2011

Experimental study of cognitive radio test-bed using USRP

Venkat vinod Patcha
Louisiana State University and Agricultural and Mechanical College

EXPERIMENTAL STUDY OF COGNITIVE RADIO TEST-BED USING USRP

Thesis

Submitted to the Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by
Venkat vinod Patcha
B.TECH in Electronics and Communication Engineering
Padmasri Dr. B.V. Raju Institute of Technology (Affiliated to JNTU), India, 2008.
August, 2011

Acknowledgments

First of all, I would like to express sincere gratitude to my adviser Dr. Shuangqing Wei for his constant support in building my thesis. I would like to thank him for introducing me to the field of Wireless Communication and Cognitive Radio. His throughout guidance and motivation in solving complex problems has provided encouragement, enthusiasm and support, while the knowledge acquired by working with him has provided deep understanding of issues that are apart from classroom work. Also, I would like to thank my Co-adviser Dr. Rajgopal Kannan, whose support has provided valuable experience. I would thank Dr. Xue-bin Liang for being a member of thesis committee.

Apart from all, my parents and elder sister have provided the emotional and financial support to build my confidence and without which this wouldn't be possible. I would like to thank my friends and roommates for their support and morale boost during the hard times. Also, I would like to mention my lab-mates for giving their resources and time in enabling to complete my thesis.

Table of Contents

List of Tables

List of Figures

Abstract

Cognitive Radio is an emerging technology that enables efficient utilization of the spectrum. As such, it has created great interests in industrial and research fields. Many people have proposed test-bed models for the performance analysis of primary and secondary users in a real-time noise environment. However, these test-beds are generally lacking in their range of capabilities as well as accurate implementation of the proposed models. In this thesis, we develop our test-bed on USRP to achieve the spectrum sensing and co-existence of primary and secondary users, while implementing the rendezvous protocols for secondary traffic coordination.

We first demonstrate the spectrum sensing on the primary users using an energy detector(Average periodogram analysis) to obtain the average power of the primary channel under two different channel conditions (busy or idle). The focus is extended on developing the Markov traffic model and the Coded OFDM transceivers, while discussing the practical limitations for Markov traffic and viable solutions for reducing the burst errors for Coded OFDM . Finally, a four-node test-bed model of primary and secondary users is analyzed with the interference metrics(packet loss and error rate) for different scenario's. Also, the throughput and the interference metrics are compared for different rendezvous protocols of the secondary users.

Chapter 1

Introduction

Wireless Communication has been a rapidly growing sector in the communication industry that led to rise of market share and economic growth. In fact, the growth of wireless communication has created new technologies, which provide services with high bandwidth. From the beginning of the 20th century, there has been an increase in the usage of radio spectrum in areas such as Mobile communication, Wireless Local Area Network (WLAN), Blue-tooth and Cordless phones. The Federal Communication Commission (FCC), which manages the radio frequency spectrum and its usage, has published a report on the precise usage of the spectrum and the rise of unlicensed users that could cause interference to the licensed users [5]. In addition, the report has stressed on frequency bands that are heavily occupied all the time or partially occupied or vacant most of the time [5]. The Figure 1.1 signifies the usage of spectrum bands over a certain period. The scarcity of spectrum and need for efficient usage has led to the development of a new field "Cognitive Radio".
The term was first coined by Joseph Mitola in his thesis that emphasizes the efficient usage of spectrum and its allocation [9]. Cognitive Radio (CR) is a type of radio, that is aware of its environment, in which the radio adapts its transmission for efficient usage of the underutilized spectrum. The central idea of CR is to allow the unlicensed bands of the Secondary Users(SU), such as WLAN and Blue-tooth to utilize the licensed bands of the Primary Users(PU), such as TV and mobile without interference. The main features of cognitive radio are intelligence, adaptivity, reliability and efficiency [8]. It is a key contribution to the 4th generation (4G) technology for the efficient spectrum utilization [9]. The practical implementation of CR has been achieved by Software Defined Radio (SDR). SDR is a radio communication system in which the components implemented on hardware are transferred to a software program that provides the flexibility of changing the operating parameters of the device. The primary advantage of SDR is to use them over wide RF bandwidth and to perform Intermediate frequency(IF) functions inside the processor. In fact, Cognitive Radio is defined as "intelligence" that sits above SDR and lets SDR determine which mode of operation and parameter to use [19]. As SDR handsets can

