_process_virt4.GetInputStream()
2114             if(stream_decode_virt4.CanRead()):
2115                 text_decode_virt4 = stream_decode_virt4.read()
2116                 self.olsr_v4_output+=text_decode_virt4
2117                 self.olsr_v4_output+=">Decoding Complete.  Starting Analysis.\n"
2118                 self.analyzeReceivedData("virt4")
2119             self.decode_process_virt4.Destroy()
2120             self.decode_process_virt4 = None
2121         if(event.GetId() == ID_PROCESS_DEC_9):
2122             stream_decode_virt5 = self.decode_process_virt5.GetInputStream()
2123             if(stream_decode_virt5.CanRead()):
2124                 text_decode_virt5 = stream_decode_virt5.read()
2125                 self.olsr_v5_output+=text_decode_virt5
2126                 self.olsr_v5_output+=">Decoding Complete.  Starting Analysis.\n"
2127                 self.analyzeReceivedData("virt5")
2128             self.decode_process_virt5.Destroy()
2129             self.decode_process_virt5 = None
2130         if(event.GetId() == ID_PROCESS_DEC_10):
2131             stream_decode_virt6 = self.decode_process_virt6.GetInputStream()
2132             if(stream_decode_virt6.CanRead()):
2133                 text_decode_virt6 = stream_decode_virt6.read()
2134                 self.olsr_v6_output+=text_decode_virt6
2135                 self.olsr_v6_output+=">Decoding Complete.  Starting Analysis.\n"
2136                 self.analyzeReceivedData("virt6")
2137             self.decode_process_virt6.Destroy()
2138             self.decode_process_virt6 = None
2139
2140
2141
2142     def makePacket(self,node_id):
2143         #packet creation function
2144         path = self.experiment_path+node_id+"/"
2145
2146         if(self.FindWindowById(self.getTxRxId(node_id)).GetStringSelection()=="TX"):
2147             filename="logtx.txt"
2148             f=open(path+filename,'r')
2149             msg_marks=f.readline().strip()
2150             msg_count = "0"
2151             pkts_lost = "0"
```

```
2152                current_node_name = node_id
2153                current_node_address = self.getNodeAddress(node_id)
2154                pkt_to_address = self.getNodeAddress(self.FindWindowById(self.getRecipientId(node_id))
                        .GetValue())
2155                pkt_from_address = self.getNodeAddress(node_id)
2156                pkt_file_name = self.FindWindowById(self.getSrcNameId(node_id)).GetValue()
2157                pkt_rssi_value = self.rssi
2158            else:
2159                filename = "logrx.txt"
2160                f=open(path+filename,'r')
2161                msg_marks = f.readline().strip()
2162                msg_count = f.readline().strip()
2163                pkts_lost = f.readline().strip()
2164                current_node_name = f.readline().strip()
2165                current_node_address = f.readline().strip()
2166                pkt_to_address = f.readline().strip()
2167                pkt_from_address = f.readline().strip()
2168                pkt_file_name = f.readline().strip()
2169                pkt_rssi_value = f.readline().strip()
2170
2171
2172            currentPacket = packet.packet(node_id=current_node_name, node_address=current_node_address
                    , to_address=pkt_to_address,
2173                                          from_address=pkt_from_address,filename=pkt_file_name,marks=msg_marks
                                                ,messages_found=msg_count,
2174                                          packets_lost=pkts_lost, rssi_value=pkt_rssi_value)
2175            #print packet for debugging
2176            #currentPacket.print_pkt()
2177
2178            return currentPacket
2179
2180        def analyzeReceivedData(self,node_id):
2181            path = self.experiment_path+node_id+"/"
2182            receivedPacket = self.makePacket(node_id)
2183
2184            if(receivedPacket.GetToAddress()==receivedPacket.GetNodeAddress()):  #accept packet
2185                #self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Accepting packet
                        addressed to "+receivedPacket.GetToAddress()+" from "+receivedPacket.
                        GetFromAddress()+".")
2186                #needs to be more efficient....but this works
2187                if(node_id == "usrp0A"):#add packet to RX queue for this node
2188                    self.rx_pkt_queue_usrp0A.append(receivedPacket)
2189                    self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Accepting packet
                            addressed to "+receivedPacket.GetToAddress()+" from "+receivedPacket.
                            GetFromAddress()+".")
```

```
2190                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Adding RX'd
                        Packet to node "+receivedPacket.GetToAddress() +"'s RX Pkt Queue.")
2191            elif(node_id == "usrp0B"):
2192                self.rx_pkt_queue_usrp0B.append(receivedPacket)
2193                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Accepting packet
                        addressed to "+receivedPacket.GetToAddress()+" from "+receivedPacket.
                        GetFromAddress()+".")
2194                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Adding RX'd
                        Packet to node "+receivedPacket.GetToAddress() +"'s RX Pkt Queue.")
2195            elif(node_id == "usrp1A"):
2196                self.rx_pkt_queue_usrp1A.append(receivedPacket)
2197                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Accepting packet
                        addressed to "+receivedPacket.GetToAddress()+" from "+receivedPacket.
                        GetFromAddress()+".")
2198                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Adding RX'd
                        Packet to node "+receivedPacket.GetToAddress() +"'s RX Pkt Queue.")
2199            elif(node_id == "usrp1B"):
2200                self.rx_pkt_queue_usrp1B.append(receivedPacket)
2201                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Accepting packet
                        addressed to "+receivedPacket.GetToAddress()+" from "+receivedPacket.
                        GetFromAddress()+".")
2202                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Adding RX'd
                        Packet to node "+receivedPacket.GetToAddress() +"'s RX Pkt Queue.")
2203            elif(node_id == "virt1"):
2204                self.rx_pkt_queue_virt1.append(receivedPacket)
2205                self.olsr_v1_output+="\n>Accepting packet addressed to "+receivedPacket.
                        GetToAddress()+" from "+receivedPacket.GetFromAddress()+"."
2206                self.olsr_v1_output+="\n>Adding RX'd Packet to node "+receivedPacket.GetToAddress
                        () +"'s RX Pkt Queue."
2207            elif(node_id == "virt2"):
2208                self.rx_pkt_queue_virt2.append(receivedPacket)
2209                self.olsr_v2_output+="\n>Accepting packet addressed to "+receivedPacket.
                        GetToAddress()+" from "+receivedPacket.GetFromAddress()+"."
2210                self.olsr_v2_output+="\n>Adding RX'd Packet to node "+receivedPacket.GetToAddress
                        () +"'s RX Pkt Queue."
2211            elif(node_id == "virt3"):
2212                self.rx_pkt_queue_virt3.append(receivedPacket)
2213                self.olsr_v3_output+="\n>Accepting packet addressed to "+receivedPacket.
                        GetToAddress()+" from "+receivedPacket.GetFromAddress()+"."
2214                self.olsr_v3_output+="\n>Adding RX'd Packet to node "+receivedPacket.GetToAddress
                        () +"'s RX Pkt Queue."
2215            elif(node_id == "virt4"):
2216                self.rx_pkt_queue_virt4.append(receivedPacket)
2217                self.olsr_v4_output+="\n>Accepting packet addressed to "+receivedPacket.
                        GetToAddress()+" from "+receivedPacket.GetFromAddress()+"."
```

```
2218                self.olsr_v4_output+="\n>Adding RX'd Packet to node "+receivedPacket.GetToAddress
                        () +"'s RX Pkt Queue."
2219            elif(node_id == "virt5"):
2220                self.rx_pkt_queue_virt5.append(receivedPacket)
2221                self.olsr_v5_output+="\n>Accepting packet addressed to "+receivedPacket.
                        GetToAddress()+" from "+receivedPacket.GetFromAddress()+"."
2222                self.olsr_v5_output+="\n>Adding RX'd Packet to node "+receivedPacket.GetToAddress
                        () +"'s RX Pkt Queue."
2223            elif(node_id == "virt6"):
2224                self.rx_pkt_queue_virt6.append(receivedPacket)
2225                self.olsr_v6_output+="\n>Accepting packet addressed to "+receivedPacket.
                        GetToAddress()+" from "+receivedPacket.GetFromAddress()+"."
2226                self.olsr_v6_output+="\n>Adding RX'd Packet to node "+receivedPacket.GetToAddress
                        () +"'s RX Pkt Queue."
2227
2228            #self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Adding RX'd Packet
                    to node "+receivedPacket.GetToAddress() +"'s RX Pkt Queue.")
2229
2230            #if this packet received is not an ACK, trigger the transmission of an ACK packet
2231            if(receivedPacket.IsAckPacket()==False):
2232                #form ack packet for the received file
2233                ack_name = "Ack"+receivedPacket.GetFilename()
2234                ack_file_name = path+ack_name
2235
2236                #create ACK packet
2237                file=open(ack_file_name,'w')
2238                file.write("ACK"+receivedPacket.GetFilename()+"\n")
2239                file.close()
2240
2241                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Sending ACK: "+
                        ack_file_name+" from "+receivedPacket.node_id+"("+receivedPacket.
                        GetNodeAddress()+") to "+receivedPacket.GetFromAddress() + ".")
2242                self.transmitToFrom(to_radio=receivedPacket.GetFromAddress(), from_radio=
                        receivedPacket.GetNodeAddress(), message=ack_name)
2243                #transmission of ACK is treated as "fire-and-forget" activity since
2244                #  if a transmitter doesn't receive the ACK, it will retransmit anyway, so slide
                        the RX window
2245                self.slideRxWindow(node_id, receivedPacket)  #move received packet from RX to RX
                        History after sending ACK
2246
2247
2248            else: #packet is an ACK
2249                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Received ACK ("+
                        receivedPacket.GetFilename()+") from "+receivedPacket.GetFromAddress()+".")
```

```
2250                    self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Removing ACK'd
                            Packet from "+receivedPacket.GetToAddress() +"'s TX Pkt Queue.")
2251                    #move ACK'd packet from TX queue to TX history to keep record of successes
2252                    self.slideTxWindow(node_id, receivedPacket)
2253                    #add ACK packet to RX history queue for record keeping purposes
2254                    self.slideRxWindow(node_id, receivedPacket)
2255            else:
2256                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>Rejecting packet
                        addressed to "+receivedPacket.GetToAddress()+" from "+receivedPacket.
                        GetFromAddress()+".")
2257
2258
2259        #setup a queue for each node, and if you transmit a response, move to a historical data queue
2260        #instead of keeping the information in the ready-for-action queue
2261        def slideTxWindow(self,node_name,packet):
2262            if(node_name=="usrp0A"):
2263                self.tx_pkt_queue_historical_usrp0A.append(self.tx_pkt_queue_usrp0A.pop(self.
                            getTxIndexInList(node_name, packet)))
2264            elif(node_name=="usrp0B"):
2265                self.tx_pkt_queue_historical_usrp0B.append(self.tx_pkt_queue_usrp0B.pop(self.
                            getTxIndexInList(node_name, packet)))
2266            elif(node_name=="usrp1A"):
2267                self.tx_pkt_queue_historical_usrp1A.append(self.tx_pkt_queue_usrp1A.pop(self.
                            getTxIndexInList(node_name, packet)))
2268            elif(node_name=="usrp1B"):
2269                self.tx_pkt_queue_historical_usrp1B.append(self.tx_pkt_queue_usrp1B.pop(self.
                            getTxIndexInList(node_name, packet)))
2270            elif(node_name=="virt1"):
2271                self.tx_pkt_queue_historical_virt1.append(self.tx_pkt_queue_virt1.pop(self.
                            getTxIndexInList(node_name, packet)))
2272            elif(node_name=="virt2"):
2273                self.tx_pkt_queue_historical_virt2.append(self.tx_pkt_queue_virt2.pop(self.
                            getTxIndexInList(node_name, packet)))
2274            elif(node_name=="virt3"):
2275                self.tx_pkt_queue_historical_virt3.append(self.tx_pkt_queue_virt3.pop(self.
                            getTxIndexInList(node_name, packet)))
2276            elif(node_name=="virt4"):
2277                self.tx_pkt_queue_historical_virt4.append(self.tx_pkt_queue_virt4.pop(self.
                            getTxIndexInList(node_name, packet)))
2278            elif(node_name=="virt5"):
2279                self.tx_pkt_queue_historical_virt5.append(self.tx_pkt_queue_virt5.pop(self.
                            getTxIndexInList(node_name, packet)))
2280            elif(node_name=="virt6"):
2281                self.tx_pkt_queue_historical_virt6.append(self.tx_pkt_queue_virt6.pop(self.
                            getTxIndexInList(node_name, packet)))
```

```
2282
2283        def slideRxWindow(self,node_name,packet):
2284            if(node_name=="usrp0A"):
2285                self.rx_pkt_queue_historical_usrp0A.append(self.rx_pkt_queue_usrp0A.pop(self.
                        getRxIndexInList(node_name, packet)))
2286            elif(node_name=="usrp0B"):
2287                self.rx_pkt_queue_historical_usrp0B.append(self.rx_pkt_queue_usrp0B.pop(self.
                        getRxIndexInList(node_name, packet)))
2288            elif(node_name=="usrp1A"):
2289                self.rx_pkt_queue_historical_usrp1A.append(self.rx_pkt_queue_usrp1A.pop(self.
                        getRxIndexInList(node_name, packet)))
2290            elif(node_name=="usrp1B"):
2291                self.rx_pkt_queue_historical_usrp1B.append(self.rx_pkt_queue_usrp1B.pop(self.
                        getRxIndexInList(node_name, packet)))
2292            elif(node_name=="virt1"):
2293                self.rx_pkt_queue_historical_virt1.append(self.rx_pkt_queue_virt1.pop(self.
                        getRxIndexInList(node_name, packet)))
2294            elif(node_name=="virt2"):
2295                self.rx_pkt_queue_historical_virt2.append(self.rx_pkt_queue_virt2.pop(self.
                        getRxIndexInList(node_name, packet)))
2296            elif(node_name=="virt3"):
2297                self.rx_pkt_queue_historical_virt3.append(self.rx_pkt_queue_virt3.pop(self.
                        getRxIndexInList(node_name, packet)))
2298            elif(node_name=="virt4"):
2299                self.rx_pkt_queue_historical_virt4.append(self.rx_pkt_queue_virt4.pop(self.
                        getRxIndexInList(node_name, packet)))
2300            elif(node_name=="virt5"):
2301                self.rx_pkt_queue_historical_virt5.append(self.rx_pkt_queue_virt5.pop(self.
                        getRxIndexInList(node_name, packet)))
2302            elif(node_name=="virt6"):
2303                self.rx_pkt_queue_historical_virt6.append(self.rx_pkt_queue_virt6.pop(self.
                        getRxIndexInList(node_name, packet)))
2304
2305
2306        def getTxIndexInList(self,node_name,packet):
2307            if(node_name=="usrp0A"):
2308                for i in self.tx_pkt_queue_usrp0A:
2309                    if(packet.GetFilename().find(i.GetFilename())>=0):
2310                        return self.tx_pkt_queue_usrp0A.index(i)
2311            elif(node_name=="usrp0B"):
2312                for i in self.tx_pkt_queue_usrp0B:
2313                    if(packet.GetFilename().find(i.GetFilename())>=0):
2314                        return self.tx_pkt_queue_usrp0B.index(i)
2315            elif(node_name=="usrp1A"):
2316                for i in self.tx_pkt_queue_usrp1A:
```

```
2317                 if(packet.GetFilename().find(i.GetFilename())>=0):
2318                     return self.tx_pkt_queue_usrp1A.index(i)
2319         elif(node_name=="usrp1B"):
2320             for i in self.tx_pkt_queue_usrp1B:
2321                 if(packet.GetFilename().find(i.GetFilename())>=0):
2322                     return self.tx_pkt_queue_usrp1B.index(i)
2323         elif(node_name=="virt1"):
2324             for i in self.tx_pkt_queue_virt1:
2325                 if(packet.GetFilename().find(i.GetFilename())>=0):
2326                     return self.tx_pkt_queue_virt1.index(i)
2327         elif(node_name=="virt2"):
2328             for i in self.tx_pkt_queue_virt2:
2329                 if(packet.GetFilename().find(i.GetFilename())>=0):
2330                     return self.tx_pkt_queue_virt2.index(i)
2331         elif(node_name=="virt3"):
2332             for i in self.tx_pkt_queue_virt3:
2333                 if(packet.GetFilename().find(i.GetFilename())>=0):
2334                     return self.tx_pkt_queue_virt3.index(i)
2335         elif(node_name=="virt4"):
2336             for i in self.tx_pkt_queue_virt4:
2337                 if(packet.GetFilename().find(i.GetFilename())>=0):
2338                     return self.tx_pkt_queue_virt4.index(i)
2339         elif(node_name=="virt5"):
2340             for i in self.tx_pkt_queue_virt5:
2341                 if(packet.GetFilename().find(i.GetFilename())>=0):
2342                     return self.tx_pkt_queue_virt5.index(i)
2343         elif(node_name=="virt6"):
2344             for i in self.tx_pkt_queue_virt6:
2345                 if(packet.GetFilename().find(i.GetFilename())>=0):
2346                     return self.tx_pkt_queue_virt6.index(i)
2347 
2348 
2349     def getRxIndexInList(self,node_name,packet):
2350         if(node_name=="usrp0A"):
2351             for i in self.rx_pkt_queue_usrp0A:
2352                 if(packet.GetFilename().find(i.GetFilename())>=0):
2353                     return self.rx_pkt_queue_usrp0A.index(i)
2354         elif(node_name=="usrp0B"):
2355             for i in self.rx_pkt_queue_usrp0B:
2356                 if(packet.GetFilename().find(i.GetFilename())>=0):
2357                     return self.rx_pkt_queue_usrp0B.index(i)
2358         elif(node_name=="usrp1A"):
2359             for i in self.rx_pkt_queue_usrp1A:
2360                 if(packet.GetFilename().find(i.GetFilename())>=0):
2361                     return self.rx_pkt_queue_usrp1A.index(i)
```

```
2362            elif(node_name=="usrp1B"):
2363                for i in self.rx_pkt_queue_usrp1B:
2364                    if(packet.GetFilename().find(i.GetFilename())>=0):
2365                        return self.rx_pkt_queue_usrp1B.index(i)
2366            elif(node_name=="virt1"):
2367                for i in self.rx_pkt_queue_virt1:
2368                    if(packet.GetFilename().find(i.GetFilename())>=0):
2369                        return self.rx_pkt_queue_virt1.index(i)
2370            elif(node_name=="virt2"):
2371                for i in self.rx_pkt_queue_virt2:
2372                    if(packet.GetFilename().find(i.GetFilename())>=0):
2373                        return self.rx_pkt_queue_virt2.index(i)
2374            elif(node_name=="virt3"):
2375                for i in self.rx_pkt_queue_virt3:
2376                    if(packet.GetFilename().find(i.GetFilename())>=0):
2377                        return self.rx_pkt_queue_virt3.index(i)
2378            elif(node_name=="virt4"):
2379                for i in self.rx_pkt_queue_virt4:
2380                    if(packet.GetFilename().find(i.GetFilename())>=0):
2381                        return self.rx_pkt_queue_virt4.index(i)
2382            elif(node_name=="virt5"):
2383                for i in self.rx_pkt_queue_virt5:
2384                    if(packet.GetFilename().find(i.GetFilename())>=0):
2385                        return self.rx_pkt_queue_virt5.index(i)
2386            elif(node_name=="virt6"):
2387                for i in self.rx_pkt_queue_virt6:
2388                    if(packet.GetFilename().find(i.GetFilename())>=0):
2389                        return self.rx_pkt_queue_virt6.index(i)
2390
2391
2392        def transmitToFrom(self, to_radio, from_radio, message):
2393            #print "Inside transmitToFrom()"
2394
2395            to_radio_name = self.getNodeName(to_radio)
2396            from_radio_name = self.getNodeName(from_radio)
2397            message_name = message    #path included in this name
2398
2399            #change parameters for "to" radio so that it is in RX mode
2400            if(to_radio_name != "All"):
2401                id_radio_box = self.getTxRxId(to_radio_name)
2402                self.toggleRadioBoxSelection(id_radio_box)
2403                self.SendEvent(mycontrol=self.TxOrRx, event=wx.EVT_RADIOBOX, id=id_radio_box) #calls
                        radio button event
2404                self.SendEvent(mycontrol=self.updateButton, event=wx.EVT_BUTTON, id=self.
                        getUpateButtonId(to_radio_name))#calls update button event
```

```
2405            #else:
2406                #handle event where all radios must be updated and put in RX mode
2407
2408            #change parameters for "from" radio so that it is in TX mode
2409            id_radio_box = self.getTxRxId(from_radio_name)
2410            self.toggleRadioBoxSelection(id_radio_box)
2411            self.SendEvent(mycontrol=self.TxOrRx, event=wx.EVT_RADIOBOX, id=id_radio_box) #calls radio
                    button event
2412            self.FindWindowById(self.getSrcNameId(from_radio_name)).SetValue(message_name)
2413            self.FindWindowById(self.getRecipientId(from_radio_name)).SetValue(to_radio_name)
2414            self.SendEvent(mycontrol=self.updateButton, event=wx.EVT_BUTTON, id=self.getUpateButtonId(
                    from_radio_name))#calls update button event
2415
2416            #start RX
2417            self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.getStartButtonId(
                    to_radio_name))
2418            #start TX
2419            self.SendEvent(mycontrol=self.startButton, event=wx.EVT_BUTTON, id=self.getStartButtonId(
                    from_radio_name))
2420
2421        def SendEvent (self, mycontrol, event, id):
2422            cmd = wx.CommandEvent(event.evtType[0])
2423            cmd.SetEventObject(mycontrol)
2424            cmd.SetId(id)
2425            mycontrol.GetEventHandler().ProcessEvent(cmd)
2426
2427        def display0A(self,some_text):
2428            self.FindWindowById(ID_LOGWIN_USB0A).AppendText(some_text)
2429
2430        def display0B(self,some_text):
2431            self.FindWindowById(ID_LOGWIN_USB0B).AppendText(some_text)
2432
2433        def display1A(self,some_text):
2434            self.FindWindowById(ID_LOGWIN_USB1A).AppendText(some_text)
2435
2436        def display1B(self,some_text):
2437            self.FindWindowById(ID_LOGWIN_USB1B).AppendText(some_text)
2438
2439        def getLogWindowId(self,node_name):
2440            if(node_name=="usrp0A"):
2441                return ID_LOGWIN_USB0A
2442            elif(node_name=="usrp0B"):
2443                return ID_LOGWIN_USB0B
2444            elif(node_name=="usrp1A"):
2445                return ID_LOGWIN_USB1A
```

```
2446            elif(node_name=="usrp1B"):
2447                return ID_LOGWIN_USB1B
2448
2449        def getNodeName(self, address):
2450            if(address==self.nat_table.items()[0][0]):
2451                return "usrp0A"
2452            elif(address==self.nat_table.items()[1][0]):
2453                return "usrp0B"
2454            elif(address==self.nat_table.items()[2][0]):
2455                return "usrp1A"
2456            elif(address==self.nat_table.items()[3][0]):
2457                return "usrp1B"
2458            elif(address==self.nat_table.items()[4][0]):
2459                return "virt1"
2460            elif(address==self.nat_table.items()[5][0]):
2461                return "virt2"
2462            elif(address==self.nat_table.items()[6][0]):
2463                return "virt3"
2464            elif(address==self.nat_table.items()[7][0]):
2465                return "virt4"
2466            elif(address==self.nat_table.items()[8][0]):
2467                return "virt5"
2468            elif(address==self.nat_table.items()[9][0]):
2469                return "virt6"
2470            elif(address=="888"):
2471                return "All"
2472            else: return "NO_NAME_MATCH"
2473
2474        def getNodeAddress(self, node_name):
2475            if(node_name=="usrp0A"):
2476                return self.nat_table.items()[0][0]
2477            elif(node_name=="usrp0B"):
2478                return self.nat_table.items()[1][0]
2479            elif(node_name=="usrp1A"):
2480                return self.nat_table.items()[2][0]
2481            elif(node_name=="usrp1B"):
2482                return self.nat_table.items()[3][0]
2483            elif(node_name=="virt1"):
2484                return self.nat_table.items()[4][0]
2485            elif(node_name=="virt2"):
2486                return self.nat_table.items()[5][0]
2487            elif(node_name=="virt3"):
2488                return self.nat_table.items()[6][0]
2489            elif(node_name=="virt4"):
2490                return self.nat_table.items()[7][0]
```

```
2491            elif(node_name=="virt5"):
2492                return self.nat_table.items()[8][0]
2493            elif(node_name=="virt6"):
2494                return self.nat_table.items()[9][0]
2495            elif(node_name=="All"):
2496                return "888"
2497            else: return "NO_ADDR_MATCH"
2498
2499        def getTxRxId(self,node_name):
2500            if(node_name=="usrp0A"):
2501                return ID_TXRX_USB0A
2502            elif(node_name=="usrp0B"):
2503                return ID_TXRX_USB0B
2504            elif(node_name=="usrp1A"):
2505                return ID_TXRX_USB1A
2506            elif(node_name=="usrp1B"):
2507                return ID_TXRX_USB1B
2508            elif(node_name=="virt1"):
2509                return ID_TXRX_VIRT1
2510            elif(node_name=="virt2"):
2511                return ID_TXRX_VIRT2
2512            elif(node_name=="virt3"):
2513                return ID_TXRX_VIRT3
2514            elif(node_name=="virt4"):
2515                return ID_TXRX_VIRT4
2516            elif(node_name=="virt5"):
2517                return ID_TXRX_VIRT5
2518            elif(node_name=="virt6"):
2519                return ID_TXRX_VIRT6
2520
2521        def getStartButtonId(self,node_name):
2522            if(node_name=="usrp0A"):
2523                return ID_START_USB0A
2524            elif(node_name=="usrp0B"):
2525                return ID_START_USB0B
2526            elif(node_name=="usrp1A"):
2527                return ID_START_USB1A
2528            elif(node_name=="usrp1B"):
2529                return ID_START_USB1B
2530            elif(node_name=="virt1"):
2531                return ID_START_VIRT1
2532            elif(node_name=="virt2"):
2533                return ID_START_VIRT2
2534            elif(node_name=="virt3"):
2535                return ID_START_VIRT3
```

```
2536            elif(node_name=="virt4"):
2537                return ID_START_VIRT4
2538            elif(node_name=="virt5"):
2539                return ID_START_VIRT5
2540            elif(node_name=="virt6"):
2541                return ID_START_VIRT6
2542
2543        def getStopButtonId(self,node_name):
2544            if(node_name=="usrp0A"):
2545                return ID_STOP_USB0A
2546            elif(node_name=="usrp0B"):
2547                return ID_STOP_USB0B
2548            elif(node_name=="usrp1A"):
2549                return ID_STOP_USB1A
2550            elif(node_name=="usrp1B"):
2551                return ID_STOP_USB1B
2552            elif(node_name=="virt1"):
2553                return ID_STOP_VIRT1
2554            elif(node_name=="virt2"):
2555                return ID_STOP_VIRT2
2556            elif(node_name=="virt3"):
2557                return ID_STOP_VIRT3
2558            elif(node_name=="virt4"):
2559                return ID_STOP_VIRT4
2560            elif(node_name=="virt5"):
2561                return ID_STOP_VIRT5
2562            elif(node_name=="virt6"):
2563                return ID_STOP_VIRT6
2564
2565        def getUpateButtonId(self,node_name):
2566            if(node_name=="usrp0A"):
2567                return ID_UPDATE_USB0A
2568            elif(node_name=="usrp0B"):
2569                return ID_UPDATE_USB0B
2570            elif(node_name=="usrp1A"):
2571                return ID_UPDATE_USB1A
2572            elif(node_name=="usrp1B"):
2573                return ID_UPDATE_USB1B
2574            elif(node_Name=="virt1"):
2575                return ID_UPDATE_VIRT1
2576            elif(node_Name=="virt2"):
2577                return ID_UPDATE_VIRT2
2578            elif(node_Name=="virt3"):
2579                return ID_UPDATE_VIRT3
2580            elif(node_Name=="virt4"):
```

```
2581                return ID_UPDATE_VIRT4
2582            elif(node_Name=="virt5"):
2583                return ID_UPDATE_VIRT5
2584            elif(node_Name=="virt6"):
2585                return ID_UPDATE_VIRT6
2586
2587        def getSrcNameId(self,node_name):
2588            if(node_name=="usrp0A"):
2589                return ID_SRC_NAME_USB0A
2590            elif(node_name=="usrp0B"):
2591                return ID_SRC_NAME_USB0B
2592            elif(node_name=="usrp1A"):
2593                return ID_SRC_NAME_USB1A
2594            elif(node_name=="usrp1B"):
2595                return ID_SRC_NAME_USB1B
2596
2597        def getRecipientId(self,node_name):
2598            if(node_name=="usrp0A"):
2599                return ID_RECIPIENT_USB0A
2600            elif(node_name=="usrp0B"):
2601                return ID_RECIPIENT_USB0B
2602            elif(node_name=="usrp1A"):
2603                return ID_RECIPIENT_USB1A
2604            elif(node_name=="usrp1B"):
2605                return ID_RECIPIENT_USB1B
2606
2607        def getDecodeProcessId(self,node_name):
2608            if(node_name=="usrp0A"):
2609                return ID_PROCESS_DEC_1
2610            elif(node_name=="usrp0B"):
2611                return ID_PROCESS_DEC_2
2612            elif(node_name=="usrp1A"):
2613                return ID_PROCESS_DEC_3
2614            elif(node_name=="usrp1B"):
2615                return ID_PROCESS_DEC_4
2616            elif(node_name=="virt1"):
2617                return ID_PROCESS_DEC_5
2618            elif(node_name=="virt2"):
2619                return ID_PROCESS_DEC_6
2620            elif(node_name=="virt3"):
2621                return ID_PROCESS_DEC_7
2622            elif(node_name=="virt4"):
2623                return ID_PROCESS_DEC_8
2624            elif(node_name=="virt5"):
2625                return ID_PROCESS_DEC_9
```

```
2626            elif(node_name=="virt6"):
2627                return ID_PROCESS_DEC_10
2628
2629        def getEncodeProcessId(self,node_name):
2630            if(node_name=="usrp0A"):
2631                return ID_PROCESS_ENC_1
2632            elif(node_name=="usrp0B"):
2633                return ID_PROCESS_ENC_2
2634            elif(node_name=="usrp1A"):
2635                return ID_PROCESS_ENC_3
2636            elif(node_name=="usrp1B"):
2637                return ID_PROCESS_ENC_4
2638            elif(node_name=="virt1"):
2639                return ID_PROCESS_ENC_5
2640            elif(node_name=="virt2"):
2641                return ID_PROCESS_ENC_6
2642            elif(node_name=="virt3"):
2643                return ID_PROCESS_ENC_7
2644            elif(node_name=="virt4"):
2645                return ID_PROCESS_ENC_8
2646            elif(node_name=="virt5"):
2647                return ID_PROCESS_ENC_9
2648            elif(node_name=="virt6"):
2649                return ID_PROCESS_ENC_10
2650
2651        def getDecodeProcess(self,node_name):
2652            if(node_name=="usrp0A"):
2653                return self.decode_process_usb0A
2654            elif(node_name=="usrp0B"):
2655                return self.decode_process_usb0B
2656            elif(node_name=="usrp1A"):
2657                return self.decode_process_usb1A
2658            elif(node_name=="usrp1B"):
2659                return self.decode_process_usb1B
2660            elif(node_name=="virt1"):
2661                return self.decode_process_virt1
2662            elif(node_name=="virt2"):
2663                return self.decode_process_virt2
2664            elif(node_name=="virt3"):
2665                return self.decode_process_virt3
2666            elif(node_name=="virt4"):
2667                return self.decode_process_virt4
2668            elif(node_name=="virt5"):
2669                return self.decode_process_virt5
2670            elif(node_name=="virt6"):
```