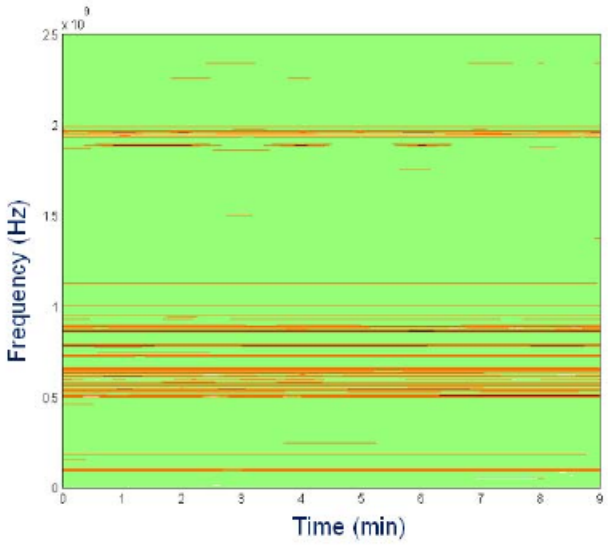

Figure 1.1: Frequency Usage of Spectrum Bands [9]

easily be re-configured to different wireless broadband technologies, they can be utilized to implement a CR Test-bed. The flexibility, robustness and efficiency of SDR has resulted in practical experimentation over RF world.

1.1 Motivation

Many features of Cognitive Radio have attracted researchers, while studies are conducted for efficient utilization. The flexibility and growth of SDR has provided ease of implementation of a Cognitive Radio test-bed. They provide the ease of implementation by using software to build a network and to modify the network based on the availability. The major contribution in our thesis is to create a test-bed model for evaluating the spectrum access and co-existence of primary and secondary users and study their performance analysis, while emphasizing the usage of existing rendezvous protocols for secondary users(SU) to reduce interference effects and provide high throughput.

1.2 Contribution of Thesis

- Energy detection methods are useful for calculating the energy statistics of the primary user that determine the presence or absence of primary traffic. We perform the energy detection of PU's using average periodogram analysis. The curve-fitting

functions, PSD, histogram and density plots of average power are evaluated under each hypothesis(presence and absence of primary traffic). Also, the histogram plots of average power are examined for varying time windows and windowing functions.

- Developed Markov traffic model consisting of ON and OFF cycles with ON cycles that are uniformly distributed and OFF cycles that are dependent on their former ON periods. Also, the practical problems with transition times for switching between ON and OFF cycles are discussed.

- Implementing Coded OFDM model using the OFDM and Trellis convolution blocks in GNU Radio. In fact, the mathematical model is demonstrated for the current OFDM and trellis blocks to develop the working model. The problems encountered in the implementation are discussed.Moreover, we have provided solutions to resolve big burst of errors in Coded OFDM.

- Experimental analysis is performed on four-node test-bed model. The co-existence and interference free communication of PU's and SU's are discussed based on performance metric for different scenarios. Also, the existing rendezvous protocol are used for efficient utilization of spectrum and secondary traffic coordination.

1.3 Background Information for Test-Bed Model

This section discuss about the Hardware and Software platform of our test-bed model. Also, we explain the capabilities, advantages and disadvantages of our model.

1.3.1 Hardware: USRP (Universal Software Radio Peripheral)

The Universal Software Radio Peripheral or USRP was developed by Ettus Research LLC that provides the low cost radio systems for commercial and research applications [1]. USRP provides digital baseband and IF section within the hardware, which aids to utilize general purpose computers to function as high bandwidth software radios. In addition, the board can provide interface to various daughter-boards for a wide range of applications. The basic philosophy behind the USRP is to provide all waveform specific features like modulation and demodulation in CPU, whereas the high speed operations like interpolation, decimation, digital up and down conversion are provided within FPGA [4][7].

1.3.1.1 Capabilities

The basic architecture for USRP version 1 contains USB 2.0, ADC's and DAC's, re-programmable FPGA and daughter-board interface. The Figure 1.2 provides the basic components of USRP version 1. Each USRP has 4 ADC and DAC, ADC outputs data at 12 bits per sample, 64 Million samples per sec (64 MS/sec) and DAC outputs data at 14 bits per sample, 128 Million samples per sec (128 MS/sec). The daughter-board interface allows to connect to ADC and DAC components. Furthermore, the USB 2.0 interface provides connection from FPGA to the computer. All samples sent over the USB interface are in 16-bit signed integers in IQ format [4][7].

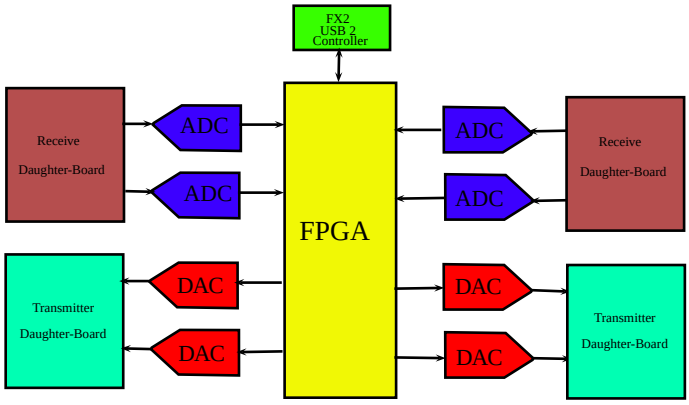

Figure 1.2: Hardware (USRP version 1) and Architecture [4]

1.3.1.2 Advantages and Disadvantages

1. It can operate in full-duplex mode. It allows the transmitter and receiver to operate independently.

2. The flexibility for utilizing wide-variety of daughter-boards can serve a wide range of commercial and research applications. For instance, RFX boards provide USRP to work as a RF transceiver system.

3. Maximum flexibility in frequency planning for RF boards.

4. The Maximum effective bandwidth is about 8 MHz. The limitation is due to the USB 2.0 interface where the combined data rate over the bus should not exceed 32 Megabytes per sec(32 MB/sec) [7]. This restricts the usage for high bandwidth signals.

5. Hardware latency restricts the effective implementation of MAC protocols.

1.3.2 Software: GNU Radio

GNU Radio is a free software development kit that provides signal processing modules to build a software radio in a real-time environment using low cost and reconfigurable radios [1]. It is based on block architecture that involves hybrid Python/C++ programming. Moreover, it provides us to integrate with analysis and plotting tools, such as octave and gnu-plot to support evaluation and computation experiments with ease.

1.3.2.1 GNU Radio Architecture

The baseline architecture of GNU Radio involves a complex flow-graph that consists of modules and low-level algorithms. Each module or low-level algorithm is structured in C++ and provides basic signal processing functions (ex: Filters, FFT, Channel Coding etc.,). They are automatically generated into python modules with the use of python 'wrapper' or interface i.e., SWIG (Simplified Wrapper and Interface Generator) [1]. The generated blocks are used to construct a flow-graph model with the help of python.
The application is built on python program that provides python framework. The python framework is responsible for communication of data through module buffers and creates a simple scheduler that helps to run blocks in a sequential order for single iteration. The GNU Radio software typically consists of four elements [2].

- **Source**: Each flow-graph has a single source. It is the head (start) of the flow-graph. For instance, USRP source or file source are common types of source blocks.
- **Sink**: Each flow-graph has a single sink. It is the tail (end) of the flow-graph. For instance, USRP sink or file sink are common types of sink blocks.
- **Flow-graph**: The application is based on a flow-graph. Each flow-graph consists of intermediate blocks along with single source and sink blocks. We can have multiple flow-graphs within a single application.
- **Scheduler**: It is created for each active flow-graph, which is based on steady stream of data flow between the blocks. It is responsible for transferring data through the flow-graph. It monitors each block for sufficient data at I/p and O/p buffers so as to trigger processing function for those blocks.

1.3.2.2 Advantages and Disadvantages

1. It has extensive library of pre-defined and test-bed functional blocks [2].
2. It provides simulation environment to build our own models.

3. The python wrapper or interface helps to provide easy access of blocks. For instance, SWIG in GNU Radio provides easy access of C++ blocks into python [2].

4. It requires experience in software; especially low-level programming.

1.4 Organization of Thesis

The thesis is organized as follows. Chapter 2 gives the spectrum sensing of the PU's based on energy detection method while evaluating the average power of primary traffic under two hypotheses. In Chapter 3, we develop the Markov traffic model consisting of ON and OFF cycles that are used to represent the PU traffic. The Chapter 4 gives the implementation of Coded OFDM model based on OFDM and Trellis convolution blocks in GNU Radio while mentioning the different approaches and their problems. The Experimental test-bed for CR is analyzed under different scenario's using performance metric is provided in Chapter 5. The Chapter 6 provides the Conclusion and Future work.