```
2671            return self.decode_process_virt6
2672
2673    def getEncodeProcess(self,node_name):
2674        if(node_name=="usrp0A"):
2675            return self.encode_process_usb0A
2676        elif(node_name=="usrp0B"):
2677            return self.encode_process_usb0B
2678        elif(node_name=="usrp1A"):
2679            return self.encode_process_usb1A
2680        elif(node_name=="usrp1B"):
2681            return self.encode_process_usb1B
2682        elif(node_name=="virt1"):
2683            return self.encode_process_virt1
2684        elif(node_name=="virt2"):
2685            return self.encode_process_virt2
2686        elif(node_name=="virt3"):
2687            return self.encode_process_virt3
2688        elif(node_name=="virt4"):
2689            return self.encode_process_virt4
2690        elif(node_name=="virt5"):
2691            return self.encode_process_virt5
2692        elif(node_name=="virt6"):
2693            return self.encode_process_virt6
2694
2695    def toggleRadioBoxSelection(self,id):
2696        if(self.FindWindowById(id).GetSelection()==0):
2697            self.FindWindowById(id).SetSelection(1)
2698        else:
2699            self.FindWindowById(id).SetSelection(0)
2700
2701    def decodeRxData(self,id):
2702        if(id==ID_PROCESS_1):
2703            if (self.decode_process_usb0A is None):
2704                self.decode_cmd = self.experiment_path+ "usrp0A/usrp "+ self.experiment_path+"
                        usrp0A/usrp0Aparams.ini"
2705                self.decode_process_usb0A = wx.Process(self,id=ID_PROCESS_DEC_1)
2706                self.decode_process_usb0A.Redirect()
2707                self.decode_process_id_usb0A = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                        decode_process_usb0A)
2708                self.display0A(">Decoding Received Message...\n")
2709                self.display0A(">Executing: "+self.decode_cmd+"\n")
2710        elif(id==ID_PROCESS_2):
2711            if (self.decode_process_usb0B is None):
2712                self.decode_cmd = self.experiment_path+ "usrp0B/usrp "+ self.experiment_path+"
                        usrp0B/usrp0Bparams.ini"
```

```
2713                self.decode_process_usb0B = wx.Process(self,id=ID_PROCESS_DEC_2)
2714                self.decode_process_usb0B.Redirect()
2715                self.decode_process_id_usb0B = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                        decode_process_usb0B)
2716                self.display0B(">Decoding Received Message...\n")
2717                self.display0B(">Executing: "+self.decode_cmd+"\n")
2718            elif(id==ID_PROCESS_3):
2719                if (self.decode_process_usb1A is None):
2720                    self.decode_cmd = self.experiment_path+ "usrp1A/usrp "+ self.experiment_path+"
                            usrp1A/usrp1Aparams.ini"   #used to be rx.ini
2721                    self.decode_process_usb1A = wx.Process(self,id=ID_PROCESS_DEC_3)
2722                    self.decode_process_usb1A.Redirect()
2723                    self.decode_process_id_usb1A = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                            decode_process_usb1A)
2724                    self.display1A(">Decoding Received Message...\n")
2725                    self.display1A(">Executing: "+self.decode_cmd+"\n")
2726            elif(id==ID_PROCESS_4):
2727                if (self.decode_process_usb1B is None):
2728                    self.decode_cmd = self.experiment_path+ "usrp1B/usrp "+ self.experiment_path+"
                            usrp1B/usrp1Bparams.ini"
2729                    self.decode_process_usb1B = wx.Process(self,id=ID_PROCESS_DEC_4)
2730                    self.decode_process_usb1B.Redirect()
2731                    self.decode_process_id_usb1B = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                            decode_process_usb1B)
2732                    self.display1B(">Decoding Received Message...\n")
2733                    self.display1B(">Executing: "+self.decode_cmd+"\n")
2734            #decode for additional nodes
2735            elif(id==ID_PROCESS_5):
2736                if (self.decode_process_virt1 is None):
2737                    self.decode_cmd = self.experiment_path+ "virt1/usrp "+ self.experiment_path+"virt1
                            /virt1params.ini"
2738                    self.decode_process_virt1 = wx.Process(self,id=ID_PROCESS_DEC_5)
2739                    self.decode_process_virt1.Redirect()
2740                    self.decode_process_id_virt1 = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                            decode_process_virt1)
2741                    self.olsr_v1_output+=">Decoding Received Message...\n"
2742                    self.olsr_v1_output+=">Executing: "+self.decode_cmd+"\n"
2743            elif(id==ID_PROCESS_6):
2744                if (self.decode_process_virt2 is None):
2745                    self.decode_cmd = self.experiment_path+ "virt2/usrp "+ self.experiment_path+"virt2
                            /virt2params.ini"
2746                    self.decode_process_virt2 = wx.Process(self,id=ID_PROCESS_DEC_6)
2747                    self.decode_process_virt2.Redirect()
2748                    self.decode_process_id_virt2 = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                            decode_process_virt2)
```

```
2749                self.olsr_v2_output+=">Decoding Received Message...\n"
2750                self.olsr_v2_output+=">Executing: "+self.decode_cmd+"\n"
2751            elif(id==ID_PROCESS_7):
2752                if (self.decode_process_virt3 is None):
2753                    self.decode_cmd = self.experiment_path+ "virt3/usrp "+ self.experiment_path+"virt3
                        /virt3params.ini"
2754                    self.decode_process_virt3 = wx.Process(self,id=ID_PROCESS_DEC_7)
2755                    self.decode_process_virt3.Redirect()
2756                    self.decode_process_id_virt3 = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                        decode_process_virt3)
2757                    self.olsr_v3_output+=">Decoding Received Message...\n"
2758                    self.olsr_v3_output+=">Executing: "+self.decode_cmd+"\n"
2759            elif(id==ID_PROCESS_8):
2760                if (self.decode_process_virt4 is None):
2761                    self.decode_cmd = self.experiment_path+ "virt4/usrp "+ self.experiment_path+"virt4
                        /virt4params.ini"
2762                    self.decode_process_virt4 = wx.Process(self,id=ID_PROCESS_DEC_8)
2763                    self.decode_process_virt4.Redirect()
2764                    self.decode_process_id_virt4 = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                        decode_process_virt4)
2765                    self.olsr_v4_output+=">Decoding Received Message...\n"
2766                    self.olsr_v4_output+=">Executing: "+self.decode_cmd+"\n"
2767            elif(id==ID_PROCESS_9):
2768                if (self.decode_process_virt5 is None):
2769                    self.decode_cmd = self.experiment_path+ "virt5/usrp "+ self.experiment_path+"virt5
                        /virt5params.ini"
2770                    self.decode_process_virt5 = wx.Process(self,id=ID_PROCESS_DEC_9)
2771                    self.decode_process_virt5.Redirect()
2772                    self.decode_process_id_virt5 = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                        decode_process_virt5)
2773                    self.olsr_v5_output+=">Decoding Received Message...\n"
2774                    self.olsr_v5_output+=">Executing: "+self.decode_cmd+"\n"
2775            elif(id==ID_PROCESS_10):
2776                if (self.decode_process_virt6 is None):
2777                    self.decode_cmd = self.experiment_path+ "virt6/usrp "+ self.experiment_path+"virt6
                        /virt6params.ini"
2778                    self.decode_process_virt6 = wx.Process(self,id=ID_PROCESS_DEC_10)
2779                    self.decode_process_virt6.Redirect()
2780                    self.decode_process_id_virt6 = wx.Execute(self.decode_cmd,wx.EXEC_ASYNC,self.
                        decode_process_virt6)
2781                    self.olsr_v6_output+=">Decoding Received Message...\n"
2782                    self.olsr_v6_output+=">Executing: "+self.decode_cmd+"\n"
2783
2784
2785        def radioClick(self,event):
```

```
2786            #change source name and sink name to defaults
2787            if(event.GetId()==ID_TXRX_USB0A):
2788                if(self.FindWindowById(ID_TXRX_USB0A).GetStringSelection()=="RX"):
2789                    self.FindWindowById(ID_SRC_NAME_USB0A).SetValue("usrp.srp")
2790                    self.FindWindowById(ID_SINK_NAME_USB0A).SetValue("usrp.rxd")
2791                else:
2792                    self.FindWindowById(ID_SRC_NAME_USB0A).SetValue("ID456.txt")
2793                    self.FindWindowById(ID_SINK_NAME_USB0A).SetValue("usrp.srp")
2794            elif(event.GetId()==ID_TXRX_USB0B):
2795                if(self.FindWindowById(ID_TXRX_USB0B).GetStringSelection()=="RX"):
2796                    self.FindWindowById(ID_SRC_NAME_USB0B).SetValue("usrp.srp")
2797                    self.FindWindowById(ID_SINK_NAME_USB0B).SetValue("usrp.rxd")
2798                else:
2799                    self.FindWindowById(ID_SRC_NAME_USB0B).SetValue("ID456.txt")
2800                    self.FindWindowById(ID_SINK_NAME_USB0B).SetValue("usrp.srp")
2801            elif(event.GetId()==ID_TXRX_USB1A):
2802                if(self.FindWindowById(ID_TXRX_USB1A).GetStringSelection()=="RX"):
2803                    self.FindWindowById(ID_SRC_NAME_USB1A).SetValue("usrp.srp")
2804                    self.FindWindowById(ID_SINK_NAME_USB1A).SetValue("usrp.rxd")
2805                else:
2806                    self.FindWindowById(ID_SRC_NAME_USB1A).SetValue("ID456.txt")
2807                    self.FindWindowById(ID_SINK_NAME_USB1A).SetValue("usrp.srp")
2808            elif(event.GetId()==ID_TXRX_USB1B):
2809                if(self.FindWindowById(ID_TXRX_USB1B).GetStringSelection()=="RX"):
2810                    self.FindWindowById(ID_SRC_NAME_USB1B).SetValue("usrp.srp")
2811                    self.FindWindowById(ID_SINK_NAME_USB1B).SetValue("usrp.rxd")
2812                else:
2813                    self.FindWindowById(ID_SRC_NAME_USB1B).SetValue("ID456.txt")
2814                    self.FindWindowById(ID_SINK_NAME_USB1B).SetValue("usrp.srp")
2815
2816        def getRSSIMeasurement(self,node_name):
2817            self.rssi=subprocess.Popen([r"/Users/marker/Documents/workspace/usrp_project/src/
                radio_interaction/usrp_rx_rssi.py",
2818                                        "-U",node_name[4],"-R",node_name[5],"-f","1250M","-d","128"],
2819                                        stdout=subprocess.PIPE).communicate()[0]
2820
2821        def onUpdate(self,event):
2822
2823
2824            if (event.GetId()==ID_UPDATE_USB0A):
2825                data = [self.FindWindowById(ID_RADIO_USB0A).GetLabelText(), self.FindWindowById(
                    ID_TXRX_USB0A).GetStringSelection(),
2826                        self.FindWindowById(ID_SRC_NAME_USB0A).GetValue(),self.FindWindowById(
                            SRC_ID_USB0A).GetValue(),
```

```
2827                        self.FindWindowById(ID_MSG_BITS_USB0A).GetValue(),self.FindWindowById(
                                ID_RAND_BITS_USB0A).GetValue(),
2828                        self.FindWindowById(ID_CLAMP_BITS_USB0A).GetValue(),self.FindWindowById(
                                ID_FRAG_BITS_USB0A).GetValue(),
2829                        self.FindWindowById(ID_STOP_BITS_USB0A).GetValue(), self.FindWindowById(
                                ID_EXPANSION_USB0A).GetValue(),
2830                        self.FindWindowById(ID_PKT_LOAD_USB0A).GetValue(),self.FindWindowById(
                                ID_DECODE_LIM_USB0A).GetValue(),
2831                        self.FindWindowById(ID_BUF_PKT_USB0A).GetValue(),self.FindWindowById(
                                ID_BUF_LAMBDA_USB0A).GetValue(),
2832                        self.FindWindowById(ID_PKT_RATE_USB0A).GetValue(),self.FindWindowById(
                                ID_SAMPLES_PER_BIT_USB0A).GetValue(),
2833                        self.FindWindowById(ID_GAIN_USB0A).GetValue(),self.FindWindowById(
                                ID_CHANNEL_LOSS_USB0A).GetValue(),
2834                        self.FindWindowById(ID_THRESHOLD_PCT_USB0A).GetValue(),self.FindWindowById(
                                ID_HYSTERESIS_USB0A).GetValue(),
2835                        self.FindWindowById(ID_JITTER_USB0A).GetValue(),self.FindWindowById(
                                ID_CUSHION_PCT_USB0A).GetValue(),
2836                        self.FindWindowById(ID_SINK_NAME_USB0A).GetValue(),self.FindWindowById(
                                ID_SINK_SAMPLE_LIM_USB0A).GetValue(),
2837                        self.FindWindowById(ID_RECIPIENT_USB0A).GetValue()]
2838                self.getRSSIMeasurement("usrp0A")
2839                self.writeParametersToFile(data)
2840                if(self.FindWindowById(ID_TXRX_USB0A).GetStringSelection()=="TX"):
2841                    self.encodeTxData(event.GetId())
2842                #print "Data: ", data
2843            elif(event.GetId()==ID_UPDATE_USB0B):
2844                data = [self.FindWindowById(ID_RADIO_USB0B).GetLabelText(),self.FindWindowById(
                        ID_TXRX_USB0B).GetStringSelection(),
2845                        self.FindWindowById(ID_SRC_NAME_USB0B).GetValue(),self.FindWindowById(
                                SRC_ID_USB0B).GetValue(),
2846                        self.FindWindowById(ID_MSG_BITS_USB0B).GetValue(),self.FindWindowById(
                                ID_RAND_BITS_USB0B).GetValue(),
2847                        self.FindWindowById(ID_CLAMP_BITS_USB0B).GetValue(),self.FindWindowById(
                                ID_FRAG_BITS_USB0B).GetValue(),
2848                        self.FindWindowById(ID_STOP_BITS_USB0B).GetValue(),self.FindWindowById(
                                ID_EXPANSION_USB0B).GetValue(),
2849                        self.FindWindowById(ID_PKT_LOAD_USB0B).GetValue(),self.FindWindowById(
                                ID_DECODE_LIM_USB0B).GetValue(),
2850                        self.FindWindowById(ID_BUF_PKT_USB0B).GetValue(),self.FindWindowById(
                                ID_BUF_LAMBDA_USB0B).GetValue(),
2851                        self.FindWindowById(ID_PKT_RATE_USB0B).GetValue(),self.FindWindowById(
                                ID_SAMPLES_PER_BIT_USB0B).GetValue(),
2852                        self.FindWindowById(ID_GAIN_USB0B).GetValue(),self.FindWindowById(
                                ID_CHANNEL_LOSS_USB0B).GetValue(),
```

```
2853                     self.FindWindowById(ID_THRESHOLD_PCT_USB0B).GetValue(),self.FindWindowById(
                             ID_HYSTERESIS_USB0B).GetValue(),
2854                     self.FindWindowById(ID_JITTER_USB0B).GetValue(),self.FindWindowById(
                             ID_CUSHION_PCT_USB0B).GetValue(),
2855                     self.FindWindowById(ID_SINK_NAME_USB0B).GetValue(),self.FindWindowById(
                             ID_SINK_SAMPLE_LIM_USB0B).GetValue(),
2856                     self.FindWindowById(ID_RECIPIENT_USB0B).GetValue()]
2857             self.getRSSIMeasurement("usrp0B")
2858             self.writeParametersToFile(data)
2859             if(self.FindWindowById(ID_TXRX_USB0B).GetStringSelection()=="TX"):
2860                 self.encodeTxData(event.GetId())
2861
2862             #print "Data: ", data
2863         elif(event.GetId()==ID_UPDATE_USB1A):
2864             data = [self.FindWindowById(ID_RADIO_USB1A).GetLabelText(),self.FindWindowById(
                     ID_TXRX_USB1A).GetStringSelection(),
2865                     self.FindWindowById(ID_SRC_NAME_USB1A).GetValue(),self.FindWindowById(
                             SRC_ID_USB1A).GetValue(),
2866                     self.FindWindowById(ID_MSG_BITS_USB1A).GetValue(),self.FindWindowById(
                             ID_RAND_BITS_USB1A).GetValue(),
2867                     self.FindWindowById(ID_CLAMP_BITS_USB1A).GetValue(),self.FindWindowById(
                             ID_FRAG_BITS_USB1A).GetValue(),
2868                     self.FindWindowById(ID_STOP_BITS_USB1A).GetValue(),self.FindWindowById(
                             ID_EXPANSION_USB1A).GetValue(),
2869                     self.FindWindowById(ID_PKT_LOAD_USB1A).GetValue(),self.FindWindowById(
                             ID_DECODE_LIM_USB1A).GetValue(),
2870                     self.FindWindowById(ID_BUF_PKT_USB1A).GetValue(),self.FindWindowById(
                             ID_BUF_LAMBDA_USB1A).GetValue(),
2871                     self.FindWindowById(ID_PKT_RATE_USB1A).GetValue(),self.FindWindowById(
                             ID_SAMPLES_PER_BIT_USB1A).GetValue(),
2872                     self.FindWindowById(ID_GAIN_USB1A).GetValue(),self.FindWindowById(
                             ID_CHANNEL_LOSS_USB1A).GetValue(),
2873                     self.FindWindowById(ID_THRESHOLD_PCT_USB1A).GetValue(),self.FindWindowById(
                             ID_HYSTERESIS_USB1A).GetValue(),
2874                     self.FindWindowById(ID_JITTER_USB1A).GetValue(),self.FindWindowById(
                             ID_CUSHION_PCT_USB1A).GetValue(),
2875                     self.FindWindowById(ID_SINK_NAME_USB1A).GetValue(),self.FindWindowById(
                             ID_SINK_SAMPLE_LIM_USB1A).GetValue(),
2876                     self.FindWindowById(ID_RECIPIENT_USB1A).GetValue()]
2877             self.getRSSIMeasurement("usrp1A")
2878             self.writeParametersToFile(data)
2879             if(self.FindWindowById(ID_TXRX_USB1A).GetStringSelection()=="TX"):
2880                 self.encodeTxData(event.GetId())
2881             #print "Data: ",data
2882         elif(event.GetId()==ID_UPDATE_USB1B):
```

```
2883            data = [self.FindWindowById(ID_RADIO_USB1B).GetLabelText(),self.FindWindowById(
                    ID_TXRX_USB1B).GetStringSelection(),
2884                    self.FindWindowById(ID_SRC_NAME_USB1B).GetValue(),self.FindWindowById(
                            SRC_ID_USB1B).GetValue(),
2885                    self.FindWindowById(ID_MSG_BITS_USB1B).GetValue(),self.FindWindowById(
                            ID_RAND_BITS_USB1B).GetValue(),
2886                    self.FindWindowById(ID_CLAMP_BITS_USB1B).GetValue(),self.FindWindowById(
                            ID_FRAG_BITS_USB1B).GetValue(),
2887                    self.FindWindowById(ID_STOP_BITS_USB1B).GetValue(),self.FindWindowById(
                            ID_EXPANSION_USB1B).GetValue(),
2888                    self.FindWindowById(ID_PKT_LOAD_USB1B).GetValue(),self.FindWindowById(
                            ID_DECODE_LIM_USB1B).GetValue(),
2889                    self.FindWindowById(ID_BUF_PKT_USB1B).GetValue(),self.FindWindowById(
                            ID_BUF_LAMBDA_USB1B).GetValue(),
2890                    self.FindWindowById(ID_PKT_RATE_USB1B).GetValue(),self.FindWindowById(
                            ID_SAMPLES_PER_BIT_USB1B).GetValue(),
2891                    self.FindWindowById(ID_GAIN_USB1B).GetValue(),self.FindWindowById(
                            ID_CHANNEL_LOSS_USB1B).GetValue(),
2892                    self.FindWindowById(ID_THRESHOLD_PCT_USB1B).GetValue(),self.FindWindowById(
                            ID_HYSTERESIS_USB1B).GetValue(),
2893                    self.FindWindowById(ID_JITTER_USB1B).GetValue(),self.FindWindowById(
                            ID_CUSHION_PCT_USB1B).GetValue(),
2894                    self.FindWindowById(ID_SINK_NAME_USB1B).GetValue(),self.FindWindowById(
                            ID_SINK_SAMPLE_LIM_USB1B).GetValue(),
2895                    self.FindWindowById(ID_RECIPIENT_USB1B).GetValue()]
2896            self.getRSSIMeasurement("usrp1B")
2897            self.writeParametersToFile(data)
2898            if(self.FindWindowById(ID_TXRX_USB1B).GetStringSelection()=="TX"):
2899                self.encodeTxData(event.GetId())
2900            #print "Data: ",data
2901        elif(event.GetId()==ID_UPDATE_VIRT1):
2902            #need to format data file correctly
2903
2904            self.rssi=random.randint(160,4092)#random rssi for these nodes
2905            self.writeParametersToFile(data)
2906            if(self.FindWindowById(ID_TXRX_USB1B).GetStringSelection()=="TX"):
2907                self.encodeTxData(event.GetId())
2908
2909    def encodeTxData(self,event_id):
2910        #print "in encode algorithm",id
2911        if(event_id==ID_UPDATE_USB0A):
2912            if (self.encode_process_usb0A is None):
2913
2914                #self.encode_cmd="ls -al"
```

```
2915                self.encode_cmd = self.experiment_path + "usrp0A/usrp " + self.experiment_path + "
                        usrp0A/usrp0Aparams.ini"
2916                self.encode_process_usb0A = wx.Process(self, id=ID_PROCESS_ENC_1)
2917                self.encode_process_usb0A.Redirect()
2918                self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                        encode_process_usb0A)
2919                self.display0A(">Encoding TX Message.\n")
2920                self.display0A(">Executing: "+self.encode_cmd+"\n")
2921            elif(event_id==ID_UPDATE_USB0B):
2922                if (self.encode_process_usb0B is None):
2923                    #self.encode_cmd="ls -al"
2924                    self.encode_cmd = self.experiment_path + "usrp0B/usrp " + self.experiment_path + "
                            usrp0B/usrp0Bparams.ini"
2925                    self.encode_process_usb0B = wx.Process(self, id=ID_PROCESS_ENC_2)
2926                    self.encode_process_usb0B.Redirect()
2927                    self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                            encode_process_usb0B)
2928                    self.display0B(">Encoding TX Message.\n")
2929                    self.display0B(">Executing: "+self.encode_cmd+"\n")
2930            elif(event_id==ID_UPDATE_USB1A):
2931                if (self.encode_process_usb1A is None):
2932                    #self.encode_cmd="ls -al"
2933                    self.encode_cmd = self.experiment_path + "usrp1A/usrp " + self.experiment_path + "
                            usrp1A/usrp1Aparams.ini"
2934                    self.encode_process_usb1A = wx.Process(self, id=ID_PROCESS_ENC_3)
2935                    self.encode_process_usb1A.Redirect()
2936                    self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                            encode_process_usb1A)
2937                    self.display1A(">Encoding TX Message.\n")
2938                    self.display1A(">Executing: "+self.encode_cmd+"\n")
2939            elif(event_id==ID_UPDATE_USB1B):
2940                if (self.encode_process_usb1B is None):
2941                    #self.encode_cmd="ls -al"
2942                    self.encode_cmd = self.experiment_path + "usrp1B/usrp " + self.experiment_path + "
                            usrp1B/usrp1Bparams.ini"
2943                    self.encode_process_usb1B = wx.Process(self, id=ID_PROCESS_ENC_4)
2944                    self.encode_process_usb1B.Redirect()
2945                    self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                            encode_process_usb1B)
2946                    self.display1B(">Encoding TX Message.\n")
2947                    self.display1B(">Executing: "+self.encode_cmd+"\n")
2948            #additional nodes for testing
2949            elif(event_id==ID_UPDATE_VIRT1):
2950                if (self.encode_process_virt1 is None):
2951                    #self.encode_cmd="ls -al"
```

```
2952                self.encode_cmd = self.experiment_path + "virt1/usrp " + self.experiment_path + "
                        virt1/virt1params.ini"
2953                self.encode_process_virt1 = wx.Process(self, id=ID_PROCESS_ENC_5)
2954                self.encode_process_virt1.Redirect()
2955                self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                        encode_process_virt1)
2956                self.olsr_v1_output+=">Encoding TX Message.\n"
2957                self.olsr_v1_output+=">Executing: "+self.encode_cmd+"\n"
2958            elif(event_id==ID_UPDATE_VIRT2):
2959                if (self.encode_process_virt2 is None):
2960                    #self.encode_cmd="ls -al"
2961                    self.encode_cmd = self.experiment_path + "virt2/usrp " + self.experiment_path + "
                            virt2/virt2params.ini"
2962                    self.encode_process_virt2 = wx.Process(self, id=ID_PROCESS_ENC_6)
2963                    self.encode_process_virt2.Redirect()
2964                    self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                            encode_process_virt2)
2965                    self.olsr_v2_output+=">Encoding TX Message.\n"
2966                    self.olsr_v2_output+=">Executing: "+self.encode_cmd+"\n"
2967            elif(event_id==ID_UPDATE_VIRT3):
2968                if (self.encode_process_virt3 is None):
2969                    #self.encode_cmd="ls -al"
2970                    self.encode_cmd = self.experiment_path + "virt3/usrp " + self.experiment_path + "
                            virt3/virt3params.ini"
2971                    self.encode_process_virt3 = wx.Process(self, id=ID_PROCESS_ENC_7)
2972                    self.encode_process_virt3.Redirect()
2973                    self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                            encode_process_virt3)
2974                    self.olsr_v3_output+=">Encoding TX Message.\n"
2975                    self.olsr_v3_output+=">Executing: "+self.encode_cmd+"\n"
2976            elif(event_id==ID_UPDATE_VIRT4):
2977                if (self.encode_process_virt4 is None):
2978                    #self.encode_cmd="ls -al"
2979                    self.encode_cmd = self.experiment_path + "virt4/usrp " + self.experiment_path + "
                            virt4/virt4params.ini"
2980                    self.encode_process_virt4 = wx.Process(self, id=ID_PROCESS_ENC_8)
2981                    self.encode_process_virt4.Redirect()
2982                    self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                            encode_process_virt4)
2983                    self.olsr_v4_output+=">Encoding TX Message.\n"
2984                    self.olsr_v4_output+=">Executing: "+self.encode_cmd+"\n"
2985            elif(event_id==ID_UPDATE_VIRT5):
2986                if (self.encode_process_virt5 is None):
2987                    #self.encode_cmd="ls -al"
```

```
2988                self.encode_cmd = self.experiment_path + "virt5/usrp " + self.experiment_path + "
                        virt5/virt5params.ini"
2989                self.encode_process_virt5 = wx.Process(self, id=ID_PROCESS_ENC_9)
2990                self.encode_process_virt5.Redirect()
2991                self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                        encode_process_virt5)
2992                self.olsr_v5_output+=">Encoding TX Message.\n"
2993                self.olsr_v5_output+=">Executing: "+self.encode_cmd+"\n"
2994            elif(event_id==ID_UPDATE_VIRT6):
2995                if (self.encode_process_virt6 is None):
2996                    #self.encode_cmd="ls -al"
2997                    self.encode_cmd = self.experiment_path + "virt6/usrp " + self.experiment_path + "
                        virt6/virt6params.ini"
2998                    self.encode_process_virt6 = wx.Process(self, id=ID_PROCESS_ENC_60)
2999                    self.encode_process_virt6.Redirect()
3000                    self.encode_process_id = wx.Execute(self.encode_cmd,wx.EXEC_ASYNC,self.
                        encode_process_virt6)
3001                    self.olsr_v6_output+=">Encoding TX Message.\n"
3002                    self.olsr_v6_output+=">Executing: "+self.encode_cmd+"\n"
3003
3004
3005        def writeParametersToFile(self,data):
3006            #print "Data From File: ",data
3007            #print "Path: ",self.experiment_path
3008
3009            if(data[0]=="USB0A"):
3010                node_id="usrp0A"
3011                node_address = self.nat_table.items()[0][0]
3012                from_address = self.nat_table.items()[0][0]
3013                filename = self.experiment_path+"usrp0A/usrp0Aparams.ini"
3014                self.display0A("\n>Writing parameters to file: "+filename+"\n")
3015            elif(data[0]=="USB0B"):
3016                node_id="usrp0B"
3017                node_address = self.nat_table.items()[1][0]
3018                from_address = self.nat_table.items()[1][0]
3019                filename = self.experiment_path+"usrp0B/usrp0Bparams.ini"
3020                self.display0B("\n>Writing parameters to file: "+filename+"\n")
3021            elif(data[0]=="USB1A"):
3022                node_id="usrp1A"
3023                node_address = self.nat_table.items()[2][0]
3024                from_address = self.nat_table.items()[2][0]
3025                filename = self.experiment_path+"usrp1A/usrp1Aparams.ini"
3026                self.display1A("\n>Writing parameters to file: "+filename+"\n")
3027            elif(data[0]=="USB1B"):
3028                node_id="usrp1B"
```

```
3029            node_address = self.nat_table.items()[3][0]
3030            from_address = self.nat_table.items()[3][0]
3031            filename = self.experiment_path+"usrp1B/usrp1Bparams.ini"
3032            self.display1B("\n>Writing parameters to file: "+filename+"\n")
3033
3034        try:
3035            file=open(filename,'w')
3036            file.write("#DIAGNOSTICS\n")
3037            file.write("\n")
3038            file.write("PATH=")
3039            file.write("\""+self.experiment_path+"\"")
3040            file.write("\n")
3041            file.write("# SCHEDULER Configuration\n")
3042            file.write("SCHEDULER_TX_notRX=")
3043            if(data[1] == "TX"):
3044                file.write("1")
3045            elif(data[1] == "RX"):
3046                file.write("0")
3047            file.write("\n")
3048            file.write("SCHEDULER_REALTIME=0")
3049            file.write("\n")
3050            file.write("# SOURCE Configuration\n")
3051            file.write("SOURCE_NAME=")
3052            file.write("\"" + data[2] + "\"")
3053            file.write("\n")
3054            if(data[1] == "TX"):
3055                file.write("SOURCE_ID = ")
3056                file.write(data[3])
3057                file.write("\n")
3058            file.write("# CODEC Configuration\n")
3059            file.write("CODEC_MESSAGE_BITS = ")
3060            file.write(data[4])
3061            file.write("\n")
3062            file.write("CODEC_RANDOM_BITS = ")
3063            file.write(data[5])
3064            file.write("\n")
3065            file.write("CODEC_CLAMP_BITS =")
3066            file.write(data[6])
3067            file.write("\n")
3068            file.write("CODEC_FRAGMENT_BITS = ")
3069            file.write(data[7])
3070            file.write("\n")
3071            file.write("CODEC_STOP_BITS = ")
3072            file.write(data[8])
3073            file.write("\n")
```

```
3074                file.write("CODEC_EXPANSION = ")
3075                file.write(data[9])
3076                file.write("\n")
3077                if(data[1] == "TX"):
3078                    file.write("CODEC_PACKET_LOAD = ")
3079                    file.write(data[10])
3080                    file.write("\n")
3081                file.write("CODEC_DECODE_LIMIT = ")
3082                file.write(data[11])
3083                file.write("\n")
3084                file.write("# BUFFER Configuration\n")
3085                file.write("BUFFER_PACKETS = ")
3086                file.write(data[12])
3087                file.write("\n")
3088                if(data[1] == "TX"):
3089                    file.write("BUFFER_LAMBDA = ")
3090                    file.write(data[13])
3091                    file.write("\n")
3092                file.write("# MODEM Configuration\n")
3093                file.write("MODEM_PACKET_RATE_BPS = ")
3094                file.write(data[14])
3095                file.write("\n")
3096                file.write("MODEM_SAMPLES_PER_BIT = ")
3097                file.write(data[15])
3098                file.write("\n")
3099                file.write("MODEM_GAIN_DB = ")
3100                file.write(data[16])
3101                file.write("\n")
3102                file.write("MODEM_CHANNEL_LOSS_DB = ")
3103                file.write(data[17])
3104                file.write("\n")
3105                file.write("MODEM_THRESHOLD_PCT = ")
3106                file.write(data[18])
3107                file.write("\n")
3108                file.write("MODEM_HYSTERESIS_PCT = ")
3109                file.write(data[19])
3110                file.write("\n")
3111                file.write("MODEM_JITTER_BITS = ")
3112                file.write(data[20])
3113                file.write("\n")
3114                file.write("MODEM_CUSHION_PCT = ")
3115                file.write(data[21])
3116                file.write("\n")
3117                file.write("# SINK Configuration\n")
3118                file.write("SINK_NAME = ")
```