Chapter 2

Spectrum Sensing of Primary User

2.1 Motivation

Spectrum sensing is considered as a primary component of a CR system. It is required for the proper allocation of secondary user traffic on primary channel based on sensing results. Many researchers have studied the mathematical analysis of spectrum sensing algorithms based on environmental design and user constraints for CR [23][11]. In fact, [3] discusses the practical implementation of spectrum sensing based on square of the FFT values. There is a need for efficient energy detector to analyze wide-band spectrum. For instance, WLAN 802.11b physical layer utilizes direct-sequence spread spectrum(DSSS), which is a wide-band signal that occupies the entire 22 MHz spectrum of the 802.11 channel. To detect wide-band signals using energy detector, we need a wide-band spectrum analyzer that can scan the entire spectrum.

2.2 Objective

- In GNU Radio, *usrp_spectrum_sense.py* program helps to scan wide-band signals. The program doesn't provide an appropriate evaluation of average power. We analyzed the periodogram method in [21] to perform energy detection using average periodogram technique for the former program.
- The power spectral density (PSD), curve-fitting functions and histogram of average channel power are evaluated for PU traffic under two hypothesis (H_0 and H_1). For the H_1, the histogram and density plots of average channel power are demonstrated for two different cases.

2.3 Implementation

This section provides the mathematical formula for average periodogram technique [21][13], which can be used as a energy detection method for determining the presence or absence of primary traffic. Also, we discuss on implementation for wide band spectrum analyzer based on the average periodogram analysis.

2.3.1 Average Periodogram Analysis

It is used for estimation of power spectrum, which is based on discrete Fourier transform (DFT) of finite length segments of signals. It helps for computation of power spectral density (PSD). It involves sectioning the data into finite segments to compute individual periodogram or modified periodogram and averaging the modified periodogram segments. The main advantage is non-negative estimate and involves less computation than other methods.
Let $x[n], n = 0, 1, \ldots, L-1$ be the discrete time signal that are divided into K finite length equal segments, where length of each segment is N i.e., $KN = L$; $x_r[n],\ n = 0, 1, \ldots, N-1$ is the r^{th} segment and $w[n], n = 0, 1, \ldots, N-1$ be the window applied to each segment.The mathematical analysis follows that in [13]
The modified periodogram for the r^{th} segment is

$$I_r[k] = \frac{1}{NU}|V_r[k]|^2; \quad \text{for } k = 0, 1 \ldots N-1 \tag{2.1}$$

Where $V_r[k]$ is a N point DFT and U is normalization factor i.e.,

$$V_r[k] = DFT\{w[n]x_r[n]\} \quad \text{and} \quad U = \frac{1}{N}\sum_{n=0}^{N-1}(W[n])^2$$

The time averaged periodogram estimate $\overline{I}[k]$ is the average of modified periodograms

$$\overline{I}[k] = \frac{1}{K}\sum_{r=0}^{K-1} I_r[k] \tag{2.2}$$

The equation 2.2 indicates the PSD of $x[n]$ sequence.

2.3.2 Wide-band Spectrum Analyzer for USRP

Hardware limitations: In USRP1 (version 1) all the samples sent over the USB interface are in 16 bit signed integer format, i.e., 16 bit I and 16 bit Q data (complex) values, which

means 4 bytes per complex sample. USRP1 is built on USB, which can sustain 32 MB/sec over the bus. This results in (32 MB/sec/4 Byte) 8 Mega complex samples/sec across the USB. Since complex processing is used, this provides a maximum effective total spectral bandwidth of about 8 MHz (Nyquist criteria). In fact, USB bus limits the maximum bandwidth to 8 MHz in USRP1 [4].

Piece-wise average periodogram analysis: In GNU Radio, *usrp_spectrum_sense.py* program [1] provides a wide-band spectrum analysis. Here, the RF front end is tuned to suitable steps to examine wide-band spectrum, but not all at the same time. In fact, the former program doesn't perform periodogram analysis on FFT samples. We modified the program to perform the periodogram analysis based on mathematical analysis (2.2) and verified the practical calculation of power in time and frequency domain. The Figure 2.2 depicts the block diagram for piece-wise periodogram analysis. Tune delay(t_n) and Dwell delay(t_d) are two important parameters in piece-wise analysis [1].

- *Tune delay (t_d)*: It is the time period over which FFT samples are discarded for the RF front end to settle to new center frequency.
- *Dwell delay (t_n)*: It is the time period over which the average values (average power in each FFT bin) of vectors are determined for each center frequency. This operation is performed after discarding tune delay samples.