```
3119                file.write("\"" + data[22] + "\"")
3120                file.write("\n")
3121                file.write("SINK_SAMPLE_LIMIT = ")
3122                file.write(data[23])
3123                file.write("\n")
3124                file.write("NODE_ID=")
3125                file.write("\""+node_id+"\"")
3126                file.write("\n")
3127                file.write("NODE_ADDRESS=")
3128                file.write(node_address)
3129                file.write("\n")
3130                if(data[1]=="TX"):
3131                    #TO_ADDRESS
3132                    file.write("TO_ADDRESS=")
3133                    if(data[24]=="All"):
3134                        file.write("888")
3135                    elif(data[24]=="usrp0A"):
3136                        file.write(self.nat_table.items()[0][0])
3137                    elif(data[24]=="usrp0B"):
3138                        file.write(self.nat_table.items()[1][0])
3139                    elif(data[24]=="usrp1A"):
3140                        file.write(self.nat_table.items()[2][0])
3141                    elif(data[24]=="usrp1B"):
3142                        file.write(self.nat_table.items()[3][0])
3143                    file.write("\n")
3144                    #FROM_ADDRESS
3145                    file.write("FROM_ADDRESS=")
3146                    file.write(from_address)
3147                    file.write("\n")
3148                    #rssi
3149                    file.write("RSSI=")
3150                    file.write(str(self.rssi))
3151                    file.write("\n")
3152            except IOError:
3153                self.FindWindowById(self.getLogWindowId(node_id)).AppendText("\n>File IO Error! Cannot
                        open " + filename + ".\n")
3154            finally:
3155                file.close()
3156
3157
3158        def createParameterControls(self):
3159            #self.parameterLabel = wx.StaticText(self, label="Parameters:")
3160            self.logwindow_label_1 = wx.StaticText(self, label="Log Window 1:")
3161            self.logwindow_label_2 = wx.StaticText(self, label="Log Window 2:")
3162            self.logwindow_label_3 = wx.StaticText(self, label="Log Window 3:")
```

```
3163            self.logwindow_label_4 = wx.StaticText(self, label="Log Window 4:")
3164            self.recipient_label = wx.StaticText(self,label="To:")
3165            self.source_name_label = wx.StaticText(self,label="MSG Source:")
3166            self.source_id_label = wx.StaticText(self,label="Source ID:")
3167            self.codec_label = wx.StaticText(self,label="Codec Configuration:")
3168            self.codec_message_bits_label = wx.StaticText(self, label="Message Bits:")
3169            self.codec_random_bits_label = wx.StaticText(self, label="Random Bits:")
3170            self.codec_clamp_bits_label = wx.StaticText(self, label="Clamp Bits:")
3171            self.codec_fragment_bits_label = wx.StaticText(self, label="Fragment Bits:")
3172            self.codec_stop_bits_label = wx.StaticText(self, label="Stop Bits:")
3173            self.codec_expansion_label = wx.StaticText(self, label="Expansion:")
3174            self.codec_packet_load_label = wx.StaticText(self, label="Packet Load:")
3175            self.codec_decode_limit_label = wx.StaticText(self, label="Decode Limit:")
3176            self.buffer_label = wx.StaticText(self,label="Buffer Configuration:")
3177            self.buffer_packets_label = wx.StaticText(self, label="Buffer Packets:")
3178            self.buffer_lambda_label = wx.StaticText(self, label="Buffer Lambda:")
3179            self.modem_label = wx.StaticText(self,label="Modem Configuration:")
3180            self.modem_pkt_rate_bps_label = wx.StaticText(self, label="Packet Rate (BPS):")
3181            self.modem_samples_per_bit_label = wx.StaticText(self, label="Samples Per Bit:")
3182            self.modem_gain_db_label = wx.StaticText(self, label="Gain (DB):")
3183            self.modem_channel_loss_db_label = wx.StaticText(self, label="Channel Loss(db):")
3184            self.modem_threshold_pct_label = wx.StaticText(self, label="Threshold Pct:")
3185            self.modem_hysteresis_pct_label = wx.StaticText(self, label="Hysteresis Pct:")
3186            self.modem_jitter_bits_label = wx.StaticText(self, label="Jitter Bits:")
3187            self.modem_cusion_pct_label = wx.StaticText(self, label="Cushion Pct:")
3188            self.sink_label = wx.StaticText(self,label="Sink Configuration:")
3189            self.sink_name_label = wx.StaticText(self, label="MSG Sink:")
3190            self.sink_sample_limit_label = wx.StaticText(self, label="Sample Limit:")
3191
3192            if(self.y_offset==0):
3193                self.logwindow = wx.TextCtrl(self,id=ID_LOGWIN_USB0A, style=wx.TE_MULTILINE|wx.
                        TE_AUTO_SCROLL)
3194            elif(self.y_offset==200):
3195                self.logwindow = wx.TextCtrl(self,id=ID_LOGWIN_USB0B, style=wx.TE_MULTILINE|wx.
                        TE_AUTO_SCROLL)
3196            elif(self.y_offset==400):
3197                self.logwindow = wx.TextCtrl(self,id=ID_LOGWIN_USB1A, style=wx.TE_MULTILINE|wx.
                        TE_AUTO_SCROLL)
3198            elif(self.y_offset==600):
3199                self.logwindow = wx.TextCtrl(self,id=ID_LOGWIN_USB1B, style=wx.TE_MULTILINE|wx.
                        TE_AUTO_SCROLL)
3200
3201            if(self.x_offset==0):
3202                self.updateButton = wx.Button(self, id=ID_UPDATE_USB0A, label="Update")
3203                self.startButton = wx.Button(self, id=ID_START_USB0A, label="Start")
```

```
3204            self.stopButton = wx.Button(self,id=ID_STOP_USB0A,label="Stop")
3205            self.radioLabel = wx.StaticText(self, id=ID_RADIO_USB0A, label="USB0A")
3206            self.recipient_combo_box = wx.ComboBox(self,id=ID_RECIPIENT_USB0A,value=self.
                    recipients[0],choices=self.recipients)
3207            self.TxOrRx = wx.RadioBox(self, id=ID_TXRX_USB0A, choices=self.transmission_mode,
                    style=wx.RA_HORIZONTAL)
3208            self.source_name_entry = wx.TextCtrl(self, id=ID_SRC_NAME_USB0A, value="ID456.txt")
3209            self.source_id_entry = wx.TextCtrl(self, id=SRC_ID_USB0A, value="456")
3210            self.codec_message_bits_entry = wx.TextCtrl(self, id=ID_MSG_BITS_USB0A, value="512")
3211            self.codec_random_bits_entry = wx.TextCtrl(self, id=ID_RAND_BITS_USB0A, value="8")
3212            self.codec_clamp_bits_entry = wx.TextCtrl(self, id=ID_CLAMP_BITS_USB0A, value="1")
3213            self.codec_fragment_bits_entry = wx.TextCtrl(self, id=ID_FRAG_BITS_USB0A, value="1")
3214            self.codec_stop_bits_entry = wx.TextCtrl(self, id=ID_STOP_BITS_USB0A, value="100")
3215            self.codec_expansion_entry = wx.TextCtrl(self, id=ID_EXPANSION_USB0A, value="100")
3216            self.codec_packet_load_entry = wx.TextCtrl(self, id=ID_PKT_LOAD_USB0A, value="2")
3217            self.codec_decode_limit_entry = wx.TextCtrl(self, id=ID_DECODE_LIM_USB0A,value="100")
3218            self.buffer_packets_entry = wx.TextCtrl(self, id=ID_BUF_PKT_USB0A,value="4.0")
3219            self.buffer_lambda_entry = wx.TextCtrl(self, id=ID_BUF_LAMBDA_USB0A,value="0.4")
3220            self.modem_pkt_rate_bps_entry = wx.TextCtrl(self, id=ID_PKT_RATE_USB0A, value="500000"
                    )
3221            self.modem_samples_per_bit_entry = wx.TextCtrl(self, id=ID_SAMPLES_PER_BIT_USB0A,
                    value="4")
3222            self.modem_gain_db_entry = wx.TextCtrl(self, id=ID_GAIN_USB0A,value="80.0")
3223            self.modem_channel_loss_db_entry = wx.TextCtrl(self, id=ID_CHANNEL_LOSS_USB0A,value="
                    3.0")
3224            self.modem_threshold_pct_entry = wx.TextCtrl(self, id=ID_THRESHOLD_PCT_USB0A,value="
                    46.0")
3225            self.modem_hysteresis_pct_entry = wx.TextCtrl(self, id=ID_HYSTERESIS_USB0A,value="5.0"
                    )
3226            self.modem_jitter_bits_entry = wx.TextCtrl(self, id=ID_JITTER_USB0A,value="2.0")
3227            self.modem_cusion_pct_entry = wx.TextCtrl(self, id=ID_CUSHION_PCT_USB0A,value="10.0")
3228            self.sink_name_entry = wx.TextCtrl(self, id=ID_SINK_NAME_USB0A,value="usrp.srp")
3229            self.sink_sample_limit_entry = wx.TextCtrl(self, id=ID_SINK_SAMPLE_LIM_USB0A,value="
                    8000000")
3230
3231        elif(self.x_offset==250):
3232            self.updateButton = wx.Button(self, id=ID_UPDATE_USB0B, label="Update")
3233            self.startButton = wx.Button(self, id=ID_START_USB0B, label="Start")
3234            self.stopButton = wx.Button(self,id=ID_STOP_USB0B,label="Stop")
3235            self.radioLabel = wx.StaticText(self, id=ID_RADIO_USB0B, label="USB0B")
3236            self.recipient_combo_box = wx.ComboBox(self,id=ID_RECIPIENT_USB0B,value=self.
                    recipients[0],choices=self.recipients)
3237            self.TxOrRx = wx.RadioBox(self, id=ID_TXRX_USB0B, choices=self.transmission_mode,
                    style=wx.RA_HORIZONTAL)
3238            self.source_name_entry = wx.TextCtrl(self, id=ID_SRC_NAME_USB0B, value="ID456.txt")
```

```
3239            self.source_id_entry = wx.TextCtrl(self, id=SRC_ID_USB0B, value="456")
3240            self.codec_message_bits_entry = wx.TextCtrl(self, id=ID_MSG_BITS_USB0B, value="512")
3241            self.codec_random_bits_entry = wx.TextCtrl(self, id=ID_RAND_BITS_USB0B, value="8")
3242            self.codec_clamp_bits_entry = wx.TextCtrl(self, id=ID_CLAMP_BITS_USB0B, value="1")
3243            self.codec_fragment_bits_entry = wx.TextCtrl(self, id=ID_FRAG_BITS_USB0B, value="1")
3244            self.codec_stop_bits_entry = wx.TextCtrl(self, id=ID_STOP_BITS_USB0B, value="100")
3245            self.codec_expansion_entry = wx.TextCtrl(self, id=ID_EXPANSION_USB0B, value="100")
3246            self.codec_packet_load_entry = wx.TextCtrl(self, id=ID_PKT_LOAD_USB0B, value="2")
3247            self.codec_decode_limit_entry = wx.TextCtrl(self, id=ID_DECODE_LIM_USB0B,value="100")
3248            self.buffer_packets_entry = wx.TextCtrl(self, id=ID_BUF_PKT_USB0B,value="4.0")
3249            self.buffer_lambda_entry = wx.TextCtrl(self, id=ID_BUF_LAMBDA_USB0B,value="0.4")
3250            self.modem_pkt_rate_bps_entry = wx.TextCtrl(self, id=ID_PKT_RATE_USB0B, value="500000"
                    )
3251            self.modem_samples_per_bit_entry = wx.TextCtrl(self, id=ID_SAMPLES_PER_BIT_USB0B,
                    value="4")
3252            self.modem_gain_db_entry = wx.TextCtrl(self, id=ID_GAIN_USB0B,value="80.0")
3253            self.modem_channel_loss_db_entry = wx.TextCtrl(self, id=ID_CHANNEL_LOSS_USB0B,value="
                    3.0")
3254            self.modem_threshold_pct_entry = wx.TextCtrl(self, id=ID_THRESHOLD_PCT_USB0B,value="
                    46.0")
3255            self.modem_hysteresis_pct_entry = wx.TextCtrl(self, id=ID_HYSTERESIS_USB0B,value="5.0"
                    )
3256            self.modem_jitter_bits_entry = wx.TextCtrl(self, id=ID_JITTER_USB0B,value="2.0")
3257            self.modem_cusion_pct_entry = wx.TextCtrl(self, id=ID_CUSHION_PCT_USB0B,value="10.0")
3258            self.sink_name_entry = wx.TextCtrl(self, id=ID_SINK_NAME_USB0B,value="usrp.srp")
3259            self.sink_sample_limit_entry = wx.TextCtrl(self, id=ID_SINK_SAMPLE_LIM_USB0B,value="
                    8000000")
3260
3261        elif(self.x_offset==500):
3262            self.updateButton = wx.Button(self, id=ID_UPDATE_USB1A, label="Update")
3263            self.startButton = wx.Button(self, id=ID_START_USB1A, label="Start")
3264            self.stopButton = wx.Button(self,id=ID_STOP_USB1A,label="Stop")
3265            self.radioLabel = wx.StaticText(self, id=ID_RADIO_USB1A, label="USB1A")
3266            self.recipient_combo_box = wx.ComboBox(self,id=ID_RECIPIENT_USB1A,value=self.
                    recipients[0],choices=self.recipients)
3267            self.TxOrRx = wx.RadioBox(self, id=ID_TXRX_USB1A, choices=self.transmission_mode,
                    style=wx.RA_HORIZONTAL)
3268            self.source_name_entry = wx.TextCtrl(self, id=ID_SRC_NAME_USB1A, value="ID456.txt")
3269            self.source_id_entry = wx.TextCtrl(self, id=SRC_ID_USB1A, value="456")
3270            self.codec_message_bits_entry = wx.TextCtrl(self, id=ID_MSG_BITS_USB1A, value="512")
3271            self.codec_random_bits_entry = wx.TextCtrl(self, id=ID_RAND_BITS_USB1A, value="8")
3272            self.codec_clamp_bits_entry = wx.TextCtrl(self, id=ID_CLAMP_BITS_USB1A, value="1")
3273            self.codec_fragment_bits_entry = wx.TextCtrl(self, id=ID_FRAG_BITS_USB1A, value="1")
3274            self.codec_stop_bits_entry = wx.TextCtrl(self, id=ID_STOP_BITS_USB1A, value="100")
3275            self.codec_expansion_entry = wx.TextCtrl(self, id=ID_EXPANSION_USB1A, value="100")
```

```
3276            self.codec_packet_load_entry = wx.TextCtrl(self, id=ID_PKT_LOAD_USB1A, value="2")
3277            self.codec_decode_limit_entry = wx.TextCtrl(self, id=ID_DECODE_LIM_USB1A,value="100")
3278            self.buffer_packets_entry = wx.TextCtrl(self, id=ID_BUF_PKT_USB1A,value="4.0")
3279            self.buffer_lambda_entry = wx.TextCtrl(self, id=ID_BUF_LAMBDA_USB1A,value="0.4")
3280            self.modem_pkt_rate_bps_entry = wx.TextCtrl(self, id=ID_PKT_RATE_USB1A, value="500000"
                    )
3281            self.modem_samples_per_bit_entry = wx.TextCtrl(self, id=ID_SAMPLES_PER_BIT_USB1A,
                    value="4")
3282            self.modem_gain_db_entry = wx.TextCtrl(self, id=ID_GAIN_USB1A,value="80.0")
3283            self.modem_channel_loss_db_entry = wx.TextCtrl(self, id=ID_CHANNEL_LOSS_USB1A,value="
                    3.0")
3284            self.modem_threshold_pct_entry = wx.TextCtrl(self, id=ID_THRESHOLD_PCT_USB1A,value="
                    46.0")
3285            self.modem_hysteresis_pct_entry = wx.TextCtrl(self, id=ID_HYSTERESIS_USB1A,value="5.0"
                    )
3286            self.modem_jitter_bits_entry = wx.TextCtrl(self, id=ID_JITTER_USB1A,value="2.0")
3287            self.modem_cusion_pct_entry = wx.TextCtrl(self, id=ID_CUSHION_PCT_USB1A,value="10.0")
3288            self.sink_name_entry = wx.TextCtrl(self, id=ID_SINK_NAME_USB1A,value="usrp.srp")
3289            self.sink_sample_limit_entry = wx.TextCtrl(self, id=ID_SINK_SAMPLE_LIM_USB1A,value="
                    8000000")
3290
3291        elif(self.x_offset==750):
3292            self.updateButton = wx.Button(self, id=ID_UPDATE_USB1B, label="Update")
3293            self.startButton = wx.Button(self, id=ID_START_USB1B, label="Start")
3294            self.stopButton = wx.Button(self,id=ID_STOP_USB1B,label="Stop")
3295            self.radioLabel = wx.StaticText(self, id=ID_RADIO_USB1B, label="USB1B")
3296            self.recipient_combo_box = wx.ComboBox(self,id=ID_RECIPIENT_USB1B,value=self.
                    recipients[0],choices=self.recipients)
3297            self.TxOrRx = wx.RadioBox(self, id=ID_TXRX_USB1B, choices=self.transmission_mode,
                    style=wx.RA_HORIZONTAL)
3298            self.source_name_entry = wx.TextCtrl(self, id=ID_SRC_NAME_USB1B, value="ID456.txt")
3299            self.source_id_entry = wx.TextCtrl(self, id=SRC_ID_USB1B, value="456")
3300            self.codec_message_bits_entry = wx.TextCtrl(self, id=ID_MSG_BITS_USB1B, value="512")
3301            self.codec_random_bits_entry = wx.TextCtrl(self, id=ID_RAND_BITS_USB1B, value="8")
3302            self.codec_clamp_bits_entry = wx.TextCtrl(self, id=ID_CLAMP_BITS_USB1B, value="1")
3303            self.codec_fragment_bits_entry = wx.TextCtrl(self, id=ID_FRAG_BITS_USB1B, value="1")
3304            self.codec_stop_bits_entry = wx.TextCtrl(self, id=ID_STOP_BITS_USB1B, value="100")
3305            self.codec_expansion_entry = wx.TextCtrl(self, id=ID_EXPANSION_USB1B, value="100")
3306            self.codec_packet_load_entry = wx.TextCtrl(self, id=ID_PKT_LOAD_USB1B, value="2")
3307            self.codec_decode_limit_entry = wx.TextCtrl(self, id=ID_DECODE_LIM_USB1B,value="100")
3308            self.buffer_packets_entry = wx.TextCtrl(self, id=ID_BUF_PKT_USB1B,value="4.0")
3309            self.buffer_lambda_entry = wx.TextCtrl(self, id=ID_BUF_LAMBDA_USB1B,value="0.4")
3310            self.modem_pkt_rate_bps_entry = wx.TextCtrl(self, id=ID_PKT_RATE_USB1B, value="500000"
                    )
```

```
3311            self.modem_samples_per_bit_entry = wx.TextCtrl(self, id=ID_SAMPLES_PER_BIT_USB1B,
                    value="4")
3312            self.modem_gain_db_entry = wx.TextCtrl(self, id=ID_GAIN_USB1B,value="80.0")
3313            self.modem_channel_loss_db_entry = wx.TextCtrl(self, id=ID_CHANNEL_LOSS_USB1B,value="
                    3.0")
3314            self.modem_threshold_pct_entry = wx.TextCtrl(self, id=ID_THRESHOLD_PCT_USB1B,value="
                    46.0")
3315            self.modem_hysteresis_pct_entry = wx.TextCtrl(self, id=ID_HYSTERESIS_USB1B,value="5.0"
                    )
3316            self.modem_jitter_bits_entry = wx.TextCtrl(self, id=ID_JITTER_USB1B,value="2.0")
3317            self.modem_cusion_pct_entry = wx.TextCtrl(self, id=ID_CUSHION_PCT_USB1B,value="10.0")
3318            self.sink_name_entry = wx.TextCtrl(self, id=ID_SINK_NAME_USB1B,value="usrp.srp")
3319            self.sink_sample_limit_entry = wx.TextCtrl(self, id=ID_SINK_SAMPLE_LIM_USB1B,value="
                    8000000")
3320
3321
3322    def doLayout(self):
3323        for control, x, y, width, height in \
3324        [
3325            #(self.parameterLabel,0, 0, -1, -1),
3326            (self.logwindow_label_1,1000,0,-1,-1),
3327            (self.logwindow_label_2,1000,200,-1,-1),
3328            (self.logwindow_label_3,1000,400,-1,-1),
3329            (self.logwindow_label_4,1000,600,-1,-1),
3330            (self.logwindow,980,self.y_offset+20,420,175),
3331            (self.updateButton,self.x_offset+20,10,-1,-1),
3332            (self.startButton,self.x_offset+95,10,50,30),
3333            (self.stopButton,self.x_offset+150,10,50,30),
3334
3335            (self.recipient_label,self.x_offset+130,40,-1,-1),
3336            (self.recipient_combo_box,self.x_offset+145,40,85,30),
3337            #(self.radioComboBox,self.x_offset+10,40,-1,-1),
3338            (self.radioLabel,self.x_offset,40,-1,-1),
3339            #TX or RX
3340            (self.TxOrRx,self.x_offset+40,40,85,30),
3341            #source configuration
3342            (self.source_name_label,self.x_offset+20,70,-1,-1),
3343            (self.source_name_entry,self.x_offset+100,70,-1,-1),
3344            (self.source_id_label,self.x_offset+20,100,-1,-1),
3345            (self.source_id_entry,self.x_offset+100,100,-1,-1),
3346            #codec configuration
3347            (self.codec_label,self.x_offset+10,130,-1,-1),
3348            (self.codec_message_bits_label,self.x_offset+20,160,-1,-1),
3349            (self.codec_message_bits_entry,self.x_offset+120,160,-1,-1),
3350            (self.codec_random_bits_label,self.x_offset+20,190,-1,-1),
```

```
3351                 (self.codec_random_bits_entry, self.x_offset+120,190,-1,-1),
3352                 (self.codec_clamp_bits_label, self.x_offset+20,220,-1,-1),
3353                 (self.codec_clamp_bits_entry, self.x_offset+120,220,-1,-1),
3354                 (self.codec_fragment_bits_label, self.x_offset+20,250,-1,-1),
3355                 (self.codec_fragment_bits_entry, self.x_offset+120,250,-1,-1),
3356                 (self.codec_stop_bits_label, self.x_offset+20,280,-1,-1),
3357                 (self.codec_stop_bits_entry, self.x_offset+120,280,-1,-1),
3358                 (self.codec_expansion_label, self.x_offset+20,310,-1,-1),
3359                 (self.codec_expansion_entry, self.x_offset+120,310,-1,-1),
3360                 (self.codec_packet_load_label, self.x_offset+20,340,-1,-1),
3361                 (self.codec_packet_load_entry, self.x_offset+120,340,-1,-1),
3362                 (self.codec_decode_limit_label, self.x_offset+20,370,-1,-1),
3363                 (self.codec_decode_limit_entry, self.x_offset+120,370,-1,-1),
3364                 #buffer configuration
3365                 (self.buffer_label, self.x_offset+10,400,-1,-1),
3366                 (self.buffer_packets_label, self.x_offset+20,430,-1,-1),
3367                 (self.buffer_packets_entry, self.x_offset+120,430,-1,-1),
3368                 (self.buffer_lambda_label, self.x_offset+20,460,-1,-1),
3369                 (self.buffer_lambda_entry, self.x_offset+120,460,-1,-1),
3370                 #modem configuration
3371                 (self.modem_label, self.x_offset+10,490,-1,-1),
3372                 (self.modem_pkt_rate_bps_label, self.x_offset+20,520,-1,-1),
3373                 (self.modem_pkt_rate_bps_entry, self.x_offset+130,520,-1,-1),
3374                 (self.modem_samples_per_bit_label, self.x_offset+20,550,-1,-1),
3375                 (self.modem_samples_per_bit_entry, self.x_offset+120,550,-1,-1),
3376                 (self.modem_gain_db_label, self.x_offset+20,580,-1,-1),
3377                 (self.modem_gain_db_entry, self.x_offset+120,580,-1,-1),
3378                 (self.modem_channel_loss_db_label, self.x_offset+20,610,-1,-1),
3379                 (self.modem_channel_loss_db_entry, self.x_offset+125,610,-1,-1),
3380                 (self.modem_threshold_pct_label, self.x_offset+20,640,-1,-1),
3381                 (self.modem_threshold_pct_entry, self.x_offset+125,640,-1,-1),
3382                 (self.modem_hysteresis_pct_label, self.x_offset+20,670,-1,-1),
3383                 (self.modem_hysteresis_pct_entry, self.x_offset+120,670,-1,-1),
3384                 (self.modem_jitter_bits_label, self.x_offset+20,700,-1,-1),
3385                 (self.modem_jitter_bits_entry, self.x_offset+120,700,-1,-1),
3386                 (self.modem_cusion_pct_label, self.x_offset+20,730,-1,-1),
3387                 (self.modem_cusion_pct_entry, self.x_offset+120,730,-1,-1),
3388                 #sink configuration
3389                 (self.sink_label, self.x_offset+10,760,-1,-1),
3390                 (self.sink_name_label, self.x_offset+20,790,-1,-1),
3391                 (self.sink_name_entry, self.x_offset+120,790,-1,-1),
3392                 (self.sink_sample_limit_label, self.x_offset+20,820,-1,-1),
3393                 (self.sink_sample_limit_entry, self.x_offset+120,820,-1,-1),
3394                 ]:
3395                 control.SetDimensions(x=x, y=y, width=width, height=height)
```

```
3396
3397        def OnClearWindow(self,event):
3398            dlg = wx.MessageDialog(self, "Are you sure you want to clear this log window?","Confirm
                    Clear Log", wx.OK|wx.CANCEL|wx.ICON_QUESTION)
3399            result = dlg.ShowModal()
3400            dlg.Destroy()
3401            if result == wx.ID_OK:
3402                if(event.GetId()==ID_CLEAR_LOGWIN_1):
3403                    self.FindWindowById(ID_LOGWIN_USB0A).Clear()
3404                elif(event.GetId()==ID_CLEAR_LOGWIN_2):
3405                    self.FindWindowById(ID_LOGWIN_USB0B).Clear()
3406                elif(event.GetId()==ID_CLEAR_LOGWIN_3):
3407                    self.FindWindowById(ID_LOGWIN_USB1A).Clear()
3408                elif(event.GetId()==ID_CLEAR_LOGWIN_4):
3409                    self.FindWindowById(ID_LOGWIN_USB1B).Clear()
3410                elif(event.GetId()==ID_CLEAR_LOGWIN_5):
3411                    self.olsr_v1_output=""
3412                elif(event.GetId()==ID_CLEAR_LOGWIN_6):
3413                    self.olsr_v2_output=""
3414                elif(event.GetId()==ID_CLEAR_LOGWIN_7):
3415                    self.olsr_v3_output=""
3416                elif(event.GetId()==ID_CLEAR_LOGWIN_8):
3417                    self.olsr_v4_output=""
3418                elif(event.GetId()==ID_CLEAR_LOGWIN_9):
3419                    self.olsr_v5_output=""
3420                elif(event.GetId()==ID_CLEAR_LOGWIN_10):
3421                    self.olsr_v6_output=""
3422                elif(event.GetId()==ID_CLEAR_LOGWIN_ALL):
3423                    self.FindWindowById(ID_LOGWIN_USB0A).Clear()
3424                    self.FindWindowById(ID_LOGWIN_USB0B).Clear()
3425                    self.FindWindowById(ID_LOGWIN_USB1A).Clear()
3426                    self.FindWindowById(ID_LOGWIN_USB1B).Clear()
3427                    self.olsr_v1_output=""
3428                    self.olsr_v2_output=""
3429                    self.olsr_v3_output=""
3430                    self.olsr_v4_output=""
3431                    self.olsr_v5_output=""
3432                    self.olsr_v6_output=""
3433
3434        def OnResetQueue(self,event):
3435            dlg = wx.MessageDialog(self, "Do you really want to reset all queues in this application?"
                    ,"Confirm Reset", wx.OK|wx.CANCEL|wx.ICON_QUESTION)
3436            result = dlg.ShowModal()
3437            dlg.Destroy()
3438            if result == wx.ID_OK:
```

```
3439                self.rx_pkt_queue_usrp0A=[]
3440                self.rx_pkt_queue_usrp0B=[]
3441                self.rx_pkt_queue_usrp1A=[]
3442                self.rx_pkt_queue_usrp1B=[]
3443                self.rx_pkt_queue_virt1=[]
3444                self.rx_pkt_queue_virt2=[]
3445                self.rx_pkt_queue_virt3=[]
3446                self.rx_pkt_queue_virt4=[]
3447                self.rx_pkt_queue_virt5=[]
3448                self.rx_pkt_queue_virt6=[]
3449                self.tx_pkt_queue_usrp0A=[]
3450                self.tx_pkt_queue_usrp0B=[]
3451                self.tx_pkt_queue_usrp1A=[]
3452                self.tx_pkt_queue_usrp1B=[]
3453                self.tx_pkt_queue_virt1=[]
3454                self.tx_pkt_queue_virt2=[]
3455                self.tx_pkt_queue_virt3=[]
3456                self.tx_pkt_queue_virt4=[]
3457                self.tx_pkt_queue_virt5=[]
3458                self.tx_pkt_queue_virt6=[]
3459                self.rx_pkt_queue_historical_usrp0A=[]
3460                self.rx_pkt_queue_historical_usrp0B=[]
3461                self.rx_pkt_queue_historical_usrp1A=[]
3462                self.rx_pkt_queue_historical_usrp1B=[]
3463                self.rx_pkt_queue_historical_virt1=[]
3464                self.rx_pkt_queue_historical_virt2=[]
3465                self.rx_pkt_queue_historical_virt3=[]
3466                self.rx_pkt_queue_historical_virt4=[]
3467                self.rx_pkt_queue_historical_virt5=[]
3468                self.rx_pkt_queue_historical_virt6=[]
3469                self.tx_pkt_queue_historical_usrp0A=[]
3470                self.tx_pkt_queue_historical_usrp0B=[]
3471                self.tx_pkt_queue_historical_usrp1A=[]
3472                self.tx_pkt_queue_historical_usrp1B=[]
3473                self.tx_pkt_queue_historical_virt1=[]
3474                self.tx_pkt_queue_historical_virt2=[]
3475                self.tx_pkt_queue_historical_virt3=[]
3476                self.tx_pkt_queue_historical_virt4=[]
3477                self.tx_pkt_queue_historical_virt5=[]
3478                self.tx_pkt_queue_historical_virt6=[]
3479
3480
3481        def OnClose(self, event):
3482            dlg = wx.MessageDialog(self, "Do you really want to close this application?","Confirm Exit
                    ", wx.OK|wx.CANCEL|wx.ICON_QUESTION)
```

```
3483            result = dlg.ShowModal()
3484            dlg.Destroy()
3485            if result == wx.ID_OK:
3486                self.olsr_bring_down_interfaces()
3487                self.killOLSRProcesses()
3488                self.Destroy()
3489
3490                #raise SystemExit
3491
3492        def OnAbout(self, event):
3493            dial = wx.MessageDialog(self, 'USRP GUI App v.1.1\nMark Kuhr', 'About', wx.OK | wx.
                    ICON_INFORMATION)
3494            dial.ShowModal()
3495
3496
3497    if __name__ == '__main__':
3498        app = wx.App(redirect=True)   # Error messages go to popup window
3499        top = Frame("USRP GUI Control Application")
3500        top.Show(True)
3501        app.MainLoop()
```

C.1.1 BBC Packet Class