The piece-wise spectrum analysis can also be represented on a time and frequency plot. Assume the primary carrier frequency f_c lies between f_L and f_H i.e., $f_L \leq f_c \leq f_H$, where $f_L$andf_H are the lowest and highest frequency components of the primary spectrum band and $f_H - f_L > WMHz$. As USRP cannot scan more than 8 MHz, we scan in steps of W MHz ($W < 8MHz$)[1] over the entire spectrum band. In addition, frequency overlap between spectrum bands are considered to prevent frequency holes at spectrum edges [1]. The Figure 2.1 explains the 2-D illustration of piece-wise spectrum analysis for 25% overlap.

2.3.3 Methodology

The analysis is performed for two hypotheses(H_0 and H_1)

- H_0: Primary Traffic is OFF
- H_1: Primary traffic is ON

[1]To prevent the loss of samples at low decimation rates i.e., high data rate. The loss of sample or USRP overrun is indicated as UOUOUO..... on the print screen [1].

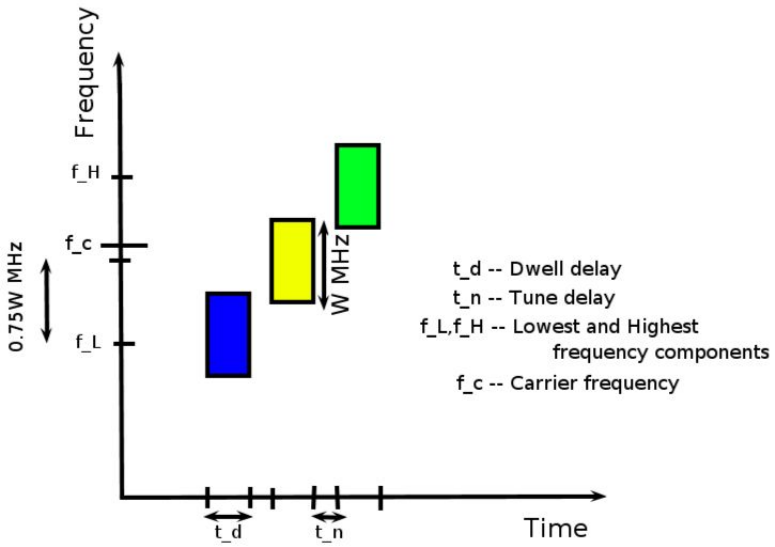

Figure 2.1: 2-D Illustration of Piece-wise Periodogram Analysis

For each hypothesis, we evaluate the PSD, curve-fitting functions, histogram and density plot of average channel power. In addition, the histogram and density plots of average channel power are analyzed for different time windows and windowing functions.
Primary Users(PU's): We consider, two USRPs communicate on ISM (Industrial, Scientific and Medical) band. The parameters of primary traffic are

- Modulation : DBPSK
- Data Rate : 500 Kb/sec
- Carrier Frequency : 2.422GHz
- Primary Bandwidth (ΔW): 675 kHz

Secondary User(SU): We consider, USRP node to sense 3 MHz (2420.5-2423.5MHz) of the primary channel around the primary carrier frequency. In USRP, the ADC clock runs at 64 MS/sec. With decimation rate as 64 [2], it can analyze 1 MHz of spectrum and the scan is performed in frequency steps to cover the entire spectrum. Also, a 25% overlap is considered to avoid the frequency holes at the spectral edges and due to which it scan 2.5 MHz of spectrum covering from (2420.5 - 2423 MHz).
First step: A command is sent to RF front end to tune to 2421 MHz (f_0). It will collect the statistics from frequencies 2420.5 MHz to 2421 .5 MHz
Second step: A command is sent to RF front end to tune to 2421.750 MHz (f_1). It will

[2]To avoid the loss of USRP samples, where the UoUoUo.... appears with the decrease of decimation