```
1
2   ##############################
3   ## BEGIN BBC PACKET CLASS ##
4   ##############################
5
6   class packet(object):
7       def __init__(self, node_id=None, node_address=None, to_address=None, from_address=None,
8                    filename=None, marks=0, messages_found=0, packets_lost=0, rssi_value=0):
9           self.node_id = node_id
10          self.node_address = node_address
11          self.to_address=to_address
12          self.from_address=from_address
13          self.filename=filename
14          self.marks=marks
15          self.messages_found=messages_found
16          self.packets_lost=packets_lost
17          self.rssi_value = rssi_value
18
19
20      def IsAckPacket(self):
21          if(self.filename.find("Ack")>=0):
22              return True
```

```
23          else:
24              return False
25
26      def GetNodeAddress(self):
27          return self.node_address
28
29      def GetToAddress(self):
30          return self.to_address
31
32      def GetFromAddress(self):
33          return self.from_address
34
35      def GetNodeId(self):
36          return self.node_id
37
38      def GetFilename(self):
39          return self.filename
40
41      def GetRSSI(self):
42          return self.rssi_value
43
44      def print_pkt(self):
45          print "Node id: "+self.node_id
46          print "Node Address: "+self.node_address
47          print "To Address: "+self.to_address
48          print "From Address: "+self.from_address
49          print "Filename: "+self.filename
50          print "Marks: "+self.marks
51          print "Messages Found: "+self.messages_found
52          print "Packets lost: "+self.packets_lost
53          print "RSSI Value: "+self.rssi_value
54          print "\n"
```

C.2 BBC Encoder/Decoder [Bahn 2007]

C.2.1 usrp.c

```
1  /*****************************************************************************
2   * Main TX/RX Module for the Real-time BBC Codec/Modem                       *
3   *****************************************************************************
4   * Modified By:
5   * Mark G. Kuhr
6   * Auburn University
7   *
```

```
 8   * Original Author:
 9   * William L. Bahn                                                    *
10   * Academy Center for Information Security                             *
11   * Department of Computer Science                                      *
12   * United States Air Force Academy                                     *
13   * USAFA, CO 80840                                                     *
14   **************************************************************************
15   *
16   * DESCRIPTION
17   *
18   *
19   * The basic, high-level, flow is as follows:
20   *
21   * TX: The Transmitter
22   *
23   * The transmitter uses the following signal flow:
24   *
25   * SOURCE -> ENCODER -> BUFFER -> MODULATOR -> SINK
26   *
27   *
28   * RX: The Receiver
29   *
30   * The receiver used the following signal flow
31   *
32   * SOURCE -> MODEM -> BUFFER -> CODEC -> SINK
33   *
34   *
35           //-------------------------------------------------------------------

36
37   * the module supports both the transmitter and recevier functions.
38   *
39   *
40   */
41  /* Real-time BBC CODEC
42   *
43   * This program is designed to process the raw USRP output data and decode
44   * the resulting packets in real-time in a streaming fashion. Since it is
45   * a real-time application, structural overhead has been minimized and
46   * global variables have been used extensively.
47   *
48   * THE DATA BUFFER
49   *
50   * The data is stored in a circular buffer with the following variables:
51   *     buffer: Pointer to the block of memory where the buffer starts.
52   *     read:   Index of the first byte of the present packet.
```

```
53   *     write:  Index of the next unused buffer location.
54   *     fill:   How many bytes are in buffer beyond the scope of the CODEC.
55   *     unused: How many unused bytes are available in the buffer.
56   *
57   * The buffer is seen by two functions, the one that is demodulating the
58   * data packet and the one that is decoding the resulting data. The
59   * demodulating function writes to the buffer at a nominally constant
60   * rate dictated by the communications link. In this application, this is
61   * simulated by reading the stored waveform data from a file and querying
62   * the clock to determine how many bytes to add to the buffer each time
63   * the function is called. The decoding function, on the other hand, always
64   * to decodes eight packets each time it is called provided sufficient data
65   * is available. Specifically, it decodes the eight packets that start with
66   * the bits in the byte stored at the "read" pointer. Since it can't decode
67   * packets that are not completely contained in the buffer, the decoding
68   * function first checks to see if "fill" is non-negative. If it isn't, then
69   * it returns immediately. At the other end of the spectrum, the MODEM
70   * may run out of unused memory to write to. If this happens, data is going
71   * to be lost. It is cleaner to throw away old data instead of introducing
72   * a gap in present data, therefore the MODEM will push the "read"
73   * pointer forward as it overwrites the beginning of the existing packet
74   * data.
75   *
76   */
77
78  //-------------------------------------------------------------------------
79  // FILE INCLUSIONS
80  //-------------------------------------------------------------------------
81
82  #include <stdio.h>  // printf()
83  #include <stdlib.h> // exit(), EXIT_SUCCESS, EXIT_FAILURE
84  #include <string.h>
85  #include <time.h>   // clock(), CLOCKS_PER_SEC
86
87
88  #include "bbcftp.h"
89
90  #include "config.h"
91  #include "source.h"
92  #include "codec.h"
93  #include "buffer.h"
94  #include "modem.h"
95  #include "sink.h"
96
97  //-------------------------------------------------------------------------
```

```
98  // TRANSMITTER
99  //-------------------------------------------------------------------------
100
101 int tx(BBCFTP *sys)
102 {
103         int state;
104         FILE *log_file;
105         char *log_file_name;
106
107         log_file_name = strdup(sys->config->path);
108         strcat(log_file_name,sys->config->node_id);
109         strcat(log_file_name,"/");
110         strcat(log_file_name,"logtx.txt");
111         //printf("Log File TX is: %s\n",log_file_name);
112
113         //-------------------------------------------------------------------
114         // Runtime scheduler
115         //-------------------------------------------------------------------
116
117         state = 0;
118         sys->config->tot_ticks = clock();
119         while ( (sys->sink->streaming) && ( sys->source->streaming || sys->buffer->ready ) )
120         {
121                 switch (state)
122                 {
123                         case 0: // Scheduler
124                                 if ( (sys->sink->streaming) && (sys->config->actual_trx_bytes <
                                        sys->config->nominal_trx_bytes) )
125                                 {
126                                         state = 1; // Run MODEM until sampling is caught up
127                                 }
128                                 else if ((sys->source->streaming) &&(0 <= sys->buffer->margin))
129                                 {
130                                         state = 2; // Encode packets subject to maximum amount of
                                                time.
131                                 }
132                                 else
133                                 {
134                                         if (sys->config->scheduler_realtime)
135                                                 sys->config->nominal_trx_bytes = (DWORD) ((clock()
                                                        - sys->config->tot_ticks) * sys->config->
                                                        bytespertick);
136                                         else
137                                                 sys->config->nominal_trx_bytes += 1;//(DWORD) (
                                                        config->bytespertick);
```

```
138                                     }
139
140                                     break;
141
142                             case 1: // Modulator
143                                     if (sys->buffer->ready == 1000)
144                                             state = 100;
145                                     if (sys->buffer->ready == 100)
146                                             state = 10;
147                                     if (sys->buffer->ready == 10)
148                                             state = 1;
149                                     if (sys->buffer->ready == 1)
150                                             state = 1;
151                                     if (sys->buffer->ready == 0)
152                                             state = 0;
153                                     Modulate(sys->config, sys->buffer, sys->modem, sys->sink);
154                                     state = 0;
155                                     break;
156
157                             case 2: // Encoder
158                                     Encode(sys->config, sys->source, sys->codec, sys->buffer);
159                                     if (!sys->source->streaming)
160                                             state = 100;
161                                     state = 0;
162                                     break;
163                     }
164             }
165             sys->config->tot_ticks = clock() - sys->config->tot_ticks;
166
167             //---------------------------------------------------------------------
168             // POST RUN CODE
169             //---------------------------------------------------------------------
170
171
172             //add 'marks' to TX log file
173             log_file = fopen(log_file_name,"wb");
174             if(log_file)
175             {
176                     fprintf(log_file,"%li\n", sys->config->marks);//Marks
177             }
178
179             printf("\n");
180             printf("Marks: %li\n", sys->config->marks);
181             printf("Total time:..... %0.3f sec.\n", ( (double)sys->config->tot_ticks / (double)
                    CLOCKS_PER_SEC ));
```

```
182         printf("MODEM time:.......... %0.3f sec.\n", ( (double)sys->config->dem_ticks / (double)
                CLOCKS_PER_SEC ));
183         printf("CODEC time:.......... %0.3f sec.\n", ( (double)sys->config->dec_ticks / (double)
                CLOCKS_PER_SEC ));
184
185         SINK_Purge(sys->config, sys->sink, sys->buffer);
186
187         return EXIT_SUCCESS;
188 }
189
190
191 //-------------------------------------------------------------------------
192 // RECEVIER
193 //-------------------------------------------------------------------------
194
195 int rx(BBCFTP *sys)
196 {
197         int state;
198         double vmax;
199
200         FILE *log_file;
201         char *log_file_name;
202
203         log_file_name = strdup(sys->config->path);
204         strcat(log_file_name,sys->config->node_id);
205         strcat(log_file_name,"/");
206         strcat(log_file_name,"logrx.txt");
207
208         //---------------------------------------------------------------------
209         // Runtime scheduler
210         //---------------------------------------------------------------------
211
212         vmax = 0;
213         state = 0;
214         sys->config->tot_ticks = clock();
215         while ( (sys->source->streaming) || (0 <= sys->buffer->margin) )
216         {
217                 switch (state)
218                 {
219                         case 0: // Scheduler
220                                 if ( (sys->source->streaming) && (sys->config->actual_trx_bytes <
                                        sys->config->nominal_trx_bytes) )
221                                 {
222                                         state = 1; // Run MODEM until sampling is caught up
223                                 }
```

```
224                                 else if (0 <= sys->buffer->margin)
225                                 {
226                                         state = 2; // Decode packets subject to maximum amount of
                                                time.
227                                 }
228                                 else
229                                 {
230                                         if (sys->config->scheduler_realtime)
231                                                 sys->config->nominal_trx_bytes = (DWORD) ((clock()
                                                        - sys->config->tot_ticks) * sys->config->
                                                        bytespertick);
232                                         else
233                                                 sys->config->nominal_trx_bytes += (DWORD) (sys->
                                                        config->bytespertick);
234                                 }
235
236                                 break;
237
238                         case 1: // MODEM
239                                 Demodulate(sys->config, sys->source, sys->modem, sys->buffer);
240                                 state = 0;
241                                 break;
242
243                         case 2: // CODEC
244                                 Decode(sys->config, sys->buffer, sys->codec, sys->sink);
245                                 state = 0;
246                                 break;
247                 }
248         }
249         sys->config->tot_ticks = clock() - sys->config->tot_ticks;
250
251         //-----------------------------------------------------------------
252         // POST RUN CODE
253         //-----------------------------------------------------------------
254         if(sys->config->marks == 0)
255         {
256                 //add 'marks' to TX log file
257                 log_file = fopen(log_file_name,"wb");
258                 if(log_file)
259                 {
260                         fprintf(log_file,"%li\n", sys->config->marks);//Marks
261                 }
262         }
263         printf("\n");
264         printf("Marks: %li\n", sys->config->marks);
```

```
265         printf("Messages found: %lu\n", sys->config->message_count);
266         printf("Packets lost: %lu\n", (DWORD) (sys->buffer->overflows * 8));
267         printf("Total time:.......... %0.3f sec.\n", ( (double)sys->config->tot_ticks / (double)
                CLOCKS_PER_SEC ));
268         printf("MODEM time:.......... %0.3f sec.\n", ( (double)sys->config->dem_ticks / (double)
                CLOCKS_PER_SEC ));
269         printf("CODEC time:.......... %0.3f sec.\n", ( (double)sys->config->dec_ticks / (double)
                CLOCKS_PER_SEC ));
270
271         //prints out the contents of the sink
272         SINK_Purge(sys->config, sys->sink, sys->buffer);
273
274         return EXIT_SUCCESS;
275 }
276
277 //---------------------------------------------------------------------------
278 // MAIN PROGRAM
279 //---------------------------------------------------------------------------
280
281 int main(int argc, char *argv[])
282 {
283         BBCFTP *sys;
284
285         char *config_file_name;
286         DWORD errcode;
287
288         int res;
289
290         //-----------------------------------------------------------------
291         // Read configuration information
292         //-----------------------------------------------------------------
293
294         config_file_name = NULL;
295         if (argc < 2)
296         {
297                 printf("Mode (T or R): ");
298                 res = getc(stdin);
299                 switch (res)
300                 {
301                         case 'T':
302                         case 't':
303                                 config_file_name = "tx.ini";
304                                 break;
305                         case 'R':
306                         case 'r':
```

```
307                         default :
308                                 config_file_name = "rx.ini";
309                 }
310                 while ('\n' != res)
311                         res = getc(stdin);
312         }
313         else
314                 config_file_name = argv[1];
315
316         //printf("file name: %s",config_file_name);
317
318         sys = BBCFTP_New(config_file_name, &errcode);
319         if (errcode)
320         {
321                 printf("BBC FTP System Constructor exited with error code: %lu\n", errcode);
322                 exit(EXIT_FAILURE);
323         }
324
325         //-----------------------------------------------------------------
326         // Launch transmitter or recever as appropriate
327         //-----------------------------------------------------------------
328
329         if (sys->config->scheduler_TX_notRX)
330                 tx(sys);
331         else
332                 rx(sys);
333
334         //-----------------------------------------------------------------
335         // Runtime Scheduler
336         //-----------------------------------------------------------------
337
338         // The components of the new scheduler are not yet complete.
339         // while (sys->sink->streaming)
340         // {
341         //              SOURCE_Run(sys);
342         //              CODEC_Run(sys);
343         //              MODEM_Run(sys);
344         //              SINK_Run(sys);
345         // }
346
347         //-----------------------------------------------------------------
348         // Final Housekeeping
349         //-----------------------------------------------------------------
350
351         BBCFTP_Del(sys);
```

```
352
353         /*printf("\n - Hit ENTER to exit.");
354         getc(stdin);*/
355
356         return EXIT_SUCCESS;
357 }
```

C.2.2 source.c and source.h

```
1  /*************************************************************************
2   * Data Source Module for the Real-time BBC Codec/Modem                  *
3   *************************************************************************
4   * Modified By:
5   * Mark G. Kuhr
6   * Auburn University
7   *
8   * Original Author:
9   * William L. Bahn                                                       *
10  * Academy Center for Information Security                               *
11  * Department of Computer Science                                        *
12  * United States Air Force Academy                                       *
13  * USAFA, CO 80840                                                       *
14  *************************************************************************
15  *
16  * DESCRIPTION
17  *
18  * This module supports the data source for both the TX and the RX.
19  *
20  */
21
22  //-----------------------------------------------------------------------
23  // REQUIRED INCLUDES
24  //-----------------------------------------------------------------------
25
26  #include <string.h>  // memmove()
27  #include <stdlib.h> // malloc(), free()
28
29  #include "bbcftp.h"
30  #include "source.h"
31  #include "dirtyd.h"
32
33  //-----------------------------------------------------------------------
34  // STRUCTURE DEFINITIONS
35  //-----------------------------------------------------------------------
36
```

```
37  // NOTE: Normally the structure definition would be in the *.c file to make
38  // the structure members inaccessible to outside functions except through
39  // public function calls. But for the real-time code it has been decided
40  // to make the structure members directly visible to the functions that
41  // manipulate them.
42
43  //---------------------------------------------------------------------------
44  // PUBLIC FUNCTION DEFINITIONS
45  //---------------------------------------------------------------------------
46
47  SOURCE *SOURCE_Del(SOURCE *p)
48  {
49          if (p)
50          {
51                  if (p->v)     { free(p->v);   p->v = NULL;       }
52                  free(p);
53                  p = NULL;
54          }
55
56          return p;
57  }
58
59  /*
60   * Eventually the source and sink will be Gnu Radio and therefore very
61   * little effort has been made to make this temporary source flexible
62   * or sophisticated. The SOURCE_New() function opens the source file,
63   * allocated memory for the entire contents, loads the entire contents
64   * into memory, and then closes the source file.
65   *
66   * TX: If configured as a transmitter, the data file is assumed to be a
67   * binary file that is to be transmitted across a BBC link. The file
68   * is brought up into memory as a series of messages using the following
69   * format:
70   *
71   * [Checksum] [SeqNum] [DataBits] [Data]
72   *
73   * The sequence number is a 16-bit number starting at 0 and incrementing
74   * by one for each packet. The length field is also a 16-bit number that
75   * contains the number of bits of actual data follows. The data field
76   * contains a string of bits read directly from the file being transmitted.
77   * It is a fixed width field and is zero padded if necessary. The checksum
78   * field is the last 32-bits of the message and contains a CRC checksum for
79   * message up to, but not including, the checksum field. At the present time,
80   * the checksum field is set to all zeros.
81   *
```

```
82   */
83
84  DWORD SOURCE_NewTX(SOURCE *p, CONFIG *c)
85  {
86          DWORD err;
87          FILE *fp;
88          BYTE *base;
89          DWORD bytes_read;
90          WORD seqnum, loadbits, id, length, toaddr, fromaddr, rssi;
91          BYTE *buffer;
92
93          err = 0;
94
95          // Initialize state information
96          p->streaming = TRUE;
97          p->sample = 0;
98          p->samples = 0;
99
100         // Data Source
101         fp = NULL;
102         if (c->source_name)
103         {
104                 //printf("source name in newTx: %s\n",c->source_name);
105                 fp = fopen(c->source_name, "rb");
106                 if (!fp)
107                         err |= 1 << 7;
108         }
109
110         // Create Data Read Buffer
111         if (fp)
112         {
113                 // Determine the size of the file
114                 fseek(fp, 0, SEEK_END);
115                 p->file_bytes = ftell(fp);
116                 fseek(fp, 0, SEEK_SET);
117
118                 // How much memory each message needs in the Source Buffer
119                 p->sample_size_bytes = c->bytes_per_message;
120
121                 // Determine if each message can carry at least one file byte.
122                 if ( !((c->codec_message_bits/8) > BBC_FTP_HEADER_BYTES) )
123                         err |= 1 << 4;
124         }
125
126         if (!err)
```

```
127         {
128                 // Calculate how many bytes of the file each message can hold.
129                 p->chunk_size = (c->codec_message_bits/8) - BBC_FTP_HEADER_BYTES;       // File
                        bytes per message
130                 p->sample_limit = p->file_bytes / p->chunk_size; // Messages needed for whole
                        chunks
131                 if ( p->file_bytes % p->chunk_size )              // Plus one for any partial chunk
132                         p->sample_limit++;
133                 p->sample_limit+=2;                             // Plus one each for header/trailer
134
135                 p->buffer_size = p->sample_limit * p->sample_size_bytes;
136                 if (p->buffer_size)
137                         p->v = malloc(p->buffer_size);
138                 else
139                         err |= 1 << 2;
140                 if (!p->v)
141                         err |= 1 << 3;
142
143                 buffer = malloc(p->chunk_size);
144                 if (!buffer)
145                         err |= 1 << 5;
146         }
147
148         // Fill Data Buffer
149         if (!err)
150         {
151                 p->samples = 0;
152                 id = c->source_id;
153                 toaddr = c->to_address;
154                 fromaddr = c->from_address;
155                 rssi = c->rssi;
156                 seqnum = 0;
157                 bytes_read = 0;
158                 do
159                 {
160                         base = p->v + p->samples * p->sample_size_bytes;
161
162                         length = p->chunk_size;
163                         if ((p->file_bytes - bytes_read) <= length)
164                                 length = p->file_bytes - bytes_read;
165
166                         if (seqnum)
167                         {
168                                 if (length)
169                                         length = fread(buffer, 1, length, fp);
```

```
170                                     SetMessagePayload(base, buffer, length, 0);
171                                     bytes_read += length;
172                             }
173                             else
174                             {
175                                     length = strlen(c->source_name);
176                                     if (length > p->chunk_size)
177                                             length = p->chunk_size;
178                                     SetMessagePayload(base, (BYTE *) c->source_name, length, 0);
179                             }
180
181                             loadbits = 8 * length;
182                             SetMessageChecksum(base, 0); // Force checksum to zero (temporary
                                    convenience)
183                             SetMessageSeq(base, seqnum);
184                             SetMessageLoadBits(base, loadbits);
185                             SetMessageID(base, c->source_id);
186                             SetMessageToAddr(base,c->to_address);
187                             SetMessageFromAddr(base,c->from_address);
188                             SetMessageRSSI(base,c->rssi);
189
190                             seqnum++;
191                             if (c->diagnostics)
192                                     PrintMessage(base);
193                             p->samples++;
194
195                     } while (length);
196
197                     fclose(fp);
198                     fp = NULL;
199             }
200
201             return err;
202     }
203
204     DWORD SOURCE_NewRX(SOURCE *p, CONFIG *c)
205     {
206             DWORD err;
207             FILE *fp;
208
209             err = 0;
210
211             // Initialize state information
212             p->streaming = TRUE;
213             p->samples = 0;
```

```
214
215         // Data Source
216         fp = NULL;
217         if (c->source_name)
218         {
219                 fp = fopen(c->source_name, "rb");
220                 if (!fp)
221                         err |= 1 << 7;
222         }
223
224         // Create Data Read Buffer
225         if (fp)
226         {
227                 // Determine the size of the file
228                 fseek(fp, 0, SEEK_END);
229                 p->file_bytes = ftell(fp);
230                 fseek(fp, 0, SEEK_SET);
231
232                 p->sample_size_bytes = c->source_sample_size_bytes;
233                 // Determine number of complete samples in data file
234                 p->sample_limit = p->file_bytes / p->sample_size_bytes;
235                 // Adjust sample limit if initialization file sets a lower limit
236                 if ((c->source_sample_limit)&&(p->sample_limit > c->source_sample_limit))
237                         p->sample_limit = c->source_sample_limit;
238
239                 p->buffer_size = p->sample_limit * p->sample_size_bytes;
240                 if (p->buffer_size)
241                         p->v = malloc(p->buffer_size);
242                 else
243                         err |= 1 << 2;
244                 if (!p->v)
245                         err |= 1 << 3;
246         }
247
248         // Fill Data Buffer
249         if (!err)
250         {
251                 p->sample_limit = fread(p->v, p->sample_size_bytes, p->sample_limit, fp);
252                 fclose(fp);
253                 fp = NULL;
254         }
255
256         return err;
257 }
258
```

```
259 SOURCE *SOURCE_New(CONFIG *c, DWORD *errcode)
260 {
261         DWORD err;
262         SOURCE *p;
263
264         p = NULL;
265         err = 0;
266
267         p = (SOURCE *) malloc(sizeof(SOURCE));
268         if (!p)
269                 err |= 1 << 0;
270
271         if (!err)
272                 if (c->scheduler_TX_notRX)
273                         err = SOURCE_NewTX(p, c);
274                 else
275                         err = SOURCE_NewRX(p, c);
276
277         if (c->diagnostics)
278         {
279                 // Diagnostic Report
280                 printf("----------------------------------------\n");
281                 if (c->scheduler_TX_notRX)
282                 {
283                         printf("MESSAGE SOURCE\n");
284                         printf("  To Address:.............. %lu \n", (unsigned long) c->to_address)
                                ;
285                         printf("  From Address:........... %lu \n", (unsigned long) c->
                            from_address);
286                         printf("  RSSI:................... %lu \n", (unsigned long) c->rssi);
287                 }
288                 else
289                         printf("USRP SOURCE\n");
290
291                 printf("  File name:.............. %s\n", c->source_name);
292                 printf("  Node ID:................ %s\n", c->node_id);
293                 printf("  Creation:............... %s\n", ((err)? "FAILED":"SUCCEEDED"));
294                 printf("  Location:............... %p\n", (void *) p);
295                 printf("  File size:.............. %lu bytes\n", (unsigned long) p->file_bytes);
296                 printf("  Chunk size:............. %lu bytes\n", (unsigned long) p->chunk_size);
297                 printf("  Messages needed:........ %lu\n", (unsigned long) p->sample_limit);
298                 printf("  Message requirements:... %lu bytes\n", (unsigned long) p->
                        sample_size_bytes);
299                 printf("  Buffer size:............ %lu bytes\n", (unsigned long) p->buffer_size);
300                 printf("  Buffer location:........ %p\n", (void *) p->v);
```

```
301                     printf("----------------------------------------\n");
302             }
303
304             if (err)
305                     SOURCE_Del(p);
306
307             *errcode = err;
308             return p;
309     }
310
311     /*
312     DWORD SOURCE_Run(BBCFTP *sys)
313     {
314             // Load another block of data from the file if possible.
315             while ( (sys->source->fp) && (sys->source->input_fifo_bytes <= sys->config->
                    file_block_size) )
316             {
317                     bytes_read = fread(sys->source->input_fifo + sys->source->fifo_write, 1, sys->
                            config->file_block_size, sys->source->fp);
318                     sys->source->input_fifo_bytes += bytes_read;
319                     sys->source->input_fifo_write = (sys->source->input_fifo_write + bytes_read) & (
                            sys->config->input_fifo_mask);
320                     if (bytes_read < sys->config->file_block_size)
321                     {
322                             fclose(sys->source->fp);
323                             sys->source->fp = NULL;
324                     }
325             }
326
327             // Process as much data from input FIFO to output FIFO as possible
328             if( (sys->source->input_fifo_bytes > sys->source->input_chunk_size) && (sys->source->
                    output_fifo_items < sys->source->output_fifo_size) )
329             {
330                     // Process a chunk of data
331                     if (sys->config->scheduler_TX_notRX)
332                     {
333                             // Prepare a message for encoding
334                     }
335                     else
336                     {
337                             // Transfer raw USRP data for demodulation
338                     }
339
340                     sys->source->input_fifo_bytes -= sys->source->input_chunk_size;
```

```
341                 sys->source->input_fifo_read = (sys->source->input_fifo_bytes + sys->source->
                        input_chunk_size) & (sys->config->input_fifo_mask);
342
343                 sys->source->output_fifo_items++;
344                 sys->source->output_fifo_write = (sys->source->output_fifo_write + sys->source->
                        output_chunk_size) & (sys->config->output_fifo_mask);
345         }
346
347         // Determine if source can no longer stream data to its successor
348         if (!sys->source->fp)
349                 if (sys->source->input_fifo_bytes < sys->source->input_chunk_size)
350                         if (0 == sys->source->output_fifo_items)
351                                 sys->source->streaming = FALSE;
352
353         return 0;
354 }
355 */
356
357 //---------------------------------------------------------------------------
```

```
1  /*****************************************************************************
2   * Signal Source Module for the Real-time BBC Codec/Modem                    *
3   *****************************************************************************
4   * Modified By:
5   * Mark G. Kuhr
6   * Auburn University
7   *
8   * Original Author:
9   * William L. Bahn                                                           *
10  * Academy Center for Information Security                                   *
11  * Department of Computer Science                                            *
12  * United States Air Force Academy                                           *
13  * USAFA, CO 80840                                                           *
14  *****************************************************************************
15  *
16  * DESCRIPTION
17  *
18  * This module supports the signal source for both the TX and the RX
19  *
20  */
21
22 #ifndef SOURCEdotH
23 #define SOURCEdotH
24
25 //---------------------------------------------------------------------------
```

```
26  // REQUIRED INCLUDES
27  //---------------------------------------------------------------------------
28
29  #include "config.h"
30  #include "dirtyd.h"
31
32  //---------------------------------------------------------------------------
33  // STRUCTURE DECLARATIONS
34  //---------------------------------------------------------------------------
35
36  typedef struct SOURCE SOURCE;
37
38  //---------------------------------------------------------------------------
39  // STRUCTURE DEFINITIONS
40  //---------------------------------------------------------------------------
41
42  // NOTE: Normally the structure definition would be in the *.c file to make
43  // the structure members inaccessible to outside functions except through
44  // public function calls. But for the real-time code it has been decided
45  // to make the structure members directly visible to the functions that
46  // manipulate them.
47
48  struct SOURCE
49  {
50          int    streaming;          // Buffer active flag
51          DWORD sample;              // Number of samples that have been processed
52          DWORD samples;             // Number of samples in buffer
53          DWORD sample_size_bytes;   // Bytes required per sample
54          DWORD sample_limit;        // Number of samples space is allocated for
55          BYTE *v;                   // Buffer address
56
57          DWORD file_bytes;          // File size based on seek test
58          size_t chunk_size;         // File bytes bytes per message
59          size_t buffer_size;        // Size of allocated source buffer
60  };
61
62  //---------------------------------------------------------------------------
63  // PUBLIC FUNCTION PROTOTYPES
64  //---------------------------------------------------------------------------
65
66  SOURCE *SOURCE_Del(SOURCE *p);
67  SOURCE *SOURCE_New(CONFIG *c, DWORD *errcode);
68
69  //---------------------------------------------------------------------------
70  #endif
```

C.2.3 sink.c and sink.h

```
/*****************************************************************************
 * Signal Sink Module for the Real-time BBC Codec/Modem                      *
 *****************************************************************************
 * Modified By:
 * Mark G. Kuhr
 * Auburn University
 *
 * Original Author:
 * William L. Bahn                                                           *
 * Academy Center for Information Security                                   *
 * Department of Computer Science                                            *
 * United States Air Force Academy                                           *
 * USAFA, CO 80840                                                           *
 *****************************************************************************
 *
 * DESCRIPTION
 *
 * This module supports the signal sink for both the TX and the RX
 *
 */

//---------------------------------------------------------------------------
// REQUIRED INCLUDES
//---------------------------------------------------------------------------

#include <stdlib.h> // malloc(), free()
#include <string.h> // memmove()

#include "sink.h"
#include "bbcftp.h"
#include "dirtyd.h"

//---------------------------------------------------------------------------
// STRUCTURE DEFINITIONS
//---------------------------------------------------------------------------

// NOTE: Normally the structure definition would be in the *.c file to make
// the structure members inaccessible to outside functions except through
// public function calls. But for the real-time code it has been decided
// to make the structure members directly visible to the functions that
// manipulate them.

//---------------------------------------------------------------------------
```

```
44  // PUBLIC FUNCTION DEFINITIONS
45  //---------------------------------------------------------------------------
46
47  SINK *SINK_Del(SINK *p)
48  {
49          if (p)
50          {
51                  if (p->fp)
52                          if (stdout != p->fp)
53                          {
54                                  fclose(p->fp);
55                                  p->fp = NULL;
56                          }
57                  if (p->v)     { free(p->v);   p->v = NULL;       }
58                  free(p);
59                  p = NULL;
60          }
61
62          return p;
63  }
64
65  // Sufficient memory is allocated up front
66  // to handle a maximum amount of data. However, the present
67  // contents of the buffer can be purged using SINK_Purge().
68
69  SINK *SINK_New(CONFIG *c, DWORD *errcode)
70  {
71          SINK *p;
72          DWORD err;
73
74          p = NULL;
75          err = 0;
76
77          p = (SINK *) malloc(sizeof(SINK));
78          if (!p)
79                  err |= 1 << 0;
80
81          // Open Data Sink file
82          if (!err)
83          {
84                  p->fp = NULL;
85                  if (c->sink_name)
86                  {
87                          p->fp = fopen(c->sink_name, "wb");
88                          if (!p->fp)
```

```
89                                      err |= 1 << 7;
90                      }
91                      else
92                              p->fp = stdout;
93              }
94
95              // Initialize state information
96              if (!err)
97              {
98                      p->samples = 0;
99                      p->streaming = TRUE;
100
101                     if (c->sink_sample_limit)
102                             p->sample_limit = c->sink_sample_limit;
103                     else
104                     {
105                             if (c->scheduler_TX_notRX)
106                             {
107                                     p->sample_limit = 4*c->modem_samples_per_bit*c->packet_bits;
108                             }
109                             else
110                             {
111                                     p->sample_limit = 1000;
112                             }
113                     }
114
115                     if (c->sink_sample_size_bytes)
116                             p->sample_size_bytes = c->sink_sample_size_bytes;
117                     else
118                     {
119                             if (c->scheduler_TX_notRX)
120                             {
121                                     p->sample_size_bytes = 2*sizeof(float);
122                             }
123                             else
124                             {
125                                     // One byte for each eight full bits of message
126                                     p->sample_size_bytes = c->codec_message_bits / 8;
127                                     // Add a final byte, if necessary, to hold leftover bits
128                                     if (c->codec_message_bits % 8)
129                                             p->sample_size_bytes++;
130                                     // Add one byte for terminating NUL character
131                                     p->sample_size_bytes++;
132                             }
133                     }
```

```
134
135             }
136
137             // Allocate Memory for sink data
138             if (!err)
139             {
140                     p->buffer_size = p->sample_limit * p->sample_size_bytes;
141                     p->v = malloc(p->buffer_size);
142                     if (!p->v)
143                             err |= 1 << 1;
144             }
145
146             #ifdef DIAGNOSTICS
147             // Diagnostic Report
148             printf("-----------------------------------------\n");
149             printf("SINK\n");
150             printf("  Creation:.............. %s\n", ((err)? "FAILED":"SUCCEEDED"));
151             printf("  Location:.............. %p\n", (void *) p);
152             printf("  Sample size:........... %lu bytes\n", (unsigned long) p->sample_size_bytes);
153             printf("  Sample limit:.......... %lu\n", (unsigned long) p->sample_limit);
154             printf("  Buffer size:........... %lu bytes\n", (unsigned long) p->buffer_size);
155             printf("  Buffer location:....... %p\n", (void *) p->v);
156             printf("-----------------------------------------\n");
157             #endif
158
159             if (err)
160                     SINK_Del(p);
161
162             *errcode = err;
163             return p;
164     }
165
166     void SINK_Purge(CONFIG *c, SINK *p, BUFFER *b)
167     {
168             DWORD i, seq, missing, distinct;
169             BYTE *base;
170             int found, complete;
171             WORD id, stream_id, last_stream_id;
172             char filename[256];
173             int filenamelen;
174             FILE *fp;
175
176             //edit: Mark Kuhr
177             const char delimiter[]="/";
178             char *token;
```

```
179         char *abbrev_file_name;
180         char *log_file_name;
181         //hack around data sharing between python and c
182         FILE *log_file;
183
184         // Transmitter
185         if (c->scheduler_TX_notRX)
186         {
187                 // Leading cushion
188                 for (i = 0; i < c->cushion_bits*c->modem_samples_per_bit; i++)
189                         fwrite(&c->bitptr[0], sizeof(float), 1, p->fp);
190
191                 // Buffer dump
192                 fwrite(p->v, p->sample_size_bytes, p->samples, p->fp);
193
194                 // Trailing cushion
195                 for (i = 0; i < c->cushion_bits*c->modem_samples_per_bit; i++)
196                         fwrite(&c->bitptr[0], sizeof(float), 1, p->fp);
197         }
198         // Receiver
199         else
200         {
201                 if (c->diagnostics)
202                 {
203                         for (i = 0; i < p->samples; i++)
204                         {
205                                 base = p->v + i * p->sample_size_bytes;
206                                 PrintMessage(base);
207                         }
208
209                 }
210
211
212
213                 // The assumption is that there are multiple message streams contained
214                 // in the data. So as to operate in fixed-memory, the message streams
215                 // are processed one at a time, starting with the lowest ID. This is
216                 // not an approach that is very consistent with the notion of a streaming
217                 // real-time system, but it is a start.
218
219                 // Stream ID's of zero will be ignored. They are used to push messages
220                 // that the decoder must receive and process and are assumed to be
221                 // discriminated against at the decoder level.
222
223                 stream_id = 0;
```

```
224                 fp = NULL;
225                 log_file = NULL; //file for logging of information to be sent to python
226                 do
227                 {
228                         // Find next larger sequence ID that has a sequence number of zero.
229                         last_stream_id = stream_id;
230                         for (i = 0; i < p->samples; i++)
231                         {
232                                 base = p->v + i * p->sample_size_bytes;
233                                 if (0 == GetMessageSeq(base))
234                                 {
235                                         id = GetMessageID(base);
236                                         if (id > last_stream_id)
237                                                 if ((id < stream_id)||(stream_id == last_stream_id
                                                        ))
238                                                     stream_id = id;
239                                 }
240                         }
241
242                         // Process the next stream (if one was found)
243                         if (stream_id > last_stream_id)
244                         {
245                                 if (c->diagnostics)
246                                 {
247                                         printf("Stream ID: %lu.\n", (unsigned int) stream_id);
248
249                                 }
250                                 missing = 0;
251                                 distinct = 0;
252                                 complete = FALSE;
253                                 for (seq = 0; (!complete) && (seq < p->samples); seq++)
254                                 {
255                                         found = FALSE;
256                                         for (i = 0; (!found) && (i < p->samples); i++)
257                                         {
258                                                 base = p->v + i * p->sample_size_bytes;
259                                                 if ( (seq == GetMessageSeq(base))&&(stream_id ==
                                                        GetMessageID(base)) )
260                                                 {
261                                                     found = TRUE;
262                                                     distinct++;
263                                                 }
264                                         }
265                                         if (found)
266                                         {
```

```
267                     // Extract file name from header message and open
                            file
268                     // This is the file that is received
269                     if (0 == seq)
270                     {
271                             filenamelen = GetMessageLoadBits(base)/8;
272                             //printf("filenamelen: %d",filenamelen);
273                             if (filenamelen < 255)
274                             {
275
276                                     memmove(filename,
                                            GetMessagePayload(base),
                                            filenamelen);
277                                     filename[filenamelen] = NULL;
278
279                                     //modification: fix for full
                                            length path in wxPython
280                                     //check for recipient, change path
                                             accordingly for sink file
281                                     token = strtok(filename,delimiter)
                                            ;
282                                     while(token!=NULL)
283                                     {
284                                             abbrev_file_name = token;
285                                             token = strtok(NULL,
                                                    delimiter);
286                                     }
287
288                                     printf("Experiment Path: %s\n",c->
                                            path);
289                                     printf("Node ID: %s\n",c->node_id)
                                            ;
290                                     printf("Node Address: %lu\n",(
                                            unsigned int) c->node_address)
                                            ;
291                                     printf("To Address: %lu\n", (
                                            unsigned int) GetMessageToAddr
                                            (base));
292                                     printf("From Address: %lu\n", (
                                            unsigned int)
                                            GetMessageFromAddr(base));
293                                     printf("RSSI: %lu\n", (unsigned
                                            int) GetMessageRSSI(base));
294                                     strcat(c->path,c->node_id);
295                                     strcat(c->path,"/");
```

```
296
297                                     log_file_name = strdup(c->path);
298                                     strcat(log_file_name,"logrx.txt");
299                                     printf("Log Filename is: %s\n",
                                            log_file_name);
300                                     strcat(c->path,abbrev_file_name);
301                                     printf("Filename is: %s\n", c->
                                            path);
302                                     strcpy(filename,c->path);
303                                     fp = fopen(filename, "wb");
304
305                                     log_file = fopen(log_file_name,"wb
                                            ");
306                                     if(log_file)
307                                     {
308                                                     //write values to
                                                            file
309                                             //fwrite(&c->marks,1,32,
                                                    log_file);
310                                             fprintf(log_file,"%li\n",
                                                    c->marks);//Marks
311                                             fprintf(log_file,"%lu\n",
                                                    c->message_count);//
                                                    Message Count
312                                             fprintf(log_file,"%li\n",(
                                                    DWORD)(b->overflows*8)
                                                    );//Packets Lost
313                                             fprintf(log_file,"%s\n", c
                                                    ->node_id);//node_id (
                                                    name of current node)
314                                             fprintf(log_file,"%lu\n",
                                                    (unsigned int) c->
                                                    node_address);//
                                                    address of current
                                                    node
315                                             fprintf(log_file,"%lu\n",
                                                    (unsigned int)
                                                    GetMessageToAddr(base)
                                                    ); //packet to address
316                                             fprintf(log_file,"%lu\n",
                                                    (unsigned int)
                                                    GetMessageFromAddr(
                                                    base)); //packet from
                                                    address
```

```
317                                                     fprintf(log_file,"%s\n",
                                                            abbrev_file_name);//
                                                            transmitted file or
                                                            packet
318                                                     fprintf(log_file,"%lu\n",
                                                            (unsigned int)
                                                            GetMessageRSSI(base));
                                                             //packet rssi value
319                                             }
320
321                                             /*printf("Filename is: %s\n",
                                                    abbrev_file_name);
322                                             fp = fopen(abbrev_file_name, "wb")
                                                    ;*/
323                                     }
324                             }
325                             // Process non-header messages
326                             else
327                             {
328                                     // Check for terminal message
329                                     if (0 == GetMessageLoadBits(base))
330                                             complete = TRUE;
331                                     // Transfer next data fragment to file
332                                     else
333                                             if (fp)
334                                                     fwrite(GetMessagePayload(
                                                            base), 1,
                                                            GetMessageLoadBits(
                                                            base)/8, fp);
335                             }
336                     }
337                     else
338                     {
339                             if (c->diagnostics)
340                                     printf("*** Missing Sequence #: %lu\n", (
                                            unsigned int) seq);
341                             missing++;
342                     }
343             }
344
345             if (c->diagnostics)
346             {
347                     if (!complete)
348                             printf("Terminal message not found.\n");
```

```
349                                     printf("Total Missing Sequences: %lu\n", (unsigned int)
                                            missing);
350                                     printf("Total Distinct Messages: %lu\n", (unsigned int)
                                            distinct);
351                             }
352                     }
353                     if (fp)
354                     {
355                             fclose(fp);
356                             fp = NULL;
357                     }
358
359             } while (stream_id > last_stream_id);
360     }
361     p->samples = 0;
362 }
```

```
1  /*************************************************************************
2   * Signal Sink Module for the Real-time BBC Codec/Modem                  *
3   *************************************************************************
4   * Modified By:
5   * Mark G. Kuhr
6   * Auburn University
7   *
8   * Original Author:
9   * William L. Bahn                                                      *
10  * Academy Center for Information Security                              *
11  * Department of Computer Science                                       *
12  * United States Air Force Academy                                      *
13  * USAFA, CO 80840                                                      *
14  *************************************************************************
15  *
16  * DESCRIPTION
17  *
18  * This module supports the signal sink for both the TX and the RX
19  *
20  */
21
22 #ifndef SINKdotH
23 #define SINKdotH
24
25 //-------------------------------------------------------------------------
26 // REQUIRED INCLUDES
27 //-------------------------------------------------------------------------
28
```

```
29 #include "config.h"
30 #include "buffer.h"
31 #include "dirtyd.h"
32
33 //---------------------------------------------------------------------------
34 // STRUCTURE DECLARATIONS
35 //---------------------------------------------------------------------------
36
37 typedef struct SINK SINK;
38
39 //---------------------------------------------------------------------------
40 // STRUCTURE DEFINITIONS
41 //---------------------------------------------------------------------------
42
43 // NOTE: Normally the structure definition would be in the *.c file to make
44 // the structure members inaccessible to outside functions except through
45 // public function calls. But for the real-time code it has been decided
46 // to make the structure members directly visible to the functions that
47 // manipulate them.
48
49 struct SINK
50 {
51         FILE *fp;
52         int    streaming;
53         DWORD samples;
54         DWORD sample_size_bytes;
55         DWORD sample_limit;
56         BYTE *v;
57
58         size_t buffer_size;
59 };
60
61 //---------------------------------------------------------------------------
62 // PUBLIC FUNCTION PROTOTYPES
63 //---------------------------------------------------------------------------
64
65 SINK *SINK_Del(SINK *p);
66 SINK *SINK_New(CONFIG *config, DWORD *errcode);
67 void SINK_Purge(CONFIG *config, SINK *p, BUFFER *b);
68
69 //---------------------------------------------------------------------------
70 #endif
```

C.2.4 config.c and config.h

```
1  /*****************************************************************************
2   * Configuration Module for the Real-time BBC Codec/Modem                    *
3   *****************************************************************************
4   * Modified By:
5   * Mark G. Kuhr
6   * Auburn University
7   *
8   * Original Author:
9   * William L. Bahn                                                           *
10  * Academy Center for Information Security                                   *
11  * Department of Computer Science                                            *
12  * United States Air Force Academy                                           *
13  * USAFA, CO 80840                                                           *
14  *****************************************************************************
15  * DESCRIPTION
16  *
17  * This module imports and manages the configuration information for the
18  * modem and the codec.
19  *
20  */
21
22  //---------------------------------------------------------------------------
23  // REQUIRED INCLUDES
24  //---------------------------------------------------------------------------
25  #include <stdlib.h> // malloc(), free()
26  #include <math.h>
27  #include <string.h>
28  #include <ctype.h>
29
30  #include "config.h"
31  #include "dirtyd.h"
32
33  #define USRP_SAMPLE_SIZE (2*sizeof(float))
34
35  //---------------------------------------------------------------------------
36  // STRUCTURE DEFINITIONS
37  //---------------------------------------------------------------------------
38
39  // NOTE: Normally the structure definition would be in the *.c file to make
40  // the structure members inaccessible to outside functions except through
41  // public function calls. But for the real-time code it has been decided
42  // to make the structure members directly visible to the functions that
43  // manipulate them.
44
45  //---------------------------------------------------------------------------
```

```
46  // PRIVATE FUNCTION DEFINITIONS
47  //-------------------------------------------------------------------------
48
49  // Nominal String: xxx"filename"xxxx
50  // If both double quotes are not found, a NULL pointer is returned.
51
52  char *ExtractName(char *s)
53  {
54          char *filename;
55          char *t;
56          int len;
57
58          filename = NULL;
59
60          // Advance s to first double quote or end of string
61          while ((*s)&&('\"' != *s))
62                  s++;
63          // If double quote found, advance to next character
64          if (*s)
65                  s++;
66          // Advance t to next double quote or end of string
67          for (t = s; (*t)&&('\"' != *t); t++)
68                  EMPTYLOOP;
69
70          // Calculate length of string between first pair of double quotes
71          len = t - s;
72
73          t = filename = malloc(len + 1);
74
75          if (filename)
76          {
77                  while (len--)
78                          *t++ = *s++;
79                  *t = '\0';
80          }
81
82          return filename;
83  }
84
85  // NOTE: The character string may me changed by this function.
86
87  void UpdateConfig(CONFIG *c, char *string)
88  {
89          char *s, *v, *t;
90          DWORD vi;
```

```
91          double vf;
92          if ((!c)||(!string))
93                  return;
94
95          // Advance into string to first non-whitespace character
96          for (s = string; isspace(*s); s++)
97                  EMPTYLOOP;
98
99          // Ignore blank or comment lines
100         if ((NUL == *s)||('#' == *s))
101                 return;
102
103         // Identify parameter keyword and convert to uppercase
104         for (v = s; (*v) && (!((isspace(*v)) || (':' == *v) || ('=' == *v))); v++)
105                 *v = toupper(*v);
106         // Terminate keyword and start value immediately after (if anything there)
107         if (*v)
108                 *v++ = NUL;
109
110         // Skip over whitespace, colons, and equal signs.
111         while ( (isspace(*v)) || (':' == *v) || ('=' == *v) )
112                 v++;
113
114         // Process those parameters that use string values
115         if (!strcmp(s, "SOURCE_NAME"))          c->source_name = ExtractName(v);
116         if (!strcmp(s, "SINK_NAME"))            c->sink_name = ExtractName(v);
117         //additional parameters for identification of nodes
118         if (!strcmp(string,"NODE_ID"))              c->node_id = ExtractName(v);
119
120         if (!strcmp(s,"PATH"))                                  c->path = ExtractName(v);
121
122         // Process remaining parameters
123
124         // Extract value from string
125         vi = atoi(v);
126         vf = atof(v);
127
128         //ADDRESSING
129         if(!strcmp(s,"TO_ADDRESS"))                             c->to_address = vi;
130         if(!strcmp(s,"FROM_ADDRESS"))                   c->from_address = vi;
131         if(!strcmp(s,"NODE_ADDRESS"))                   c->node_address = vi;
132         if(!strcmp(s,"RSSI"))                                   c->rssi = vi;
133
134         // SCHEDULER Configuration
135         if (!strcmp(s, "DIAGNOSTICS"))          c->diagnostics = TRUE;
```

```
136         if (!strcmp(s, "SCHEDULER_TX_NOTRX"))     c->scheduler_TX_notRX = vi;
137         if (!strcmp(s, "SCHEDULER_REALTIME"))     c->scheduler_realtime = vi;
138
139         // SOURCE Configuration
140         if (!strcmp(s, "SOURCE_ID"))              c->source_id = vi;
141         // SOURCE_NAME processed above due to string value
142
143         // CODEC Configuration
144         if (!strcmp(s, "CODEC_MESSAGE_BITS"))     c->codec_message_bits = vi;
145         if (!strcmp(s, "CODEC_RANDOM_BITS"))      c->codec_random_bits = vi;
146         if (!strcmp(s, "CODEC_CLAMP_BITS"))       c->codec_clamp_bits = vi;
147         if (!strcmp(s, "CODEC_FRAGMENT_BITS"))    c->codec_fragment_bits = vi;
148         if (!strcmp(s, "CODEC_STOP_BITS"))        c->codec_stop_bits = vi;
149         if (!strcmp(s, "CODEC_EXPANSION"))        c->codec_expansion = vi;
150         if (!strcmp(s, "CODEC_PACKET_LOAD"))      c->codec_packet_load = vi;
151         if (!strcmp(s, "CODEC_DECODE_LIMIT"))     c->codec_decode_limit = vi;
152
153         // BUFFER Configuration
154         if (!strcmp(s, "BUFFER_PACKETS"))         c->buffer_packets = vi;
155         if (!strcmp(s, "BUFFER_LAMBDA"))          c->buffer_lambda = vf;
156
157         // MODEM Configuration
158         if (!strcmp(s, "MODEM_PACKET_RATE_BPS")) c->modem_packet_rate_bps = vi;
159         if (!strcmp(s, "MODEM_SAMPLES_PER_BIT")) c->modem_samples_per_bit = vi;
160         if (!strcmp(s, "MODEM_GAIN_DB"))          c->modem_gain_dB = vf;
161         if (!strcmp(s, "MODEM_CHANNEL_LOSS_DB")) c->modem_channel_loss_dB = vf;
162         if (!strcmp(s, "MODEM_THRESHOLD_PCT"))    c->modem_threshold_pct = vf;
163         if (!strcmp(s, "MODEM_HYSTERESIS_PCT"))   c->modem_hysteresis_pct = vf;
164         if (!strcmp(s, "MODEM_JITTER_BITS"))      c->modem_jitter_bits = vf;
165         if (!strcmp(s, "MODEM_CUSHION_PCT"))      c->modem_cushion_pct = vf;
166
167         // SINK Configuration
168         // SOURCE_FILE_NAME processed above due to string value
169         if (!strcmp(s, "SINK_SAMPLE_LIMIT"))      c->sink_sample_limit = vi;
170 }
171
172 //---------------------------------------------------------------------------
173 // PUBLIC FUNCTION DEFINITIONS
174 //---------------------------------------------------------------------------
175
176 CONFIG *CONFIG_Del(CONFIG *p)
177 {
178         if (p)
179         {
180                 if (p->source_name)
```

```
181                     {
182                             free(p->source_name);
183                             p->source_name = NULL;
184                     }
185                     if (p->sink_name)
186                     {
187                             free(p->sink_name);
188                             p->sink_name = NULL;
189                     }
190                     if (p->node_id)
191                     {
192                             free(p->node_id);
193                             p->node_id = NULL;
194                     }
195                     if (p->path)
196                     {
197                             free(p->path);
198                             p->path = NULL;
199                     }
200                     free(p);
201                     p = NULL;
202             }
203
204             return p;
205     }
206
207     CONFIG *CONFIG_New(char *filename, DWORD *errcode)
208     {
209             CONFIG *p;
210             FILE *fp;
211             DWORD err;
212             int i;
213             char *s;
214             char *derived_path_source;
215             char *derived_path_sink;
216
217             derived_path_source=NULL;
218             derived_path_sink=NULL;
219
220             p = NULL;
221             err = 0;
222             s = NULL;
223
224             p = (CONFIG *) malloc(sizeof(CONFIG));
225             if (!p)
```

```
226                 err |= 1 << 0;
227
228         if (!err)
229         {
230                 //--------------------------------------------------------------------
231                 // NOTE: Establish default values and then overwrite with file data
232                 //--------------------------------------------------------------------
233
234                 p->diagnostics = FALSE;
235
236                 // Direction
237                 p->scheduler_TX_notRX = TRUE;
238                 p->scheduler_realtime = FALSE;
239
240                 //addressing
241                 p->to_address = 0;
242                 p->from_address = 0;
243                 p->node_address = 0;
244
245                 //noise parameter
246                 p->rssi = 0;
247
248                 //path parameter
249                 p->path = NULL;
250
251                 // Identification Parameters
252                 p->node_id = NULL;
253
254                 // Source Parameters
255                 p->source_name = NULL;
256                 p->source_id = 0;
257
258                 // Codec Parameters
259                 p->codec_message_bits = 512;
260                 p->codec_random_bits = 0;
261                 p->codec_clamp_bits = 1;
262                 p->codec_fragment_bits = 1;
263                 p->codec_stop_bits = 100;
264                 p->codec_expansion = 100;
265                 p->codec_packet_load = 5;
266                 p->codec_decode_limit = 100;
267
268                 // Buffer Parameters
269                 p->buffer_packets = 2.0;
270                 p->buffer_lambda = 1.0;
```

```
271
272                 // Modem Parameters
273                 p->modem_packet_rate_bps = 500000;
274                 p->modem_cushion_pct = 10.0;
275                 p->modem_samples_per_bit = 4;
276                 p->modem_threshold_pct = 46.3744;
277                 p->modem_hysteresis_pct = 5.0;
278                 p->modem_gain_dB = 80.0;
279                 p->modem_channel_loss_dB = 3.0;
280                 p->modem_jitter_bits = 3.0;
281
282                 // Sink Parameters
283                 p->sink_name = NULL;
284                 p->sink_sample_limit = 0;
285                 p->sink_sample_size_bytes = 0;
286
287                 //------------------------------------------------------------------
288                 // Update values from configuration file
289                 //------------------------------------------------------------------
290
291                 if (filename)
292                 {
293                         fp = fopen(filename, "rt");
294                         if (fp)
295                         {
296                                 while (!feof(fp))
297                                 {
298                                         s = fdgets(fp);
299                                         UpdateConfig(p, s);
300                                         if (s)
301                                                 free(s);
302                                         s = NULL;
303                                 }
304                                 fclose(fp);
305                         }
306                         else
307                                 err |= 1 << 1;
308                 }
309
310                 //------------------------------------------------------------------
311                 // Calculate derived parameters
312                 //------------------------------------------------------------------
313
314                 //derive the source name and sink name
315
```

```
316                 derived_path_source = strdup(p->path);
317                 strcat(derived_path_source,p->node_id);
318                 strcat(derived_path_source,"/");
319                 strcat(derived_path_source,p->source_name);
320
321                 p->source_name = strdup(derived_path_source);
322                 //printf("path source: %s\n",p->source_name);
323
324                 derived_path_sink = strdup(p->path);
325                 strcat(derived_path_sink,p->node_id);
326                 strcat(derived_path_sink,"/");
327                 strcat(derived_path_sink,p->sink_name);
328
329                 p->sink_name = strdup(derived_path_sink);
330                 //printf("path sink: %s\n",p->sink_name);
331
332                 // bitmasks to mask bits within a byte.
333                 for (i = 0; i < 8; i++)
334                         p->bitmask[i] = ((BYTE) 1) << i;
335
336                 // Set USRP sample size to two floats (complex IQ)
337                 if (p->scheduler_TX_notRX)
338                         p->sink_sample_size_bytes = USRP_SAMPLE_SIZE;
339                 else
340                         p->source_sample_size_bytes = USRP_SAMPLE_SIZE;
341
342                 // Set sink sample limit
343                 if (!p->sink_sample_limit)
344                 {
345                         if (p->scheduler_TX_notRX)
346                                 p->sink_sample_limit = 2000000;
347                         else
348                                 p->sink_sample_limit = 1000;
349                 }
350
351                 // Set source filename to default if not set by config file
352                 if (!p->source_name)
353                 {
354                         if (p->scheduler_TX_notRX)
355                         {
356                                 p->source_name = malloc(strlen("usrp.txd")+1);
357                                 if (p->source_name)
358                                         strcpy(p->source_name, "usrp.txd");
359                         }
360                         else
```

```
361                     {
362                             p->source_name = malloc(strlen("usrp.srp")+1);
363                             if (p->source_name)
364                                     strcpy(p->source_name, "usrp.srp");
365                     }
366             }
367
368             // Set sink filename to default if not set by config file
369             if (!p->sink_name)
370             {
371                     if (p->scheduler_TX_notRX)
372                     {
373                             // Sink Parameters
374                             p->sink_name = malloc(strlen("usrp.srp")+1);
375                             if (p->sink_name)
376                                     strcpy(p->sink_name, "usrp.srp");
377                     }
378                     else
379                     {
380                             // Sink Parameters
381                             p->sink_name = malloc(strlen("usrp.rxd")+1);
382                             if (p->sink_name)
383                                     strcpy(p->sink_name, "usrp.rxd");
384                     }
385             }
386
387             // Calculate and store derived quantities
388             p->bytes_per_message = p->codec_message_bits/8;
389             if (p->bytes_per_message % 8)
390                     p->bytes_per_message++;
391             p->packet_bits = (p->codec_message_bits * p->codec_expansion);
392             p->last_packet_bit = p->packet_bits - 1;
393
394             p->bytes_per_packet = p->packet_bits/8;
395             if (p->bytes_per_packet % 8)
396                     p->bytes_per_packet++;
397             p->bufferbytes_per_packet = p->bytes_per_packet + 1;
398             p->buffer_advance = (DWORD) (p->bytes_per_packet * p->buffer_lambda);
399
400             p->cushion_bits = (DWORD) (p->packet_bits * p->modem_cushion_pct / 100.0);
401
402             p->nominal_tx_signal =  pow(10.0, (p->modem_gain_dB)/20.0);
403             p->nominal_rx_signal =  pow(10.0, (p->modem_gain_dB - p->modem_channel_loss_dB)
                    /20.0);
404
```

```
405                 // Compute storage requirments for BBC decode tree
406                 if ((0 == p->codec_clamp_bits)||(0 == p->codec_fragment_bits))
407                 {
408                         p->codec_clamp_bits = 0;
409                         p->codec_fragment_bits = p->codec_random_bits + p->codec_message_bits;
410                 }
411                 p->fragments = (p->codec_random_bits + p->codec_message_bits)/p->
                        codec_fragment_bits;
412                 p->padded_message_bits = p->fragments * (p->codec_clamp_bits + p->
                        codec_fragment_bits);
413                 if ((p->codec_random_bits + p->codec_message_bits) % p->codec_fragment_bits)
414                 {
415                         p->padded_message_bits += p->codec_clamp_bits;
416                         p->padded_message_bits += (p->codec_random_bits + p->codec_message_bits)%p
                                ->codec_fragment_bits;
417                 }
418                 p->padded_message_bits += p->codec_stop_bits;
419
420                 //Lookup tables
421                 // 0 and 1 represented as BYTEs that can be passed by reference
422                 p->bitptr[0] = 0;
423                 p->bitptr[1] = 1;
424
425                 // Tally counters;
426                 p->message_count = 0;
427
428                 // State information
429                 p->marks = 0;
430
431                 // Tally Counters
432                 p->actual_trx_bytes = 0;
433                 p->nominal_trx_bytes = 0;
434                 p->dem_ticks = 0;
435                 p->dec_ticks = 0;
436                 p->ticks = 0;
437                 p->tot_ticks = 0;
438
439                 p->trx_bytes_per_packet_byte = 8 * p->modem_samples_per_bit * USRP_SAMPLE_SIZE;
440                 p->bytespertick = (   p->modem_packet_rate_bps
441                                         * p->modem_samples_per_bit
442                                                 * USRP_SAMPLE_SIZE
443                                                ) / ((double)CLOCKS_PER_SEC);
444         }
445
446         if (err)
```

```
447                     p = CONFIG_Del(p);
448
449             *errcode = err;
450             return p;
451     }
452
453     //---------------------------------------------------------------------------
```

```
1   /*****************************************************************************
2    * Configuration Module for the Real-time BBC Codec/Modem                    *
3    *****************************************************************************
4    * Modified By:
5    * Mark G. Kuhr
6    * Auburn University
7    *
8    * Original Author:
9    * William L. Bahn                                                           *
10   * Academy Center for Information Security                                   *
11   * Department of Computer Science                                            *
12   * United States Air Force Academy                                           *
13   * USAFA, CO 80840                                                           *
14   *****************************************************************************
15   *
16   * DESCRIPTION
17   *
18   * This module imports and manages the configuration information for the
19   * modem and the codec.
20   *
21   */
22
23  #ifndef CONFIGdotH
24  #define CONFIGdotH
25
26   //---------------------------------------------------------------------------
27   // REQUIRED INCLUDES
28   //---------------------------------------------------------------------------
29
30  #include <time.h>
31
32  #include "dirtyd.h"
33
34   //---------------------------------------------------------------------------
35   // STRUCTURE DECLARATIONS
36   //---------------------------------------------------------------------------
37
```

```
38  typedef struct CONFIG CONFIG;
39
40  //--------------------------------------------------------------------------
41  // STRUCTURE DEFINITIONS
42  //--------------------------------------------------------------------------
43
44  // NOTE: Normally the structure definition would be in the *.c file to make
45  // the structure members inaccessible to outside functions except through
46  // public function calls. But for the real-time code it has been decided
47  // to make the structure members directly visible to the functions that
48  // manipulate them.
49
50  struct CONFIG
51  {
52          int diagnostics;
53
54          // Direction
55          int scheduler_TX_notRX;
56          int scheduler_realtime;
57
58          //modification: Mark Kuhr
59
60          //Addressing parameters
61          WORD to_address;
62          WORD from_address;
63          WORD node_address;
64
65          //noise parameter
66          WORD rssi;
67
68          //parameters for identification of nodes
69          char *node_id;
70          char *path;
71
72          // Source Parameters
73          char *source_name;
74          DWORD source_sample_size_bytes;
75          DWORD source_sample_limit;
76          WORD  source_id;
77
78          // Codec Parameters
79          DWORD codec_message_bits;
80          DWORD codec_random_bits;
81          DWORD codec_clamp_bits;
82          DWORD codec_fragment_bits;
```

```
83          DWORD codec_stop_bits;
84          DWORD codec_expansion;
85          DWORD codec_decode_limit;
86          DWORD codec_packet_load;
87
88          // Derived Codec Parameters
89          DWORD fragments;      // Number of complete fragement in padded message
90          DWORD padded_message_bits;      // Length of message after padding with random and check
                bits
91          DWORD packet_bits;
92          DWORD last_packet_bit;
93          DWORD bytes_per_message;
94          DWORD bytes_per_packet;
95          DWORD bufferbytes_per_packet;
96
97          // Buffer Parameters
98          double buffer_packets;
99          double buffer_lambda;
100         DWORD  buffer_advance;
101
102         // Modem Parameters
103         DWORD  modem_packet_rate_bps;
104         DWORD  modem_samples_per_bit;
105         double modem_gain_dB;
106         double modem_channel_loss_dB;
107         double modem_threshold_pct;
108         double modem_hysteresis_pct;
109         double modem_jitter_bits;
110         double modem_cushion_pct;
111         // Derived Modem Parameters
112         //DWORD bytes_per_sample;
113         double nominal_tx_signal;
114         double nominal_rx_signal;
115         DWORD trx_bytes_per_packet_byte;
116         DWORD cushion_bits;
117
118         // Sink Parameters
119         char *sink_name;
120         DWORD sink_sample_size_bytes;
121         DWORD sink_sample_limit;
122
123         // Misc
124         DWORD message_count;
125         DWORD marks;
126
```

```
127         // Lookup tables
128         BYTE  bitptr[2];          // 0 and 1 represented as BYTEs that can be passed by reference
129         BYTE  bitmask[8];  // Masks to pick off the bits within a byte
130
131         // Tally Counters
132         DWORD actual_trx_bytes;
133         DWORD nominal_trx_bytes;
134         double bytespertick;
135         clock_t dem_ticks;
136         clock_t dec_ticks;
137         clock_t ticks;
138         clock_t tot_ticks;
139
140 };
141
142 //---------------------------------------------------------------------------
143 // PUBLIC FUNCTION PROTOTYPES
144 //---------------------------------------------------------------------------
145
146 CONFIG *CONFIG_Del(CONFIG *p);
147 CONFIG *CONFIG_New(char *filename, DWORD *errcode);
148
149 //---------------------------------------------------------------------------
150 #endif
```

C.2.5 bbcftp.c and bbcftp.h

```
1  /*****************************************************************************
2   *****************************************************************************
3   * Modified By:
4   * Mark G. Kuhr
5   * Auburn University
6   *
7   * Original Author:
8   * William L. Bahn                                                           *
9   * Academy Center for Information Security                                   *
10  * Department of Computer Science                                            *
11  * United States Air Force Academy                                           *
12  * USAFA, CO 80840                                                           *
13  *****************************************************************************
14
15  *
16  * DESCRIPTION
17  *
18  * This module provides the crude application layer functions for the ftp
```

```
19   * demo.
20   */
21
22  //---------------------------------------------------------------------------
23  // REQUIRED INCLUDES
24  //---------------------------------------------------------------------------
25
26  #include <stdlib.h> // malloc(), free()
27  #include <stdio.h>  // printf()
28  #include <string.h> // memmove()
29
30  #include "bbcftp.h"
31
32  //---------------------------------------------------------------------------
33  // STRUCTURE DEFINITIONS
34  //---------------------------------------------------------------------------
35
36  // NOTE: Normally the structure definition would be in the *.c file to make
37  // the structure members inaccessible to outside functions except through
38  // public function calls. But for the real-time code it has been decided
39  // to make the structure members directly visible to the functions that
40  // manipulate them.
41
42  //---------------------------------------------------------------------------
43  // PRIVATE FUNCTION DEFINITIONS
44  //---------------------------------------------------------------------------
45
46
47  //---------------------------------------------------------------------------
48  // PUBLIC FUNCTION DEFINITIONS
49  //---------------------------------------------------------------------------
50
51  BBCFTP *BBCFTP_Del(BBCFTP *p)
52  {
53          if (p)
54          {
55                  p->config = CONFIG_Del(p->config);
56                  p->source = SOURCE_Del(p->source);
57                  p->codec  = CODEC_Del(p->codec);
58                  p->buffer = BUFFER_Del(p->buffer);
59                  p->modem  = MODEM_Del(p->modem);
60                  p->sink   = SINK_Del(p->sink);
61          }
62          return NULL;
63  }
```

```
64
65  BBCFTP *BBCFTP_New(char *filename, DWORD *errcode)
66  {
67          BBCFTP *p;
68          DWORD err;
69
70          p = NULL;
71          err = 0;
72          *errcode = 0;
73
74          p = (BBCFTP *) malloc(sizeof(BBCFTP));
75
76          if (!p) *errcode |= 1 << 0;
77
78          if (!*errcode)
79          {
80                  p->config = CONFIG_New(filename, &err);
81                  if (err) *errcode |= 1 << 1;
82          }
83
84          if (!*errcode)
85          {
86                  p->source = SOURCE_New(p->config, &err);
87                  if (err) *errcode |= 1 << 2;
88                  p->codec  = CODEC_New(p->config, &err);
89                  if (err) *errcode |= 1 << 3;
90                  p->buffer = BUFFER_New(p->config, &err);
91                  if (err) *errcode |= 1 << 4;
92                  p->modem  = MODEM_New(p->config, &err);
93                  if (err) *errcode |= 1 << 5;
94                  p->sink   = SINK_New(p->config, &err);
95                  if (err) *errcode |= 1 << 6;
96          }
97
98          return p;
99  }
100
101 void BBCFTP_ErrorCodes(DWORD err)
102 {
103         if ( err & ((DWORD) 1 << 0) )
104                 printf("BBC-FTP System Constructor failed to allocate\n");
105         if ( err & ((DWORD) 1 << 1) )
106                 printf("CONFIG Constructor exited with errors\n");
107         if ( err & ((DWORD) 1 << 2) )
108                 printf("SOURCE Constructor exited with errors\n");
```

```
109         if ( err & ((DWORD) 1 << 3) )
110                 printf("CODEC  Constructor exited with errors\n");
111         if ( err & ((DWORD) 1 << 4) )
112                 printf("BUFFER Constructor exited with errors\n");
113         if ( err & ((DWORD) 1 << 5) )
114                 printf("MODEM  Constructor exited with errors\n");
115         if ( err & ((DWORD) 1 << 6) )
116                 printf("SINK   Constructor exited with errors\n");
117 }
118
119 void PrintMessage(BYTE *base)
120 {
121         int i;
122         int chunk_size_bytes;
123
124         DWORD checksum;
125         WORD  seqnum, loadbits, id, toaddr, fromaddr, rssi;
126
127         checksum = GetMessageChecksum(base);
128         seqnum = GetMessageSeq(base);
129         loadbits = GetMessageLoadBits(base);
130         id = GetMessageID(base);
131         toaddr = GetMessageToAddr(base);
132         fromaddr = GetMessageFromAddr(base);
133         rssi = GetMessageRSSI(base);
134
135         printf("[%04lu] ", (unsigned long) checksum);
136         printf("[%04lu] ", (unsigned long) seqnum);
137         printf("[%04lu] ", (unsigned long) loadbits);
138         printf("[%04lu] ", (unsigned long) id);
139         printf("[%04lu] ", (unsigned long) toaddr);
140         printf("[%04lu] ", (unsigned long) fromaddr);
141         printf("[%04lu] ", (unsigned long) rssi);
142
143
144         chunk_size_bytes = loadbits/8;
145
146         printf("[");
147         for (i = 0; i < chunk_size_bytes; i++)
148         {
149                 putc(*(base + BBC_FTP_OFFSET_PAYLOAD + i), stdout);
150         }
151         printf("]\n");
152
153 }
```

```
154  /*
155  void PrintDistinctMessage(BYTE *base, int *strcount, char *strarray)
156  {
157          int i;
158          int chunk_size_bytes;
159          int seqnum_index=0;
160          char temp[]="";
161          char* line;
162
163          DWORD checksum;
164          WORD  seqnum, loadbits, id;
165
166          checksum = GetMessageChecksum(base);
167          seqnum = GetMessageSeq(base);
168          loadbits = GetMessageLoadBits(base);
169          id = GetMessageID(base);
170
171          printf("[%04lu] ", (unsigned long) checksum);
172          printf("[%04lu] ", (unsigned long) seqnum);
173          printf("[%04lu] ", (unsigned long) loadbits);
174          printf("[%04lu] ", (unsigned long) id);
175
176          chunk_size_bytes = loadbits/8;
177
178
179          //printf("[");
180          for (i = 0; i < chunk_size_bytes; i++)
181          {
182                  //putc(*(base + BBC_FTP_OFFSET_PAYLOAD + i), stdout);
183                  if(seqnum==seqnum_index)
184                  {
185                          printf("after comparing sequence num");
186                          line=strcpy(temp, *(base+BBC_FTP_OFFSET_PAYLOAD + i));
187                          printf("line string: %s\n",line);
188                          strarray = (char **)realloc(strarray,(*strcount+1)*sizeof(char *));
189                          strarray[*strcount++]=strdup(line);
190                      seqnum_index++;
191                  }
192          }
193          //printf("]\n");
194
195  }*/
196
197  void SetMessageRSSI(BYTE *base, WORD v)
198  {
```

```
199             memmove(base+BBC_FTP_OFFSET_RSSI, &v, BBC_FTP_BYTES_RSSI);
200     }
201
202     void SetMessageToAddr(BYTE *base, WORD v)
203     {
204             memmove(base+BBC_FTP_OFFSET_TOADDR, &v, BBC_FTP_BYTES_TOADDR);
205     }
206
207     void SetMessageFromAddr(BYTE *base, WORD v)
208     {
209             memmove(base+BBC_FTP_OFFSET_FROMADDR, &v, BBC_FTP_BYTES_FROMADDR);
210     }
211
212     void SetMessageChecksum(BYTE *base, DWORD v)
213     {
214             memmove(base+BBC_FTP_OFFSET_CHECKSUM, &v, BBC_FTP_BYTES_CHECKSUM);
215     }
216
217     void SetMessageSeq(BYTE *base, WORD v)
218     {
219             memmove(base+BBC_FTP_OFFSET_SEQNUM, &v, BBC_FTP_BYTES_SEQNUM);
220     }
221
222     void SetMessageLoadBits(BYTE *base, WORD v)
223     {
224             memmove(base+BBC_FTP_OFFSET_LOADBITS, &v, BBC_FTP_BYTES_LOADBITS);
225     }
226
227     void SetMessageID(BYTE *base, WORD v)
228     {
229             memmove(base+BBC_FTP_OFFSET_ID, &v, BBC_FTP_BYTES_ID);
230     }
231
232     void SetMessagePayload(BYTE *base, BYTE *source, DWORD bytes, int offset)
233     {
234             memmove(base+BBC_FTP_OFFSET_PAYLOAD+offset, source, bytes);
235     }
236
237     WORD GetMessageRSSI(BYTE *base)
238     {
239             return *((WORD *)(base + BBC_FTP_OFFSET_RSSI));
240     }
241
242
243     WORD GetMessageToAddr(BYTE *base)
```

```
244 {
245         return *((WORD *)(base + BBC_FTP_OFFSET_TOADDR));
246 }
247
248 WORD GetMessageFromAddr(BYTE *base)
249 {
250         return *((WORD *)(base + BBC_FTP_OFFSET_FROMADDR));
251 }
252
253 DWORD GetMessageChecksum(BYTE *base)
254 {
255         return *((DWORD *)(base + BBC_FTP_OFFSET_CHECKSUM));
256 }
257
258 WORD GetMessageSeq(BYTE *base)
259 {
260         return *((WORD *)(base + BBC_FTP_OFFSET_SEQNUM));
261 }
262
263 WORD GetMessageLoadBits(BYTE *base)
264 {
265         return *((WORD *)(base + BBC_FTP_OFFSET_LOADBITS));
266 }
267
268 WORD GetMessageID(BYTE *base)
269 {
270         return *((WORD *)(base + BBC_FTP_OFFSET_ID));
271 }
272
273 BYTE *GetMessagePayload(BYTE *base)
274 {
275         return (BYTE *)(base + BBC_FTP_OFFSET_PAYLOAD);
276 }
277
278 //---------------------------------------------------------------------------
```

```
1 /*****************************************************************************
2  *****************************************************************************
3  * Modified By:
4  * Mark G. Kuhr
5  * Auburn University
6  *
7  * Original Author:
8  * William L. Bahn                                                           *
9  * Academy Center for Information Security                                   *
```

```
10   * Department of Computer Science                                        *
11   * United States Air Force Academy                                       *
12   * USAFA, CO 80840                                                       *
13   ***************************************************************************
14   *
15   * DESCRIPTION
16   *
17   */
18
19  #ifndef BBCFTPdotH
20  #define BBCFTPdotH
21
22  //---------------------------------------------------------------------------
23  // REQUIRED INCLUDES
24  //---------------------------------------------------------------------------
25
26  #include "config.h"
27  #include "source.h"
28  #include "codec.h"
29  #include "buffer.h"
30  #include "modem.h"
31  #include "sink.h"
32
33  #include "dirtyd.h"
34
35  //---------------------------------------------------------------------------
36  // PARAMETER DEFINITIONS
37  //---------------------------------------------------------------------------
38
39  #define BBC_FTP_BYTES_CHECKSUM   (4)
40  #define BBC_FTP_BYTES_SEQNUM     (2)
41  #define BBC_FTP_BYTES_LOADBITS   (2)
42  #define BBC_FTP_BYTES_ID         (2)
43  #define BBC_FTP_BYTES_TOADDR     (2)
44  #define BBC_FTP_BYTES_FROMADDR   (2)
45  #define BBC_FTP_BYTES_RSSI               (2)
46
47
48  #define BBC_FTP_OFFSET_CHECKSUM (0)
49  #define BBC_FTP_OFFSET_ID         (BBC_FTP_OFFSET_CHECKSUM + BBC_FTP_BYTES_CHECKSUM)
50  #define BBC_FTP_OFFSET_SEQNUM     (BBC_FTP_OFFSET_ID         + BBC_FTP_BYTES_ID         )
51  #define BBC_FTP_OFFSET_LOADBITS (BBC_FTP_OFFSET_SEQNUM     + BBC_FTP_BYTES_SEQNUM     )
52  #define BBC_FTP_OFFSET_TOADDR     (BBC_FTP_OFFSET_LOADBITS + BBC_FTP_BYTES_LOADBITS)
53  #define BBC_FTP_OFFSET_FROMADDR (BBC_FTP_OFFSET_TOADDR + BBC_FTP_BYTES_TOADDR)
54  #define BBC_FTP_OFFSET_RSSI               (BBC_FTP_OFFSET_FROMADDR + BBC_FTP_BYTES_FROMADDR)
```

```
55 #define BBC_FTP_OFFSET_PAYLOAD   (BBC_FTP_OFFSET_RSSI + BBC_FTP_BYTES_RSSI)
56 #define BBC_FTP_HEADER_BYTES     (BBC_FTP_OFFSET_PAYLOAD)
57
58 //-----------------------------------------------------------------------------
59 // STRUCTURE TYPE DEFINITIONS
60 //-----------------------------------------------------------------------------
61
62 typedef struct BBCFTP BBCFTP;
63
64 //-----------------------------------------------------------------------------
65 // STRUCTURE DEFINITIONS
66 //-----------------------------------------------------------------------------
67
68 // NOTE: Normally the structure definition would be in the *.c file to make
69 // the structure members inaccessible to outside functions except through
70 // public function calls. But for the real-time code it has been decided
71 // to make the structure members directly visible to the functions that
72 // manipulate them.
73
74 struct BBCFTP
75 {
76         CONFIG *config;
77         SOURCE *source;
78         CODEC  *codec;
79         BUFFER *buffer;
80         MODEM  *modem;
81         SINK   *sink;
82 };
83
84 //-----------------------------------------------------------------------------
85 // PUBLIC FUNCTION PROTOTYPES
86 //-----------------------------------------------------------------------------
87
88 BBCFTP *BBCFTP_Del(BBCFTP *p);
89 BBCFTP *BBCFTP_New(char *filename, DWORD *errcode);
90
91 void PrintMessage(BYTE *base);
92 //void PrintDistinctMessage(BYTE *base, int *seqnum_index, char *strarray);
93
94 void SetMessageRSSI(BYTE *base, WORD v);
95 void SetMessageFromAddr(BYTE *base, WORD v);
96 void SetMessageToAddr(BYTE *base, WORD v);
97 void  SetMessageChecksum(BYTE *base, DWORD v);
98 void  SetMessageSeq(BYTE *base, WORD v);
99 void  SetMessageLoadBits(BYTE *base, WORD v);
```

```
100  void   SetMessageID(BYTE *base, WORD v);
101  void   SetMessagePayload(BYTE *base, BYTE *source, DWORD bytes, int offset);
102
103  WORD GetMessageRSSI(BYTE *base);
104  WORD GetMessageFromAddr(BYTE *base);
105  WORD GetMessageToAddr(BYTE *base);
106  DWORD GetMessageChecksum(BYTE *base);
107  WORD  GetMessageSeq(BYTE *base);
108  WORD  GetMessageLoadBits(BYTE *base);
109  WORD  GetMessageID(BYTE *base);
110  BYTE *GetMessagePayload(BYTE *base);
111
112
113  //---------------------------------------------------------------------------
114  #endif
```

C.2.6 codec.c and codec.h

```
1   /*****************************************************************************
2    * CODEC for the Real-time BBC Codec/Modem                                   *
3    *****************************************************************************
4    * William L. Bahn                                                           *
5    * Academy Center for Information Security                                   *
6    * Department of Computer Science                                            *
7    * United States Air Force Academy                                           *
8    * USAFA, CO 80840                                                           *
9    *****************************************************************************
10   * FILE:............ codec.c                                                  *
11   * DATE CREATED:.... 06 SEP 07                                               *
12   * DATE MODIFIED:... 06 SEP 07                                               *
13   *****************************************************************************
14   *
15   * REVISION HISTORY
16   *
17   *****************************************************************************
18   *
19   * DESCRIPTION
20   *
21   * The codec and its public interface are described in codec.h
22   *
23   *****************************************************************************
24   */
25
26  //---------------------------------------------------------------------------
27  // REQUIRED INCLUDES
```

```
28  //-------------------------------------------------------------------------
29
30  #include <stdlib.h> // free(), malloc()
31
32  #include "codec.h"
33  #include "sha1.h"
34
35  //-------------------------------------------------------------------------
36  // STRUCTURE DEFINITIONS
37  //-------------------------------------------------------------------------
38
39  // NOTE: Normally the structure definition would be in the *.c file to make
40  // the structure members inaccessible to outside functions except through
41  // public function calls. But for the real-time code it has been decided
42  // to make the structure members directly visible to the functions that
43  // manipulate them.
44
45  //-------------------------------------------------------------------------
46  // PRIVATE FUNCTION DEFINITIONS
47  //-------------------------------------------------------------------------
48
49  #define SHA1_HASH_DWORDS (5)
50
51  void ExportMessage(CONFIG *c, CODEC *codec, SINK *sink)
52  {
53          DWORD i;
54          DWORD bit;
55          DWORD index, offset;
56
57          BYTE *message;
58
59          // Create pointer to next element in sink memory
60          message = sink->v + (sink->samples * sink->sample_size_bytes);
61
62          // Discard leading random bits
63          for (bit = 0, i = 0; i < c->codec_random_bits; i++, bit++)
64          {
65                  while (codec->checkbit[bit])
66                          bit++;
67          }
68
69          // Extract message bits and pack into byte string
70          for (i = 0; i < c->codec_message_bits; i++, bit++)
71          {
72                  while (codec->checkbit[bit])
```

```
73                          bit++;
74                  index  = i >> 3;
75                  offset = i & 0x00000007;
76                  message[index] &= ~c->bitmask[7-offset];
77                  if (codec->msg[bit])
78                          message[index] |= c->bitmask[7-offset];
79          }
80
81          // Zero pad remainder of last byte if necessary
82          while (7 != offset)
83          {
84                  i++;
85                  offset = i & 0x00000007;
86                  message[index] &= ~c->bitmask[7-offset];
87          }
88
89          // NUL terminate byte string
90          index++;
91          message[index] = '\0';
92
93          // Advance sink memory pointer
94          sink->samples++;
95  }
96
97  //---------------------------------------------------------------------------
98  // PUBLIC FUNCTION DEFINITIONS
99  //---------------------------------------------------------------------------
100
101 CODEC *CODEC_Del(CODEC *p)
102 {
103         if (p)
104         {
105                 if (p->state)     { free(p->state);     p->state = NULL;     }
106                 if (p->digest)    { free(p->digest);    p->digest = NULL;    }
107                 if (p->hashstate) { free(p->hashstate); p->hashstate = NULL; }
108                 if (p->msg)       { free(p->msg);       p->msg = NULL;       }
109                 if (p->checkbit)  { free(p->checkbit);  p->checkbit = NULL;  }
110         }
111         return NULL;
112 }
113
114 CODEC *CODEC_New(CONFIG *c, DWORD *errcode)
115 {
116         CODEC *p;
117         DWORD err;
```

```
118         DWORD check;
119         DWORD i, run;
120         err = 0;
121
122         p = (CODEC *) malloc(sizeof(CODEC));
123         if (!p)
124                 err = 1 << 0;
125
126         if (!err)
127         {
128                 // State information
129                 p->state  = (SHA1Context *) malloc(sizeof(SHA1Context));
130                 p->digest = (SHA1Context *) malloc(sizeof(SHA1Context));
131
132                 // Decode Buffer
133                 p->msg       = (BYTE *) malloc(c->padded_message_bits);
134                 p->checkbit  = (BYTE *) malloc(c->padded_message_bits);
135                 p->hashstate = (SHA1Context *) malloc(c->padded_message_bits*sizeof(SHA1Context));
136
137                 // Verify successful memory allocation
138                 if (!(p->state))      err |= 1 << 1;
139                 if (!(p->digest))     err |= 1 << 2;
140                 if (!(p->msg))        err |= 1 << 3;
141                 if (!(p->checkbit))   err |= 1 << 4;
142                 if (!(p->hashstate))  err |= 1 << 5;
143         }
144
145         if (!err)
146         {
147                 // Initialize checkbit markers
148                 for (check = TRUE, run = 0, i = 0; i < (c->padded_message_bits - c->
                         codec_stop_bits); i++)
149                 {
150                         switch (check)
151                         {
152                                 case TRUE:
153                                         if (run >= c->codec_clamp_bits)
154                                         {
155                                                 check = FALSE;
156                                                 run = 0;
157                                         }
158                                         break;
159                                 case FALSE:
160                                         if (run >= c->codec_fragment_bits)
161                                         {
```

```
162                                             check = TRUE;
163                                             run = 0;
164                                         }
165                                         break;
166                         }
167                         p->checkbit[i] = check;
168                         run++;
169                 }
170
171                 for (i = 0; i < c->codec_stop_bits; i++)
172                         p->checkbit[c->padded_message_bits - c->codec_stop_bits + i] = TRUE;
173
174         }
175
176         if (c->diagnostics)
177         {
178                 // Diagnostic Report
179                 printf("----------------------------------------\n");
180                 printf("CODEC\n");
181                 printf("  Creation:.............. %s\n", ((err)? "FAILED":"SUCCEEDED"));
182                 printf("  Location:.............. %p\n", (void *) p);
183                 printf("  Message bits:.......... %li\n", (unsigned long) c->codec_message_bits);
184                 printf("  Random bits:........... %li\n", (unsigned long) c->codec_random_bits);
185                 printf("  Clamp bits:............ %li\n", (unsigned long) c->codec_clamp_bits);
186                 printf("  Fragment bits:......... %li\n", (unsigned long) c->codec_fragment_bits)
                        ;
187                 printf("  Stop bits:............. %li\n", (unsigned long) c->codec_stop_bits);
188                 printf("  Padded message length:.. %li\n", (unsigned long) c->padded_message_bits)
                        ;
189                 printf("  Packet expansion:...... %li\n", (unsigned long) c->codec_expansion);
190                 printf("  Packet load:........... %li messages\n", (unsigned long) c->
                        codec_packet_load);
191                 printf("  Decode limit:.......... %li messages\n", (unsigned long) c->
                        codec_decode_limit);
192                 printf("  Message buffer at:..... %p\n", p->msg);
193                 printf("  Checksum buffer at:.... %p\n", p->checkbit);
194                 printf("  Hash buffer at:........ %p\n", p->hashstate);
195                 printf("  State buffer at:....... %p\n", p->state);
196                 printf("  Digest buffer at:...... %p\n", p->digest);
197                 printf("----------------------------------------\n");
198         }
199
200         if (err)
201                 p = CODEC_Del(p);
202
```

```
203             *errcode = err;
204             return p;
205     }
206
207     /* DECODER
208      *
209      * The decoder decodes all eight of the packets that start with each of the
210      * eight bits in the byte located at the present "read" location of the buffer.
211      *
212      * The value of the variable "originbit" determines which of the eight offsets
213      * from the beginning of the byte the present packet starts at. The variable
214      * "location" refers to the location of the bit in question relative to the
215      * beginning of the packet. Therefore, relative to the beginning of the byte
216      * where the packet starts, the location is simply "origin + location". This
217      * combined location must then be turned into an index and and offset. The
218      * "index" refers to which byte within the buffer contains the bit of interest
219      * while the "offset" identifies the bit within that byte. The "index" value
220      * must further account for the fact that the first byte in the packet is
221      * located at the "read" point within the index and that the buffer is circular.
222      * The "offset" value must be used to mask the byte being examined so that only
223      * the bit of interest is considered. For speed purposes, this mask is provided
224      * by a lookup table "bitmask".
225      *
226      * Taking all of this into account, the following steps will check if a
227      * particular packet bit is set:
228      *
229      * index  = {read + floor[(location + originbit)/8]} mod bufferlength
230      * offset = (location + originbit) mod 8
231      * status = buffer[index] & bitmask[offset]
232      *
233      * Since the buffer length is exactly 2^n long, the residue of the index can
234      * be taken by simply retaining only the lower n bits. Similarly, the residue
235      * of the offset modulo-8 can be taken by only retaining the lower 3 bits. Both
236      * of these can be done by performing a bitwise-AND with an appropriate mask.
237      * Finally, the division of the effective location within the packet can be
238      * performed by right-shifting the sum by 3 bits. Hence we have the following
239      * equations:
240      *
241      * index = (read + ((location + originbit) >> 3)) & buffermask;
242      * offset = (location + originbit) & 0x00000007;
243      * status = buffer[index] & bitmask[offset]
244      *
245      * The most challenging part of the decoding algorithm is the backtracking that
246      * must take place when the present partial message is finished, either because
247      * it was found to be a dead end or because it resulted in an actual message.
```

```
248   * The basic task is to traverse the decoding tree backwards until the last
249   * partial message bit that was a zero is found. Then that bit is changed to a one
250   * and decoding moves forward again. Two special cases have to be taken into
251   * account. First, if there are no message bits that are zero, then the decoding
252   * of that packet is finished. Second, checksum bits are always zero and the
253   * decoder must skip over them without turning them to ones.
254   * index    0123456789....
255   * check    1001001001....
256   * msg      0010110110....
257   *
258   */
259
260  //---------------------------------------------------------------------------
261
262  /*
263   * The encoding function can be implemented in a more compact, efficient
264   * way. The method used here is intended to mirror the decode operation
265   * as closely as possible. This is reasonable because the encoding
266   * operation requires constant time regardless of message and is therefore
267   * well constrained.
268   *
269   */
270
271  void Encode(CONFIG *c, SOURCE *source, CODEC *codec, BUFFER *buffer)
272  {
273          DWORD location, msg_bit, pmsg_bit, r, i, index, offset;
274          int bit_value;
275          BYTE *msg;
276          DWORD message_stop;
277          clock_t ticks;
278          DWORD marks;
279
280          ticks = clock();
281
282          marks = 0;
283          message_stop = source->sample + c->codec_packet_load;
284          if (message_stop > source->samples)
285                  message_stop = source->samples;
286
287          // Place bookend marks
288          location = 0;
289          index = (buffer->write + (location >> 3)) & buffer->buffermask;
290          offset  = location & 0x00000007;
291          if (buffer->buffer[index] & c->bitmask[offset])
292                  marks--;
```

```
293             buffer->buffer[index] |= c->bitmask[offset];
294             marks++;
295
296             location = c->last_packet_bit;
297             index = (buffer->write + (location >> 3)) & buffer->buffermask;
298             offset  = location & 0x00000007;
299             if (buffer->buffer[index] & c->bitmask[offset])
300                     marks--;
301             buffer->buffer[index] |= c->bitmask[offset];
302             marks++;
303
304             while (source->sample < message_stop)
305             {
306                     if (c->diagnostics)
307                             printf("Encoding message #%lu\n", source->sample);
308                     // Compute pointer to beginning of present message in source buffer
309                     msg = (BYTE *) source->v + source->sample * source->sample_size_bytes;
310
311                     // Initialize Hash Function state to the Initial Vector
312                     SHA1Reset(codec->state);
313
314                     // Load message into the codec's message buffer
315                     for (pmsg_bit = 0, r = 0, msg_bit = 0 ; pmsg_bit < c->padded_message_bits;
                             pmsg_bit++)
316                     {
317                             if (codec->checkbit[pmsg_bit])
318                                     bit_value = 0;
319                             else
320                             {
321                                     if (r < c->codec_random_bits)
322                                     {
323                                             bit_value = rand() < (RAND_MAX >> 1);
324                                             r++;
325                                     }
326                                     else
327                                     {
328                                             index = msg_bit >> 3;
329                                             offset = msg_bit & 0x00000007;
330                                             bit_value = (msg[index] & c->bitmask[7-offset])? 1 : 0;
331                                             msg_bit++;
332                                     }
333                             }
334                             SHA1Input(codec->state, c->bitptr + bit_value, 1);
335
336                             // Compute hash result for present prefix
```

```
337                             *(codec->digest) = *(codec->state);
338                             SHA1Result(codec->digest);
339
340                             // Generate mark location for present prefix
341                             location = 0;
342                             for (i = 0; i < SHA1_HASH_DWORDS; i++)
343                                     location += ((codec->digest)->Message_Digest[i])<<i;
344                             location %= c->packet_bits;
345
346                             // Place mark for present prefix
347                             index = (buffer->write + (location >> 3)) & buffer->buffermask;
348                             offset  = location & 0x00000007;
349                             if (buffer->buffer[index] & c->bitmask[offset])
350                                     marks--;
351                             buffer->buffer[index] |= c->bitmask[offset];
352                             marks++;
353                     }
354                     source->sample++;
355             }
356
357             if (source->sample >= source->samples)
358             {
359                     // Last packet has been encoded. Advance buffer past last packet.
360                     source->streaming = FALSE;
361                     c->buffer_advance = c->bufferbytes_per_packet;
362             }
363
364             // Advance buffer write pointer to next packet write location.
365             buffer->write = (buffer->write + c->buffer_advance) & buffer->buffermask;
366             buffer->margin -= c->buffer_advance;
367             buffer->ready += c->buffer_advance;
368
369             c->dec_ticks += clock() - ticks;
370     }
371
372     void Decode(CONFIG *c, BUFFER *buf, CODEC *codec, SINK *sink)
373     {
374             SDWORD i, bit;
375             DWORD location, index, offset, originbit;
376             clock_t ticks;
377             DWORD limit;
378
379             ticks = clock();
380
381             // Process all 8 packets that begin within the byte at the front of the buffer
```

```
382         for (originbit = 0; originbit < 8; originbit++)
383         {
384                 if ((sink->sample_limit - sink->samples) > c->codec_decode_limit)
385                         limit = (sink->sample_limit - sink->samples);
386                 else
387                         limit = c->codec_decode_limit;
388
389                 // Check for bookend marks
390                 index = (buf->read + (originbit >> 3)) & buf->buffermask;
391                 offset  = (originbit) & 0x00000007;
392                 if ( !(buf->buffer[index] & c->bitmask[offset]) )
393                         break;
394
395                 index = (buf->read + ((originbit + c->last_packet_bit) >> 3)) & buf->buffermask;
396                 offset  = (originbit + c->last_packet_bit) & 0x00000007;
397                 if ( !(buf->buffer[index] & c->bitmask[offset]) )
398                         break;
399
400                 // Initialize Hash Function state to the Initial Vector
401                 SHA1Reset(codec->state);
402                 bit = 0;
403                 codec->msg[bit] = 0;
404
405                 while (TRUE) // Loop will terminate with a "break" call
406                 {
407                         // Update the hash state for the new message bit
408                         SHA1Input(codec->state, c->bitptr + codec->msg[bit], 1);
409
410                         // Compute the packet bit location corresponding to the hash
411                         *(codec->digest) = *(codec->state);
412                         SHA1Result(codec->digest);
413                         location = 0;
414                         for (i = 0; i < SHA1_HASH_DWORDS; i++)
415                                 location += ((codec->digest)->Message_Digest[i])<<i;
416                         location %= c->packet_bits;
417
418                         // Check for mark at calculated location
419                         index = (buf->read + ((originbit + location) >> 3)) & buf->buffermask;
420                         offset  = (originbit + location) & 0x00000007;
421                         if(buf->buffer[index] & c->bitmask[offset])
422                         {
423                                 // Update hash state for present partial message
424                                 codec->hashstate[bit] = *(codec->state);
425                                 bit++;
426                                 // IF a complete message hasn't been decoded yet
```

```
427                                 if ((DWORD) bit < c->padded_message_bits)
428                                 {
429                                         // Start with 0 for next bit in partial message
430                                         codec->msg[bit] = 0;
431                                         continue;
432                                 }
433                                 // ELSE a complete message has been found
434                                 c->message_count++;
435                                 ExportMessage(c, codec, sink);
436                                 bit--;
437                                 limit--;
438                                 if (0 == limit)
439                                         break;
440                         }
441
442                         // Backtrack to last message bit that is a zero
443                         while ( (bit >= 0) && (codec->checkbit[bit] || codec->msg[bit]) )
444                                 bit--;
445
446                         // If no bits are zero, then decoding is finished
447                         if (bit < 0)
448                                 break;
449
450                         // Change last zero bit to a one
451                         codec->msg[bit] = 1;
452
453                         // Reset hash state
454                         if (0 == bit) // to initial vector
455                                 SHA1Reset(codec->state);
456                         else          // to vector of previous partial message
457                                 *(codec->state) = codec->hashstate[bit-1];
458                 }
459         }
460         buf->read = (buf->read + 1) & buf->buffermask;
461         buf->empty++;
462         buf->margin--;
463
464         c->dec_ticks += clock() - ticks;
465 }
466
467 //---------------------------------------------------------------------------
```

```
1  /*****************************************************************************
2   * CODEC for the Real-time BBC Codec/Modem                                   *
3   *****************************************************************************
```

```
4    * William L. Bahn                                                    *
5    * Academy Center for Information Security                             *
6    * Department of Computer Science                                      *
7    * United States Air Force Academy                                     *
8    * USAFA, CO 80840                                                     *
9    ***************************************************************************
10   * FILE:............ codec.h                                           *
11   * DATE CREATED:.... 06 SEP 07                                         *
12   * DATE MODIFIED:... 06 SEP 07                                         *
13   ***************************************************************************
14   *
15   * REVISION HISTORY
16   *
17   ***************************************************************************
18   *
19   * DESCRIPTION
20   *
21   * The codec encodes and decodes messages to/from BBC-encoded packets.
22   *
23   ***************************************************************************
24   */
25
26   #ifndef CODECdotH
27   #define CODECdotH
28
29   //---------------------------------------------------------------------------
30   // REQUIRED INCLUDES
31   //---------------------------------------------------------------------------
32
33   #include "config.h"
34   #include "source.h"
35   #include "buffer.h"
36   #include "sink.h"
37   #include "dirtyd.h"
38   #include "sha1.h"
39
40   //---------------------------------------------------------------------------
41   // STRUCTURE DECLARATIONS
42   //---------------------------------------------------------------------------
43
44   typedef struct CODEC CODEC;
45
46   //---------------------------------------------------------------------------
47   // STRUCTURE DEFINITIONS
48   //---------------------------------------------------------------------------
```

```
49
50  // NOTE: Normally the structure definition would be in the *.c file to make
51  // the structure members inaccessible to outside functions except through
52  // public function calls. But for the real-time code it has been decided
53  // to make the structure members directly visible to the functions that
54  // manipulate them.
55
56  struct CODEC
57  {
58          // State information
59          SHA1Context *state;       // Pointer to single SHA1 structure
60          SHA1Context *digest;      // Pointer to single SHA1 structure
61
62          // Decode buffer
63          BYTE    *msg;              // Array containing the message bit contents (1 bit per byte)
64          BYTE    *checkbit;         // Array indicating whether each bit is a message or check bit
65          SHA1Context *hashstate;  // Array of SHA1 structures
66  };
67
68  //---------------------------------------------------------------------------
69  // PUBLIC FUNCTION PROTOTYPES
70  //---------------------------------------------------------------------------
71
72  CODEC *CODEC_Del(CODEC *p);
73  CODEC *CODEC_New(CONFIG *c, DWORD *errcode);
74  void Encode(CONFIG *c, SOURCE *source, CODEC *codec, BUFFER *buffer);
75  void Decode(CONFIG *c, BUFFER *buf, CODEC *codec, SINK *sink);
76
77  /* DECODER
78   *
79   * The decoder decodes all eight of the packets that start with each of the
80   * eight bits in the byte located at the present "read" location of the buffer.
81   *
82   * The value of the variable "originbit" determines which of the eight offsets
83   * from the beginning of the byte the present packet starts at. The variable
84   * "location" refers to the location of the bit in question relative to the
85   * beginning of the packet. Therefore, relative to the beginning of the byte
86   * where the packet starts, the location is simply "origin + location". This
87   * combined location must then be turned into an index and and offset. The
88   * "index" refers to which byte within the buffer contains the bit of interest
89   * while the "offset" identifies the bit within that byte. The "index" value
90   * must further account for the fact that the first byte in the packet is
91   * located at the "read" point within the index and that the buffer is circular.
92   * The "offset" value must be used to mask the byte being examined so that only
93   * the bit of interest is considered. For speed purposes, this mask is provided
```

```
 * by a lookup table "bitmask".
 *
 * Taking all of this into account, the following steps will check if a
 * particular packet bit is set:
 *
 * index  = {read + floor[(location + originbit)/8]} mod bufferlength
 * offset = (location + originbit) mod 8
 * status = buffer[index] & bitmask[offset]
 *
 * Since the buffer length is exactly 2^n long, the residue of the index can
 * be taken by simply retaining only the lower n bits. Similarly, the residue
 * of the offset modulo-8 can be taken by only retaining the lower 3 bits. Both
 * of these can be done by performing a bitwise-AND with an appropriate mask.
 * Finally, the division of the effective location within the packet can be
 * performed by right-shifting the sum by 3 bits. Hence we have the following
 * equations:
 *
 * index = (read + ((location + originbit) >> 3)) & buffermask;
 * offset = (location + originbit) & 0x00000007;
 * status = buffer[index] & bitmask[offset]
 *
 * The most challenging part of the decoding algorithm is the backtracking that
 * must take place when the present partial message is finished, either because
 * it was found to be a dead end or because it resulted in an actual message.
 * The basic task is to traverse the decoding tree backwards until the last
 * partial message bit that was a zero is found. Then that bit is changed to a one
 * and decoding moves forward again. Two special cases have to be taken into
 * account. First, if there are no message bits that are zero, then the decoding
 * of that packet is finished. Second, checksum bits are always zero and the
 * decoder must skip over them without turning them to ones.
 * index    0123456789....
 * check    1001001001....
 * msg      0010110110....
 *
 */

//---------------------------------------------------------------------------
#endif
```

C.2.7 buffer.c and buffer.h

```
/*****************************************************************************
 * Data Buffer for the Real-time BBC Codec/Modem                             *
 *****************************************************************************
 * William L. Bahn                                                           *
```

```
5   * Academy Center for Information Security                                *
6   * Department of Computer Science                                         *
7   * United States Air Force Academy                                        *
8   * USAFA, CO 80840                                                        *
9   ****************************************************************************
10  * FILE:............ buffer.c                                             *
11  * DATE CREATED:.... 01 SEP 07                                            *
12  * DATE MODIFIED:... 01 SEP 07                                            *
13  ****************************************************************************
14  *
15  * REVISION HISTORY
16  *
17  ****************************************************************************
18  *
19  * DESCRIPTION
20  *
21  * The data buffer and its programmer interface is described in buffer.h.
22  *
23  ****************************************************************************
24  */
25
26  //---------------------------------------------------------------------------
27  // REQUIRED INCLUDES
28  //---------------------------------------------------------------------------
29
30  #include <stdlib.h> // malloc(), free()
31  #include <string.h> // memset()
32
33  #include "buffer.h"
34
35  //---------------------------------------------------------------------------
36  // STRUCTURE DEFINITIONS
37  //---------------------------------------------------------------------------
38
39  // NOTE: Normally the structure definition would be in the *.c file to make
40  // the structure members inaccessible to outside functions except through
41  // public function calls. But for the real-time code it has been decided
42  // to make the structure members directly visible to the functions that
43  // manipulate them.
44
45  //---------------------------------------------------------------------------
46  // PRIMITIVE FUNCTION DEFINITIONS
47  //---------------------------------------------------------------------------
48
49  //---------------------------------------------------------------------------
```

```
50  // PRIVATE FUNCTION DEFINITIONS
51  //---------------------------------------------------------------------------
52
53  //---------------------------------------------------------------------------
54  // PUBLIC FUNCTION DEFINITIONS
55  //---------------------------------------------------------------------------
56
57  BUFFER *BUFFER_Del(BUFFER *p)
58  {
59          if (p)
60          {
61                  if (p->buffer) { free (p->buffer); p->buffer = NULL;  }
62          }
63          return NULL;
64  }
65
66  BUFFER *BUFFER_New(CONFIG *c, DWORD *errcode)
67  {
68          BUFFER *p;
69          DWORD err;
70
71          p = NULL;
72          err = 0;
73
74          if (!err)
75          {
76                  p = (BUFFER *) malloc(sizeof(BUFFER));
77                  if (!p)
78                          err |= 1 << 1;
79          }
80
81          if (!err)
82          {
83                  p->minsize = (size_t) (c->bufferbytes_per_packet * c->buffer_packets);
84                  p->size = 1;
85                  while ((0 != p->size)&&(p->size < p->minsize))
86                          p->size <<= 1;
87                  if (0 == p->size)
88                          err |= 1 << 2;
89          }
90
91          if (!err)
92          {
93                  // Allocate buffer memory
94                  p->buffer = (BYTE *) malloc(p->size*sizeof(BYTE));
```

```
95                  if (!p->buffer)
96                          err |= 1 << 3;
97
98                  // Initialize buffer state
99
100                 // Common to TX and RX
101                 p->buffermask = p->size - 1;
102                 p->scope = c->bufferbytes_per_packet;
103                 p->read = 0;
104                 p->write = 0;
105                 p->overflows = 0;
106
107                 // TX and RX specific
108                 if (c->scheduler_TX_notRX)
109                 {
110                         p->margin = p->size - p->scope;
111                         p->ready = 0;
112                         p->empty = 0; // Not used
113                 }
114                 else
115                 {
116                         p->margin = -((SDWORD)p->scope);
117                         p->ready = 0; // Not used
118                         p->empty = p->size;
119                 }
120         }
121
122         // Clear entire buffer
123         if (!err)
124                 memset(p->buffer, 0, p->size);
125
126         if (err)
127                 p = BUFFER_Del(p);
128
129         if (c->diagnostics)
130         {
131                 // Diagnostic Report
132                 printf("-----------------------------------------\n");
133                 printf("PACKET BUFFER\n");
134                 printf("  Creation:............. %s\n", ((err)? "FAILED":"SUCCEEDED"));
135                 printf("  Location:............. %p\n", (void *) p);
136                 printf("  Minimum buffer size:.... %lu bytes\n", (unsigned long) p->minsize);
137                 printf("  Buffer size:.......... %lu bytes\n", (unsigned long) p->size);
138                 printf("  Buffer location:...... %p\n", (void *) p->buffer);
139                 printf("  Packet size in buffer:.. %lu bytes\n", (unsigned long) p->scope);
```

```
140                 printf("  read:................. %lu bytes\n", (unsigned long) p->read);
141                 printf("  write:................ %lu bytes\n", (unsigned long) p->write);
142                 printf("  empty:................ %lu bytes\n", (unsigned long) p->empty);
143                 printf("  ready:................ %lu bytes\n", (unsigned long) p->ready);
144                 printf("  margin:............... %li bytes\n", (long) p->margin);
145                 printf("  overflows:............ %lu bytes\n", (unsigned long) p->overflows);
146                 printf("----------------------------------------\n");
147         }
148
149         *errcode = err;
150         return p;
151 }
152
153 //---------------------------------------------------------------------------
```

```
1  /*****************************************************************************
2   * Data Buffer for the Real-time BBC Codec/Modem                             *
3   *****************************************************************************
4   * William L. Bahn                                                           *
5   * Academy Center for Information Security                                   *
6   * Department of Computer Science                                            *
7   * United States Air Force Academy                                           *
8   * USAFA, CO 80840                                                           *
9   *****************************************************************************
10  * FILE:............ buffer.h                                                 *
11  * DATE CREATED:.... 01 SEP 07                                               *
12  * DATE MODIFIED:... 01 SEP 07                                               *
13  *****************************************************************************
14  *
15  * REVISION HISTORY
16  *
17  *****************************************************************************
18  *
19  * DESCRIPTION
20  *
21  * The data buffer stores packet data between the codec and the modem.
22  *
23  * In the receiver, the buffer accepts packet data from the codec and
24  * feeds that data to the modem. In the transmitter, it accepts data from
25  * the modem and feeds it to the codec. While the modem, by its nature,
26  * generally produces and consumes data at a uniform rate, the codec
27  * can be quite erratic in its data rate. Therefore the buffer must be
28  * sized sufficiently large to allow for the resulting ebb and flow.
29  * This is particularly important in the case of the receiver since, if
30  * the buffer can't accommodate the data as the modem delivers it, data
```

```
31   * will be lost. This is not as critical with the transmitter, depending
32   * on the nature of the data source and its buffering strategy, since it
33   * will normally only reduce the effective data rate as opposed to causing
34   * dropped packets.
35   *
36   * The data is stored in a circular buffer with the following variables:
37   *
38   *    buffer: Pointer to the block of memory where the buffer starts.
39   *    read:   Index of the first byte of the present packet.
40   *    write:  Index of the next unused buffer location.
41   *    margin: How many bytes are in buffer beyond the scope of the decoder.
42   *    unused: How many unused bytes are available in the buffer.
43   *
44   * The buffer is seen by two functions, the one that is demodulating the
45   * data packet and the one that is decoding the resulting data. The
46   * demodulating function writes to the buffer at a nominally constant
47   * rate dictated by the communications link. In this application, this is
48   * simulated by reading the stored waveform data from a file and querying
49   * the clock to determine how many bytes to add to the buffer each time
50   * the function is called. The decoding function, on the other hand, always
51   * to decodes eight packets each time it is called provided sufficient data
52   * is available. Specifically, it decodes the eight packets that start with
53   * the bits in the byte stored at the "read" pointer. Since it can't decode
54   * packets that are not completely contained in the buffer, the decoding
55   * function first checks to see if "fill" is non-negative. If it isn't, then
56   * it returns immediately. At the other end of the spectrum, the demodulator
57   * may run out of unused memory to write to. If this happens, data is going
58   * to be lost. It is cleaner to throw away old data instead of introducing
59   * a gap in present data, therefore the demodulator will push the "read"
60   * pointer forward as it overwrites the beginning of the existing packet
61   * data.
62   *
63   */
64
65  #ifndef BUFFERdotH
66  #define BUFFERdotH
67
68  //-------------------------------------------------------------------------
69  // REQUIRED INCLUDES
70  //-------------------------------------------------------------------------
71
72  #include "config.h"
73  #include "dirtyd.h"
74
75  //-------------------------------------------------------------------------
```

```
76  // STRUCTURE DECLARATIONS
77  //---------------------------------------------------------------------------
78
79  typedef struct BUFFER BUFFER;
80
81  //---------------------------------------------------------------------------
82  // STRUCTURE DEFINITIONS
83  //---------------------------------------------------------------------------
84
85  // NOTE: Normally the structure definition would be in the *.c file to make
86  // the structure members inaccessible to outside functions except through
87  // public function calls. But for the real-time code it has been decided
88  // to make the structure members directly visible to the functions that
89  // manipulate them.
90
91  struct BUFFER
92  {
93          size_t   size;          // Allocated size of buffer (in bytes)
94          size_t   minsize;       // Minimum acceptable buffer size (in bytes)
95          BYTE    *buffer;        // Pointer to the actual buffer
96          DWORD    read;          // Index of next position to be read.
97          DWORD    write;         // Index of next position to be written.
98          DWORD    scope;         // The number of bytes recipient must.
99          SDWORD   margin;        // Number of bytes beyond scope of recipient
100         DWORD    empty;         // Number of bytes available for new data.
101         DWORD    ready;         // Number of bytes ready for modulation.
102         DWORD    buffermask;    // The used bits in the buffer size
103         DWORD    overflows;     // Number of data pushes into read pointer.
104 };
105
106 //---------------------------------------------------------------------------
107 // PUBLIC FUNCTION PROTOTYPES
108 //---------------------------------------------------------------------------
109
110 BUFFER *BUFFER_Del(BUFFER *p);
111 BUFFER *BUFFER_New(CONFIG *c, DWORD *errcode);
112
113 //---------------------------------------------------------------------------
114 #endif
```

C.2.8 bytes.c and bytes.h

```
1  /* ======================================================================
2   * PROGRAMMER "BAHN, William"
3   * TITLE      "Integer Storage Size Type Definitions"
```

```
4    * CREATED      06 FEB 07
5    * MODIFIED     06 FEB 07
6    * FILENAME     "bytes.c"
7    * ================================================================
8    * GENERAL DESCRIPTION
9    *
10   * NOTE: ANY AVAILABLE "USER GUIDE" IS IN THE ASSOCIATED HEADER FILE.
11   *
12   * This file contains type definitions so that porting from one processor
13   * to another is simpler.
14   *
15   * ================================================================
16   */
17
18  #include "bytes.h"
19
20  int VerifyBYTE(void)
21  {
22          return (8*sizeof(BYTE) != BITSinBYTE);
23  }
24
25  int VerifyWORD(void)
26  {
27          return (8*sizeof(WORD) != BITSinWORD);
28  }
29
30  int VerifyDWORD(void)
31  {
32          return (8*sizeof(DWORD) != BITSinDWORD);
33  }
34
35  int VerifyQWORD(void)
36  {
37          return (8*sizeof(QWORD) != BITSinQWORD);
38  }
39
40  unsigned int VerifySIZES(unsigned int maxlength)
41  {
42          unsigned int flags;
43          unsigned int mask;
44
45          // Generate a flag vector with a 1 set anyplace that does not
46          // verify properly. Note that the bit position is equal to base-2
47          // log of the number of bytes in the integer type.
48
```

```
49          flags = 0;
50          flags = (flags << 1) + VerifyQWORD();
51          flags = (flags << 1) + VerifyDWORD();
52          flags = (flags << 1) + VerifyWORD();
53          flags = (flags << 1) + VerifyBYTE();
54
55          // Convert length from bits to smallest compatible number of bytes.
56
57          maxlength = (maxlength/8) + ((maxlength%8)?1:0);
58
59          // Generate a mask that is set only in those flag positions of interest.
60
61          if (maxlength) // report on sizes up to and including maxlength.
62                  for (mask = 0; maxlength > 0; maxlength /= 2)
63                  {
64                          mask = (mask << 1) + 1;
65                          if ((maxlength > 1)&&(maxlength%2))
66                                  mask = (mask << 1) + 1;
67                  }
68          else // report on all defined sizes
69                  mask = ~0;
70
71          return (flags & mask);
72  }
```

```
1   /* ========================================================================
2    * PROGRAMMER "BAHN, William"
3    * TITLE      "Integer Storage Size Type Definitions"
4    * CREATED    06 FEB 07
5    * MODIFIED   06 FEB 07
6    * FILENAME   "bytes.h"
7    * ========================================================================
8    * GENERAL DESCRIPTION
9    *
10   * This file contains type definitions so that porting from one processor
11   * to another is simpler.
12   * ========================================================================
13   * SIZE DEFINITIONS
14   *
15   * The following definitions are used:
16   *
17   *   SIZE    UNSIGNED    SIGNED
18   *   8-bit     BYTE      SBYTE
19   *  16-bit     WORD      SWORD
20   *  32-bit     DWORD     SDWORD
```

```
21   *  64-bit     QWORD    SQWORD (not available on most systems)
22   *
23   * ============================================================
24   * To Verify Sizes
25   *
26   * Use the VerifySIZES() function passing the largest integer size, in
27   * bits, that is of interest.
28   *
29   * The function returns TRUE if conflicts are found.
30   *
31   * If an argument of 0 is used, then the return value has a bit set for
32   * each type definition that didn't verify, starting with the shortest
33   * length in the LSB.
34   *
35   * Example - you are interested only in integer sizes up to 32-bits.
36   *
37   * VerifySIZES(32) or VerifySIZES(BITSinDWORD);
38   *
39   * ============================================================
40   */
41
42  #ifndef BYTESdotH
43  #define BYTESdotH
44
45  #define BITSinBYTE  ( 8)
46  #define BITSinWORD  (16)
47  #define BITSinDWORD (32)
48  #define BITSinQWORD (64)
49
50  /* ============================================================
51   * Normal definitions
52   *
53   * This the only section that should need to be changed.
54   *
55   * Determine which integer type is the correct number of bits and update
56   * the following list. Do not worry about signed/unsigned.
57   *
58   * It is not recommended that you actually use these definitions in your
59   * code - they are simply used in the following type definitions.
60   * ============================================================
61   */
62
63  #define NBYTE  char
64  #define NWORD  short
65  #define NDWORD int
```

```
66 #define NQWORD long
67
68 /* ============================================================================
69  * UNSIGNED TYPE DEFINITIONS
70  * ============================================================================
71  */
72
73 typedef unsigned NBYTE   BYTE;
74 typedef unsigned NWORD   WORD;
75 typedef unsigned NDWORD  DWORD;
76 typedef unsigned NQWORD  QWORD;
77
78 /* ============================================================================
79  * SIGNED TYPE DEFINITIONS
80  * ============================================================================
81  */
82
83 typedef signed   NBYTE   SBYTE;
84 typedef signed   NWORD   SWORD;
85 typedef signed   NDWORD  SDWORD;
86 typedef signed   NQWORD  SQWORD;
87
88 /* ============================================================================
89  * UTILITY FUNCTIONS
90  * ============================================================================
91  */
92
93 unsigned int VerifySIZES(unsigned int maxlength);
94
95 #endif
```

C.2.9 modem.c and modem.h

```
1  /*****************************************************************************
2   * MODEM for the Real-time BBC Codec/Modem                                   *
3   *****************************************************************************
4   * William L. Bahn                                                           *
5   * Academy Center for Information Security                                   *
6   * Department of Computer Science                                            *
7   * United States Air Force Academy                                           *
8   * USAFA, CO 80840                                                           *
9   *****************************************************************************
10  * FILE:.......... modem.c                                                   *
11  * DATE CREATED:.... 06 SEP 07                                               *
12  * DATE MODIFIED:... 06 SEP 07                                               *
```

```
13   ***************************************************************************
14   *
15   * REVISION HISTORY
16   *
17   ***************************************************************************
18   *
19   * DESCRIPTION
20   *
21   * The modem and its public interface is described in modem.h.
22   *
23   ***************************************************************************
24   */
25
26  //---------------------------------------------------------------------------
27  // REQUIRED INCLUDES
28  //---------------------------------------------------------------------------
29
30  #include <stdlib.h> // malloc()
31  #include <math.h>   // exp()
32  #include "modem.h"
33
34  //---------------------------------------------------------------------------
35  // STRUCTURE DEFINITIONS
36  //---------------------------------------------------------------------------
37
38  // NOTE: Normally the structure definition would be in the *.c file to make
39  // the structure members inaccessible to outside functions except through
40  // public function calls. But for the real-time code it has been decided
41  // to make the structure members directly visible to the functions that
42  // manipulate them.
43
44  //---------------------------------------------------------------------------
45  // PUBLIC FUNCTION DEFINITIONS
46  //---------------------------------------------------------------------------
47
48  MODEM *MODEM_Del(MODEM *p)
49  {
50          if (p)
51          {
52                  free(p);
53          }
54          return NULL;
55  }
56
57  MODEM *MODEM_New(CONFIG *c, DWORD *errcode)
```

```
58  {
59          MODEM *p;
60          DWORD err;
61          double nominal_steady_state_peak;
62
63          p = NULL;
64          err = 0;
65
66          p = (MODEM *) malloc(sizeof(MODEM));
67          if (!p)
68                  err |= 1 << 0;
69
70          if (!err)
71          {
72                  // Derived quantities
73                  p->jitter_samples = (int)(c->modem_samples_per_bit * c->modem_jitter_bits);
74
75                  // Integrator parameter
76                  p->alpha = exp((2.0/c->modem_samples_per_bit) - 1.0);
77
78                  // Threshold parameters
79                  nominal_steady_state_peak = (c->nominal_rx_signal*c->nominal_rx_signal) *
                        (1.0/(1.0-p->alpha));
80                  p->t_hi = nominal_steady_state_peak * ((c->modem_threshold_pct + c->
                        modem_hysteresis_pct/2.0)/100.0);
81                  p->t_lo = nominal_steady_state_peak * ((c->modem_threshold_pct - c->
                        modem_hysteresis_pct/2.0)/100.0);
82
83                  // State information
84                  p->state = 0;
85                  p->integrator = 0.0;
86                  p->stamp = 0;
87          }
88
89          if (err)
90                  p = MODEM_Del(p);
91
92          if (c->diagnostics)
93          {
94                  // Diagnostic Report
95                  printf("-----------------------------------------\n");
96                  printf("MODEM\n");
97                  printf("  Creation:................. %s\n", ((err)? "FAILED":"SUCCEEDED"));
98                  printf("  Location:................. %p\n", (void *) p);
99                  printf("  Integrator alpha:......... %f\n", p->alpha);
```

```
100                 printf("  Jitter tolerance:.......... %f\n", p->jitter_samples);
101                 printf("  Modem gain:................ %f (%f dB)\n", c->nominal_tx_signal, c->
                        modem_gain_dB);
102                 printf("  Nominal channel loss:...... %f dB\n", c->modem_channel_loss_dB);
103                 printf("  Nominal rx signal peak:.... %f (%f dB)\n", c->nominal_rx_signal, (c->
                        modem_gain_dB-c->modem_channel_loss_dB));
104                 printf("  Nominal integrator peak:... %f\n", nominal_steady_state_peak);
105                 printf("  LO -> HI threshold:........ %f\n", p->t_hi);
106                 printf("  HI -> LO threshold:........ %f\n", p->t_lo);
107                 printf("-----------------------------------------\n");
108         }
109
110         *errcode = err;
111         return p;
112 }
113
114 //---------------------------------------------------------------------------
115
116 /* MODEM
117  *
118  * The MODEM reads/writes USRP in bursts of samples corresponding to
119  * 8 packet bits. The calling function is responsible for ensuring that
120  * valid data and/or sufficient room for new data exists in the buffer.
121  *
122  */
123
124 /* MODULATOR
125  *
126  * The modulator reads one byte of packet data from the buffer and generates
127  * USRP data for the entire set of 8 packet bits.
128  *
129  */
130
131 void Modulate(CONFIG *c, BUFFER *buffer, MODEM *modem, SINK *sink)
132 {
133         DWORD originbit, sample;
134         float signal;
135         clock_t ticks;
136         float *v;
137         ticks = clock();
138
139         // Push write pointer if packet byte is not available
140         if (!buffer->ready)
141         {
142                 buffer->write = (buffer->write + 1) & buffer->buffermask;
```

```
143                     buffer->ready++;
144                     buffer->margin--;
145             }
146
147             // For each bit in the packet byte at the buffer's read pointer
148             for (originbit = 0; originbit < 8; originbit++)
149             {
150                     // Determine if the bit is a mark or a space
151                     if (buffer->buffer[buffer->read] & c->bitmask[originbit])
152                     {
153                             c->marks++;
154                             signal = (float) c->nominal_tx_signal;
155                     }
156                     else
157                             signal = 0.0;
158
159                     // Determine if the sink can take all the samples for the present bit
160                     if (sink->samples + c->modem_samples_per_bit < sink->sample_limit)
161                     {
162                             // Establish the base location within the sink's buffer
163                             v = ((float *) sink->v) + (2 * sink->samples);
164
165                             // Generate and write the baseband samples to the sink
166                             for (sample = 0; sample < c->modem_samples_per_bit; sample++)
167                             {
168                                     v[2*sample]     = signal; // I(t) (actual data)
169                                     v[2*sample + 1] = 0.0;    // Q(t) (forced to zero)
170                             }
171                             sink->samples += c->modem_samples_per_bit;
172                     }
173                     else
174                             sink->streaming = FALSE;
175             }
176
177             buffer->buffer[buffer->read] = 0;
178             buffer->read = (buffer->read + 1) & buffer->buffermask;
179             buffer->ready--;
180             buffer->margin++;
181
182             c->actual_trx_bytes += c->trx_bytes_per_packet_byte;
183             c->dem_ticks += clock() - ticks;
184     }
185
186     void Demodulate(CONFIG *c, SOURCE *source, MODEM *modem, BUFFER *buf)
187     {
```

```
188         DWORD sample;
189         DWORD originbit;
190         clock_t ticks;
191         float *v;
192         double v2;
193
194         ticks = clock();
195
196         for (originbit = 0; originbit < 8; originbit++)
197         {
198                 v = ((float *) source->v) + (2 * source->samples);
199                 for (sample = 0; sample < c->modem_samples_per_bit; sample++)
200                 {
201                         if (source->samples < source->sample_limit)
202                         {
203
204                                 v2 = v[2*sample] * v[2*sample]  + v[2*sample+1] * v[2*sample+1];
205                                 source->samples++;
206                         }
207                         else
208                         {
209                                 v2 = 0;
210                                 source->streaming = FALSE;
211                         }
212
213                         modem->integrator = v2 + modem->alpha*(modem->integrator - v2);
214
215                         switch (modem->state)
216                         {
217                                 case 0:
218                                         if (modem->integrator > modem->t_hi)
219                                         {
220                                                 modem->state = 1;
221                                         }
222                                         break;
223                                 case 1:
224                                         if (modem->integrator < modem->t_lo)
225                                         {
226                                                 modem->state = 2;
227                                                 modem->stamp = (SDWORD) (sample + modem->
                                                       jitter_samples);
228                                         }
229                                         break;
230                                 case 2:
231                                         if (modem->integrator > modem->t_hi)
```

```
232                                                     modem->state = 1;
233                                             else
234                                                     if (((SDWORD) sample > modem->stamp)&&(modem->
                                                            integrator < modem->t_lo))
235                                                     {
236                                                             modem->state = 0;
237                                                     }
238                                             break;
239                             }
240                     }
241                     modem->stamp -= c->modem_samples_per_bit;
242
243                     if (0 == buf->empty)
244                     {
245                             buf->read = (buf->read + 1) & buf->buffermask;
246                             buf->empty++;
247                             buf->margin--;
248                             buf->overflows++;
249                     }
250
251                     // Step packet forward and mark next location
252                     if (modem->state > 0)
253                     {
254                             c->marks++;
255                             buf->buffer[buf->write] |= c->bitmask[originbit];
256                     }
257                     else
258                             buf->buffer[buf->write] &= ~c->bitmask[originbit];
259             }
260
261             buf->write = (buf->write + 1) & buf->buffermask;
262             buf->margin++;
263             buf->empty--;
264
265             c->actual_trx_bytes += c->trx_bytes_per_packet_byte;
266             c->dem_ticks += clock() - ticks;
267     }
268
269     //---------------------------------------------------------------------------
```

```
1    /*****************************************************************************
2     * MODEM for the Real-time BBC Codec/Modem                                   *
3     *****************************************************************************
4     * William L. Bahn                                                           *
5     * Academy Center for Information Security                                   *
```

```
6    * Department of Computer Science                                        *
7    * United States Air Force Academy                                       *
8    * USAFA, CO 80840                                                       *
9    ****************************************************************************
10   * FILE:.......... modem.h                                               *
11   * DATE CREATED:.... 06 SEP 07                                           *
12   * DATE MODIFIED:... 06 SEP 07                                           *
13   ****************************************************************************
14   *
15   * REVISION HISTORY
16   *
17   ****************************************************************************
18   *
19   * DESCRIPTION
20   *
21   * The modem converts baseband signal data to/from packet data.
22   *
23   */
24
25   #ifndef MODEMdotH
26   #define MODEMdotH
27
28   //---------------------------------------------------------------------------
29   // REQUIRED INCLUDES
30   //---------------------------------------------------------------------------
31
32   #include <time.h>  // clock_t
33
34   #include "config.h"
35   #include "source.h"
36   #include "buffer.h"
37   #include "sink.h"
38   #include "dirtyd.h"
39
40   //---------------------------------------------------------------------------
41   // STRUCTURE DECLARATIONS
42   //---------------------------------------------------------------------------
43
44   typedef struct MODEM MODEM;
45
46   //---------------------------------------------------------------------------
47   // STRUCTURE DEFINITIONS
48   //---------------------------------------------------------------------------
49
50   // NOTE: Normally the structure definition would be in the *.c file to make
```

```
51  // the structure members inaccessible to outside functions except through
52  // public function calls. But for the real-time code it has been decided
53  // to make the structure members directly visible to the functions that
54  // manipulate them.
55
56  struct MODEM
57  {
58          // Derived quantities
59          DWORD jitter_samples;
60          double alpha;
61          double t_hi, t_lo;
62
63          // State information
64          DWORD state;
65          double integrator;
66          SDWORD stamp;
67  };
68
69  //---------------------------------------------------------------------------
70  // PUBLIC FUNCTION PROTOTYPES
71  //---------------------------------------------------------------------------
72
73  MODEM *MODEM_Del(MODEM *p);
74  MODEM *MODEM_New(CONFIG *c, DWORD *errcode);
75  void Modulate(CONFIG *c, BUFFER *buffer, MODEM *modem, SINK *sink);
76  void Demodulate(CONFIG *c, SOURCE *source, MODEM *modem, BUFFER *buf);
77
78  //---------------------------------------------------------------------------
79  #endif
```

C.3 USRP Radio Scripts

C.3.1 USRP BBC Transmitter

```
1  #!/usr/bin/env python
2  #Adapted from a GNU Radio TX script by Mark Kuhr.
3
4  # The file format is complex IQ data pairs where both values are IEEE
5  # single-precision floating point numbers in little endian format.
6  # The first value is I and the second value is Q. The data is present
7  # only on the I data. The Q data is all zeros.
8
9  from gnuradio import gr, gru
10 from gnuradio import usrp
```

```
11  from gnuradio.eng_option import eng_option
12  from gnuradio import eng_notation
13  from gnuradio.eng_notation import num_to_str, str_to_num
14  from optparse import OptionParser
15  import sys
16  import time
17
18
19  class bbc_tx_graph(gr.top_block):
20      def __init__ (self, usb_num, sink_path, sink_path_B, nchan): #included usb_num from parameter
            list to define which usb
21
22          gr.top_block.__init__(self)
23              #default interpolator rate
24          self.interp = 64
25
26          if(nchan==1):
27              self.txfile = gr.file_source (gr.sizeof_gr_complex, sink_path, 1)
28              self.usrp = usrp.sink_c (usb_num, self.interp) # change from 0 to 1 if necessary
29              self.connect (self.txfile, self.usrp)
30          else:  #more than 1 channel, TX on both daughterboards
31              self.txfileA = gr.file_source (gr.sizeof_gr_complex, sink_path, 1)
32              self.txfileB = gr.file_source (gr.sizeof_gr_complex, sink_path_B, 1)
33
34              self.usrp = usrp.sink_c(which=usb_num, interp_rate=self.interp,nchan=nchan)
35              #do connect
36              intl = gr.interleave(gr.sizeof_gr_complex)
37              self.connect(self.txfileA, (intl, 0))
38              self.connect(self.txfileB, (intl, 1))
39              self.connect(intl,self.usrp)
40
41      def usb_freq (self):
42          return self.usrp.dac_freq() / self.interp
43
44      def usb_throughput (self):
45          return self.usb_freq () * 4
46
47      def set_interpolator (self, interp):
48          self.interp = interp
49          self.usrp.set_interp_rate (interp)
50
51      def set_freq_single(self, target_freq):
52          """
53          Set the center frequency we're interested in.
54
```

```
55          @param target_freq: frequency in Hz
56          @rypte: bool
57
58          Tuning is a two step process.  First we ask the front-end to
59          tune as close to the desired frequency as it can.  Then we use
60          the result of that operation and our target_frequency to
61          determine the value for the digital up converter.
62          """
63          r = self.usrp.tune(self.subdev._which, self.subdev, target_freq)
64          if r:
65              print "r.baseband_freq =", eng_notation.num_to_str(r.baseband_freq)
66              print "r.dxc_freq      =", eng_notation.num_to_str(r.dxc_freq)
67              print "r.residual_freq =", eng_notation.num_to_str(r.residual_freq)
68              print "r.inverted      =", r.inverted
69              print "  OK"
70              return True
71
72          return False
73
74      def set_freq_multi(self, side, target_freq):
75          """
76          Set the center frequency we're interested in.
77
78          @param side: 0 = side A, 1 = side B
79          @param target_freq: frequency in Hz
80          @rtype: bool
81
82          Tuning is a two step process.  First we ask the front-end to
83          tune as close to the desired frequency as it can.  Then we use
84          the result of that operation and our target_frequency to
85          determine the value for the digital up converter.
86          """
87
88          print "Tuning side %s to %sHz" % (("A", "B")[side], num_to_str(target_freq))
89          r = self.usrp.tune(self.subdev[side]._which, self.subdev[side], target_freq)
90          if r:
91              print "  r.baseband_freq =", num_to_str(r.baseband_freq)
92              print "  r.dxc_freq      =", num_to_str(r.dxc_freq)
93              print "  r.residual_freq =", num_to_str(r.residual_freq)
94              print "  r.inverted      =", r.inverted
95              print "  OK"
96              return True
97
98          else:
99              print "  Failed!"
```

```
100
101         return False
102
103 def main ():
104     parser = OptionParser (option_class=eng_option)
105     parser.add_option ("-T", "--tx_subdev_spec", type="subdev", default=(0, 0),
106                        help="select USRP Tx side A or B (may also use A:0 or A:1 format)")
107     parser.add_option ("-f", "--rf_freq", type="eng_float", default=None,
108                        help="set RF center frequency to FREQ")
109     parser.add_option ("-i", "--interp", type="int", default=64,
110                        help="set fgpa interpolation rate to INTERP")
111     parser.add_option("-U", "--usb_num", type="int", default=0,
112                        help="select USRP USB location 0 or 1 (default=0)")
113     parser.add_option("-C", "--nchan", type="int", default=1,
114                        help="select number of channels for TX (1 or 2) (default=1)")
115     parser.add_option("-S", "--sink_path",type="string",default=None, help="set sink file path for
          transmission 1")
116     parser.add_option("-N", "--sink_path_B",type="string",default=None, help="set sink file path
         for transmission 2")
117     parser.add_option("-P", "--tx_subdev_spec_B", type="subdev", default=(0, 0),
118                        help="select USRP Tx side A or B (may also use A:0 or A:1 format)")
119     (options, args) = parser.parse_args ()
120
121     if len(args) != 0:
122         parser.print_help()
123         raise SystemExit
124
125     if options.rf_freq is None:
126         sys.stderr.write("usrp_siggen: must specify RF center frequency with -f RF_FREQ\n")
127         parser.print_help()
128         raise SystemExit
129
130
131     fg = bbc_tx_graph(options.usb_num,options.sink_path, options.sink_path_B, options.nchan)
132
133     fg.set_interpolator (options.interp)
134
135     print "Using USB Port %d" % (options.usb_num)
136     print "Sink path: %s" % (options.sink_path)
137     if(options.sink_path_B):
138         print "Sink path B: %s" % (options.sink_path_B)
139
140     if(options.nchan == 1):
141         print "Using %d Channel" % (options.nchan)
142         # determine the daughterboard subdevice we're using
```

```
143             if options.tx_subdev_spec is None:
144                 options.tx_subdev_spec = usrp.pick_tx_subdevice(fg.u)
145 
146             m = usrp.determine_tx_mux_value(fg.usrp, options.tx_subdev_spec)
147             #print "mux = %#04x" % (m,)
148             fg.usrp.set_mux(m)
149             #fg.usrp.set_mux(0xba98)
150 
151             fg.subdev = usrp.selected_subdev(fg.usrp, options.tx_subdev_spec)
152             print "Using TX daughterboard %s" % (fg.subdev.side_and_name(),)
153 
154             fg.subdev.set_gain(fg.subdev.gain_range()[1])    # set max Tx gain
155 
156             if not fg.set_freq_single(options.rf_freq):
157                 sys.stderr.write('Failed to set RF frequency\n')
158                 raise SystemExit
159 
160             fg.subdev.set_enable(True)# enable transmitter
161 
162         else:  # we're using both daughterboard slots, thus subdev is a 2-tuple
163             print "Using %d Channels" % (options.nchan)
164             fg.subdev = (fg.usrp.db[0][0], fg.usrp.db[1][0])
165             print "Using TX daughterboard %s" % (fg.subdev[0].side_and_name(),)
166             print "Using TX daughterboard %s" % (fg.subdev[1].side_and_name(),)
167 
168 
169             #m_A = usrp.determine_tx_mux_value(fg.usrp, options.tx_subdev_spec)
170             #m_B = usrp.determine_tx_mux_value(fg.usrp, options.tx_subdev_spec_B)
171             #print "mux = %#04x" % (m,)
172             #fg.subdev[0].set_mux(m_A)
173             #fg.subdev[1].set_mux(m_B)
174             #fg.usrp.set_mux(m_A)
175             fg.usrp.set_mux(gru.hexint(0xBA98))
176 
177             fg.subdev[0].set_gain(fg.subdev[0].gain_range()[1])    # set max Tx gain
178             fg.subdev[1].set_gain(fg.subdev[1].gain_range()[1])    # set max Tx gain
179 
180             #use same frequency for both transmitters
181             fg.set_freq_multi(0, options.rf_freq)
182             fg.set_freq_multi(1, options.rf_freq)
183             #fg.subdev[0].set_freq(options.rf_freq)
184             #fg.subdev[1].set_freq(options.rf_freq)
185 
186             fg.subdev[0].set_enable(True) # enable transmitter
187             fg.subdev[1].set_enable(True) # enable transmitter
```

```
188
189     try:
190         #fg.run()
191         print "Starting Transmission\n"
192         fg.start()
193         time.sleep(1)
194         fg.stop()
195         print "Transmission Completed.\n"
196     except KeyboardInterrupt:
197         #pass
198         fg.stop()
199
200 if __name__ == '__main__':
201     main ()
```

C.3.2 USRP BBC Receiver

```
1  #!/usr/bin/env python
2
3  #Adapted from a GNU Radio RX (rx_cfile.py) script by Mark Kuhr.
4
5  """
6  Read samples from the USRP and write to file formatted as binary
7  outputs single precision complex float values or complex short values (interleaved 16 bit signed
       short integers).
8
9  """
10
11 from gnuradio import gr, gru, eng_notation
12 #from gnuradio import audio
13 from gnuradio import usrp
14 from gnuradio.eng_option import eng_option
15 from optparse import OptionParser
16 from usrpm import usrp_dbid
17 import time
18 import sys
19
20 class my_graph(gr.flow_graph):
21 #class my_graph(gr.top_block):
22
23     def __init__(self):
24         gr.flow_graph.__init__(self)
25         #gr.top_block.__init__(self)
26
27
```

```
28          usage="%prog: [options] output_filename output_filename2"
29          parser = OptionParser(option_class=eng_option, usage=usage)
30          parser.add_option("-R", "--rx-subdev-spec", type="subdev", default=(0, 0),
31                            help="select USRP Rx side A or B (default=A)")
32          parser.add_option("-U", "--usb_num", type="int", default=0,
33                            help="select USRP USB location 0 or 1 (default=0)")
34          parser.add_option("-d", "--decim", type="int", default=16,
35                            help="set fgpa decimation rate to DECIM [default=%default]")
36          parser.add_option("-f", "--freq", type="eng_float", default=None,
37                            help="set frequency to FREQ", metavar="FREQ")
38          parser.add_option("-g", "--gain", type="eng_float", default=None,
39                            help="set gain in dB (default is midpoint)")
40          parser.add_option("-8", "--width-8", action="store_true", default=False,
41                            help="Enable 8-bit samples across USB")
42          parser.add_option( "--no-hb", action="store_true", default=False,
43                            help="don't use halfband filter in usrp")
44          parser.add_option( "-s","--output-shorts", action="store_true", default=False,
45                            help="output interleaved shorts in stead of complex floats")
46          parser.add_option("-N", "--nsamples", type="eng_float", default=None,
47                            help="number of samples to collect [default=+inf]")
48          parser.add_option("-C", "--nchan", type="int", default=1,
49                            help="set number of channels to use (RX on both daughterboards)")
50          (options, args) = parser.parse_args ()
51
52          if len(args) < 1:
53              parser.print_help()
54              raise SystemExit, 1
55
56          #with multiple channels, need multiple files for receiver sinks so both receivers are not
57          #     writing to the same file on the driver computer
58          if options.nchan > 1:
59              filename_A = args[0]
60              filename_B = args[1]
61          else:
62              filename = args[0]
63
64          if options.freq is None:
65              parser.print_help()
66              sys.stderr.write('You must specify the frequency with -f FREQ\n');
67              raise SystemExit, 1
68
69          if options.no_hb or (options.decim<8):
70              self.fpga_filename="std_4rx_0tx.rbf" #Min decimation of this firmware is 4. contains 4
                    Rx paths without halfbands and 0 tx paths.
71              if options.output_shorts:
```

```
72                  self.u = usrp.source_s(which=options.usb_num,decim_rate=options.decim,
                        fpga_filename=self.fpga_filename)
73              else:
74                  self.u = usrp.source_c(which=options.usb_num,decim_rate=options.decim,
                        fpga_filename=self.fpga_filename)
75          else:
76              #standard fpga firmware "std_2rxhb_2tx.rbf" contains 2 Rx paths with halfband filters
                    and 2 tx paths (the default) min decimation 8
77              if options.output_shorts:
78                  self.u = usrp.source_s(which=options.usb_num,decim_rate=options.decim)
79              else:
80                  self.u = usrp.source_c(which=options.usb_num,decim_rate=options.decim)
81
82          #use more than 1 channel if specified
83          #this will allow a USRP to TX or RX on both daughterboards simultaneously
84          if options.nchan > 1:
85              nchan = options.nchan
86              if self.u.nddc() < nchan:
87                  sys.stderr.write('This code requires an FPGA build with %d DDCs.  This FPGA has
                        only %d.\n' % (nchan, self.u.nddc()))
88                  raise SystemExit
89
90              if not self.u.set_nchannels(nchan):
91                  sys.stderr.write('set_nchannels(%d) failed\n' % (nchan,))
92                  raise SystemExit
93
94              #self.subdev = self.u.db[0] + self.u.db[1]
95
96              self.subdev = (self.u.db[0][0], self.u.db[1][0])
97
98              print "Using RX daughterboard %s" % (self.subdev[0].side_and_name(),)
99              print "Using RX daughterboard %s" % (self.subdev[1].side_and_name(),)
100
101             if options.gain is None:
102                 g_A = self.subdev[0].gain_range()
103                 options.gain = float(g_A[0]+g_A[1])/2
104
105             #use the same gain for both sides
106             self.subdev[0].set_gain(options.gain)
107             self.subdev[1].set_gain(options.gain)
108             # r = usrp.tune(self.u, i, self.subdev[i], target_freq)
109             r_A = self.u.tune(0,self.subdev[0],options.freq)
110             if not r_A:
111                 sys.stderr.write('Failed to set frequency for RX daughterboard %s\n' % (self.
                        subdev[0].side_and_name()))
```

```
112                 raise SystemExit, 1
113
114             r_B = self.u.tune(1,self.subdev[1],options.freq)
115             if not r_B:
116                 sys.stderr.write('Failed to set frequency for RX daughterboard %s\n' % (self.
                        subdev[1].side_and_name()))
117                 raise SystemExit, 1
118
119         else:
120             # using only 1 channel in this case
121             # determine the daughterboard subdevice we're using per argument list
122             self.subdev = usrp.selected_subdev(self.u, options.rx_subdev_spec)
123             print "Using RX daughterboard %s" % (self.subdev.side_and_name(),)
124
125             #set the gain
126             if options.gain is None:
127                 # if no gain was specified, use the mid-point in dB
128                 g = self.subdev.gain_range()
129                 options.gain = float(g[0]+g[1])/2
130
131             self.subdev.set_gain(options.gain)
132
133             r = self.u.tune(0, self.subdev, options.freq)
134             if not r:
135                 sys.stderr.write('Failed to set frequency\n')
136                 raise SystemExit, 1
137
138         if options.width_8:
139             sample_width = 8
140             sample_shift = 8
141             format = self.u.make_format(sample_width, sample_shift)
142             r = self.u.set_format(format)
143         if options.output_shorts:
144             #default value is fine here for multiple channels since
145             #we will be using complex floats
146             self.dst = gr.file_sink(gr.sizeof_short, filename)
147         else:
148             if options.nchan == 1:
149                 self.dst = gr.file_sink(gr.sizeof_gr_complex, filename)
150             else:
151                 #establish separate file sinks for the two channels
152                 self.dst_A = gr.file_sink(gr.sizeof_gr_complex, filename_A)
153                 self.dst_B = gr.file_sink(gr.sizeof_gr_complex, filename_B)
154
155         if options.nsamples is None:#this is the default
```

```
156             if options.nchan == 1:
157                 self.connect(self.u, self.dst)
158             else:  #multiple channels
159                 di = gr.deinterleave(gr.sizeof_gr_complex)
160                 self.connect(self.u, di)
161                 self.connect((di,0),self.dst_A)
162                 self.connect((di,1),self.dst_B)
163 
164         else:
165             if options.output_shorts:
166               self.head = gr.head(gr.sizeof_short, int(options.nsamples)*2)
167             else:
168               self.head = gr.head(gr.sizeof_gr_complex, int(options.nsamples))
169             self.connect(self.u, self.head, self.dst)
170 
171         if options.rx_subdev_spec is None:
172             options.rx_subdev_spec = usrp.pick_rx_subdevice(self.u)
173 
174         self.u.set_mux(usrp.determine_rx_mux_value(self.u, options.rx_subdev_spec))
175         #self.u.set_mux(gru.hexint(0xf3f2f1f0))
176 
177         #PRINT STATEMENTS
178         print "Using USB Port %d" % (options.usb_num)
179         if(options.nchan > 1):
180             print "Using %d Channels" % (options.nchan)
181         else:
182             print "Using %d Channel" % (options.nchan)
183 
184         #display USB sample rate
185         input_rate = self.u.adc_freq() / self.u.decim_rate()
186         print "USB sample rate %s" % (eng_notation.num_to_str(input_rate))
187 
188 
189 if __name__ == '__main__':
190     try:
191 #         my_graph().run()
192         #run for 10 seconds
193         receive = my_graph()
194 
195         receive.start()
196         time.sleep(15)
197         receive.stop()
198         print "Receiving Complete."
199 
200 
```

```
201     except KeyboardInterrupt:
202         # pass
203         receive.stop()
```

C.3.3 USRP Signal Jammer

```
1  #!/usr/bin/env python
2  #Adapted from a GNU Radio script (siggen.py) by Mark Kuhr.
3
4  from gnuradio import gr, gru
5  from gnuradio import usrp
6  from gnuradio.eng_option import eng_option
7  from gnuradio import eng_notation
8  from optparse import OptionParser
9  import sys
10
11
12 class my_top_block(gr.top_block):
13     def __init__ (self,usb_num):
14         gr.top_block.__init__(self)
15
16         # controllable values
17         self.interp = 64
18         self.waveform_type = gr.GR_SIN_WAVE
19         self.waveform_ampl = 16000
20         self.waveform_freq = 100.12345e3
21         self.waveform_offset = 0
22         self._instantiate_blocks (usb_num)
23         self.set_waveform_type (self.waveform_type)
24
25     def usb_freq (self):
26         return self.u.dac_freq() / self.interp
27
28     def usb_throughput (self):
29         return self.usb_freq () * 4
30
31     def set_waveform_type (self, type):
32         '''
33         valid waveform types are: gr.GR_SIN_WAVE, gr.GR_CONST_WAVE,
34         gr.GR_UNIFORM and gr.GR_GAUSSIAN
35         '''
36         self._configure_graph (type)
37         self.waveform_type = type
38
39     def set_waveform_ampl (self, ampl):
```

```
40          self.waveform_ampl = ampl
41          self.siggen.set_amplitude (ampl)
42          self.noisegen.set_amplitude (ampl)
43
44      def set_waveform_freq (self, freq):
45          self.waveform_freq = freq
46          self.siggen.set_frequency (freq)
47
48      def set_waveform_offset (self, offset):
49          self.waveform_offset = offset
50          self.siggen.set_offset (offset)
51
52      def set_interpolator (self, interp):
53          self.interp = interp
54          self.siggen.set_sampling_freq (self.usb_freq ())
55          self.u.set_interp_rate (interp)
56
57      def _instantiate_blocks (self,usb_num):
58          self.src = None
59          self.u = usrp.sink_c (usb_num, self.interp)
60
61          self.siggen = gr.sig_source_c (self.usb_freq (),
62                                         gr.GR_SIN_WAVE,
63                                         self.waveform_freq,
64                                         self.waveform_ampl,
65                                         self.waveform_offset)
66
67          self.noisegen = gr.noise_source_c (gr.GR_UNIFORM,
68                                             self.waveform_ampl)
69
70          # self.file_sink = gr.file_sink (gr.sizeof_gr_complex, "siggen.dat")
71
72      def _configure_graph (self, type):
73          try:
74              self.lock()
75              self.disconnect_all ()
76              if type == gr.GR_SIN_WAVE or type == gr.GR_CONST_WAVE:
77                  self.connect (self.siggen, self.u)
78                  # self.connect (self.siggen, self.file_sink)
79                  self.siggen.set_waveform (type)
80                  self.src = self.siggen
81              elif type == gr.GR_UNIFORM or type == gr.GR_GAUSSIAN:
82                  self.connect (self.noisegen, self.u)
83                  self.noisegen.set_type (type)
84                  self.src = self.noisegen
```

```
85              else:
86                  raise ValueError, type
87          finally:
88              self.unlock()
89
90      def set_freq(self, target_freq):
91          """
92          Set the center frequency we're interested in.
93
94          @param target_freq: frequency in Hz
95          @rypte: bool
96
97          Tuning is a two step process.  First we ask the front-end to
98          tune as close to the desired frequency as it can.  Then we use
99          the result of that operation and our target_frequency to
100         determine the value for the digital up converter.
101         """
102         r = self.u.tune(self.subdev._which, self.subdev, target_freq)
103         if r:
104             print "r.baseband_freq =", eng_notation.num_to_str(r.baseband_freq)
105             print "r.dxc_freq      =", eng_notation.num_to_str(r.dxc_freq)
106             print "r.residual_freq =", eng_notation.num_to_str(r.residual_freq)
107             print "r.inverted      =", r.inverted
108             print "    OK     "
109             return True
110
111         return False
112
113
114
115 def main ():
116     parser = OptionParser (option_class=eng_option)
117     parser.add_option ("-T", "--tx-subdev-spec", type="subdev", default=(0, 0),
118                        help="select USRP Tx side A or B")
119     parser.add_option ("-f", "--rf-freq", type="eng_float", default=None,
120                        help="set RF center frequency to FREQ")
121     parser.add_option ("-i", "--interp", type="int", default=64,
122                        help="set fgpa interpolation rate to INTERP [default=%default]")
123     parser.add_option("-U", "--usb_num", type="int", default=0,
124                           help="select USRP USB location 0 or 1 (default=0)")
125
126     parser.add_option ("--sine", dest="type", action="store_const", const=gr.GR_SIN_WAVE,
127                        help="generate a complex sinusoid [default]", default=gr.GR_SIN_WAVE)
128     parser.add_option ("--const", dest="type", action="store_const", const=gr.GR_CONST_WAVE,
129                        help="generate a constant output")
```

```
130     parser.add_option ("--gaussian", dest="type", action="store_const", const=gr.GR_GAUSSIAN,
131                        help="generate Gaussian random output")
132     parser.add_option ("--uniform", dest="type", action="store_const", const=gr.GR_UNIFORM,
133                        help="generate Uniform random output")
134
135     parser.add_option ("-w", "--waveform-freq", type="eng_float", default=100e3,
136                        help="set waveform frequency to FREQ [default=%default]")
137     parser.add_option ("-a", "--amplitude", type="eng_float", default=16e3,
138                        help="set waveform amplitude to AMPLITUDE [default=%default]", metavar="
                               AMPL")
139     parser.add_option ("-g", "--gain", type="eng_float", default=None,
140                        help="set output gain to GAIN [default=%default]")
141     parser.add_option ("-o", "--offset", type="eng_float", default=0,
142                        help="set waveform offset to OFFSET [default=%default]")
143     (options, args) = parser.parse_args ()
144
145     if len(args) != 0:
146         parser.print_help()
147         raise SystemExit
148
149     if options.rf_freq is None:
150         sys.stderr.write("usrp_siggen: must specify RF center frequency with -f RF_FREQ\n")
151         parser.print_help()
152         raise SystemExit
153
154     tb = my_top_block(options.usb_num)
155     tb.set_interpolator (options.interp)
156     tb.set_waveform_type (options.type)
157     tb.set_waveform_freq (options.waveform_freq)
158     tb.set_waveform_ampl (options.amplitude)
159     tb.set_waveform_offset (options.offset)
160
161     # determine the daughterboard subdevice we're using
162     if options.tx_subdev_spec is None:
163         options.tx_subdev_spec = usrp.pick_tx_subdevice(tb.u)
164
165     m = usrp.determine_tx_mux_value(tb.u, options.tx_subdev_spec)
166     #print "mux = %#04x" % (m,)
167     tb.u.set_mux(m)
168     tb.subdev = usrp.selected_subdev(tb.u, options.tx_subdev_spec)
169     print "Using TX daughterboard %s" % (tb.subdev.side_and_name(),)
170     print "Using USB Port %d" % (options.usb_num)
171         #print "USB Throughput %d" % ()
172
173     #if(options.type == "gaussian"):
```

```
174     #    print "Generating Gaussian Random Noise"
175
176     if options.gain is None:
177         tb.subdev.set_gain(tb.subdev.gain_range()[1])    # set max Tx gain
178         print "Using Gain of %d" % (tb.subdev.gain_range()[1])
179     else:
180         tb.subdev.set_gain(options.gain)    # set Tx gain
181         print "Using Gain of %d" % (options.gain)
182
183
184     if not tb.set_freq(options.rf_freq):
185         sys.stderr.write('Failed to set RF frequency\n')
186         raise SystemExit
187
188     tb.subdev.set_enable(True)                       # enable transmitter
189
190     try:
191         tb.run()
192     except KeyboardInterrupt:
193         pass
194
195 if __name__ == '__main__':
196     main ()
```

C.3.4 USRP RSSI Measurement

```
1  #!/usr/bin/env python
2
3  #Adapted from a GNU Radio script (spectrum_sense.py) by Mark Kuhr.
4
5  """
6  Read samples from the USRP and write to file formatted as binary
7  outputs single precision complex float values or complex short values (interleaved 16 bit signed
       short integers).
8
9  """
10
11 from gnuradio import gr, gru, eng_notation
12 #from gnuradio import audio
13 from gnuradio import usrp
14 from gnuradio.eng_option import eng_option
15 from optparse import OptionParser
16 from usrpm import usrp_dbid
17 import time
18 import sys
```

```
19
20 class my_graph(gr.flow_graph):
21 #class my_graph(gr.top_block):
22
23     def __init__(self):
24         gr.flow_graph.__init__(self)
25         #gr.top_block.__init__(self)
26
27
28         usage="%prog: [options] "
29         parser = OptionParser(option_class=eng_option, usage=usage)
30         parser.add_option("-R", "--rx-subdev-spec", type="subdev", default=(0, 0),
31                           help="select USRP Rx side A or B (default=A)")
32         parser.add_option("-U", "--usb_num", type="int", default=0,
33                           help="select USRP USB location 0 or 1 (default=0)")
34         parser.add_option("-f", "--freq", type="eng_float", default=None,
35                           help="set frequency to FREQ", metavar="FREQ")
36         parser.add_option("-g", "--gain", type="eng_float", default=None,
37                           help="set gain in dB (default is midpoint)")
38         parser.add_option("-8", "--width-8", action="store_true", default=False,
39                           help="Enable 8-bit samples across USB")
40         parser.add_option("-d", "--decim", type="int", default=16,
41                           help="set fgpa decimation rate to DECIM [default=%default]")
42
43         (options, args) = parser.parse_args ()
44
45
46
47         if options.freq is None:
48             parser.print_help()
49             sys.stderr.write('You must specify the frequency with -f FREQ\n');
50             raise SystemExit, 1
51
52         self.u = usrp.source_c(which=options.usb_num,decim_rate=options.decim)
53
54         # determine the daughterboard subdevice we're using per argument list
55         self.subdev = usrp.selected_subdev(self.u, options.rx_subdev_spec)
56         #print "Using RX daughterboard %s" % (self.subdev.side_and_name(),)
57
58         #set the gain
59         if options.gain is None:
60             # if no gain was specified, use the mid-point in dB
61             g = self.subdev.gain_range()
62             options.gain = float(g[0]+g[1])/2
63
```

```
64          self.subdev.set_gain(options.gain)
65
66          r = self.u.tune(0, self.subdev, options.freq)
67          if not r:
68              sys.stderr.write('Failed to set frequency\n')
69              raise SystemExit, 1
70          #if r:
71              #print "r.baseband_freq =", eng_notation.num_to_str(r.baseband_freq)
72              #print "r.dxc_freq      =", eng_notation.num_to_str(r.dxc_freq)
73              #print "r.residual_freq =", eng_notation.num_to_str(r.residual_freq)
74              #print "r.inverted      =", r.inverted
75              #print "  OK"
76
77          if options.width_8:
78              sample_width = 8
79              sample_shift = 8
80              format = self.u.make_format(sample_width, sample_shift)
81              r = self.u.set_format(format)
82
83
84          if options.rx_subdev_spec is None:
85              options.rx_subdev_spec = usrp.pick_rx_subdevice(self.u)
86
87          self.u.set_mux(usrp.determine_rx_mux_value(self.u, options.rx_subdev_spec))
88          #self.u.set_mux(gru.hexint(0xf3f2f1f0))
89
90          #PRINT STATEMENTS
91          #print "Using USB Port %d" % (options.usb_num)
92
93          #display USB sample rate
94          input_rate = self.u.adc_freq() / self.u.decim_rate()
95          #print "USB sample rate %s" % (eng_notation.num_to_str(input_rate))
96
97
98          if(self.subdev.side_and_name() == "A: Flex 1200 Rx MIMO B"):
99              subdev = 0
100         else:
101             subdev = 1
102
103         #print "subdev = %d" % (subdev)
104
105         sample = 3000
106         rssi_sum = 0
107         for i in range (sample):
108             rssi = self.u.read_aux_adc(subdev, 0)
```

```
109                 #print rssi
110                 rssi_sum += rssi
111             #print "average: ",(rssi_sum/sample)
112             #return average RSSI value to calling script
113             #return (rssi_sum/sample)
114             print (rssi_sum/sample)
115             raise SystemExit, 1
116 
117 if __name__ == '__main__':
118     try:
119         my_graph().run()
120         #run for 10 seconds
121 #         receive = my_graph()
122 #         receive.start()
123 #         time.sleep(120)
124 #         receive.stop()
125 
126 
127     except KeyboardInterrupt:
128         pass
129         #receive.stop()
```