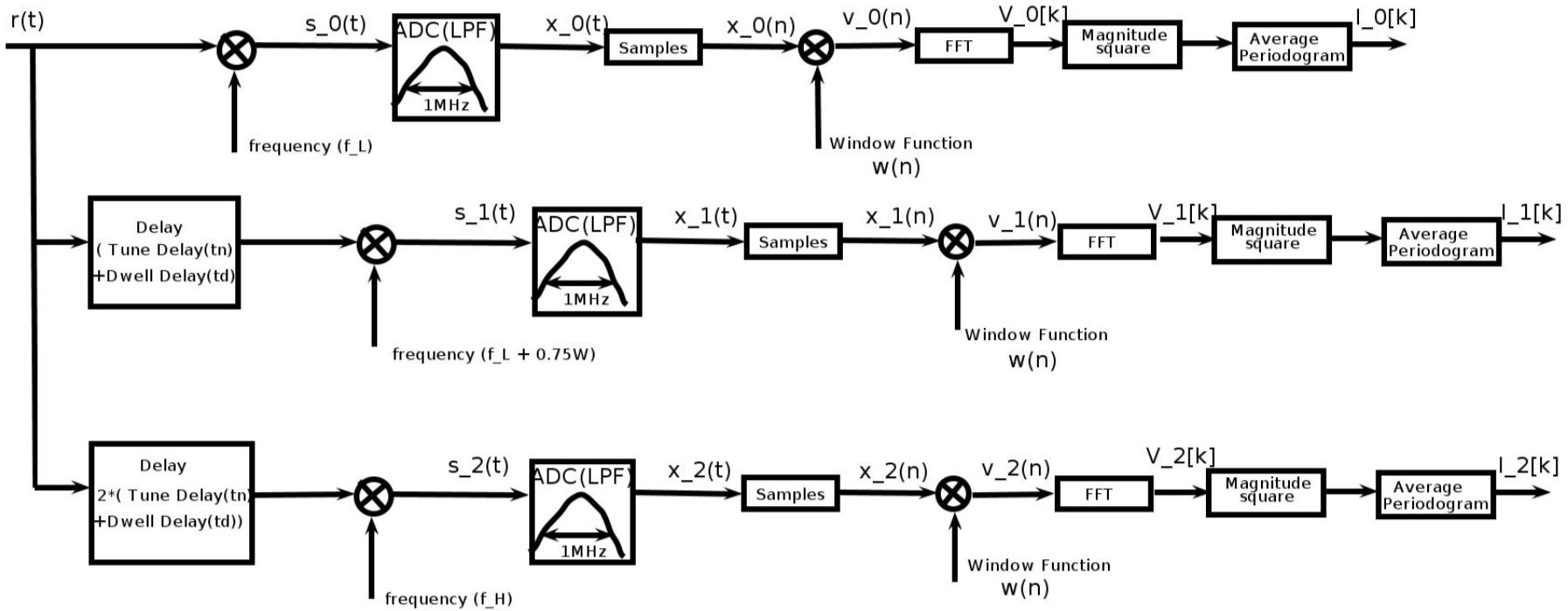

Figure 2.2: Block Diagram for Piece-Wise Periodogram Analysis

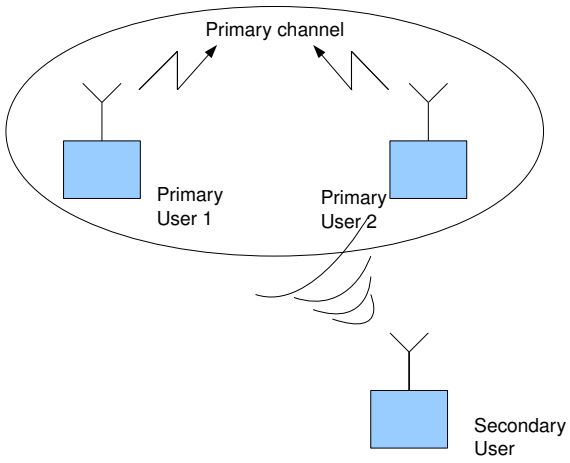

Figure 2.3: Block Diagram for sensing model

collect the statistics from frequencies 2421.125 MHz to 2422.225 MHz
Third step: A command is sent to RF front end to tune to 2422.500 MHz (f_2). It will collect the statistics from frequencies 2422 MHz to 2423 MHz
For each center frequency, the modified periodograms(FFT bins) collected for the dwell delay (t_d) are used to obtain the average periodogram based on (2.2). Also, the overlapping frequency bins for the three center frequencies are averaged. The calculated values are plotted to obtain the power spectral density(PSD).

2.4 Analysis

This section provides the analysis for the section 2.3 with plots for PSD and curve-fitting functions. Also, the variations in average channel power are studied by providing the histogram and density plots with different cases.

2.4.1 PSD and Curve-fitting Functions

The Figure 2.4 and 2.5 provides the PSD and curve-fitting functions around carrier frequency for the two hypothesis.

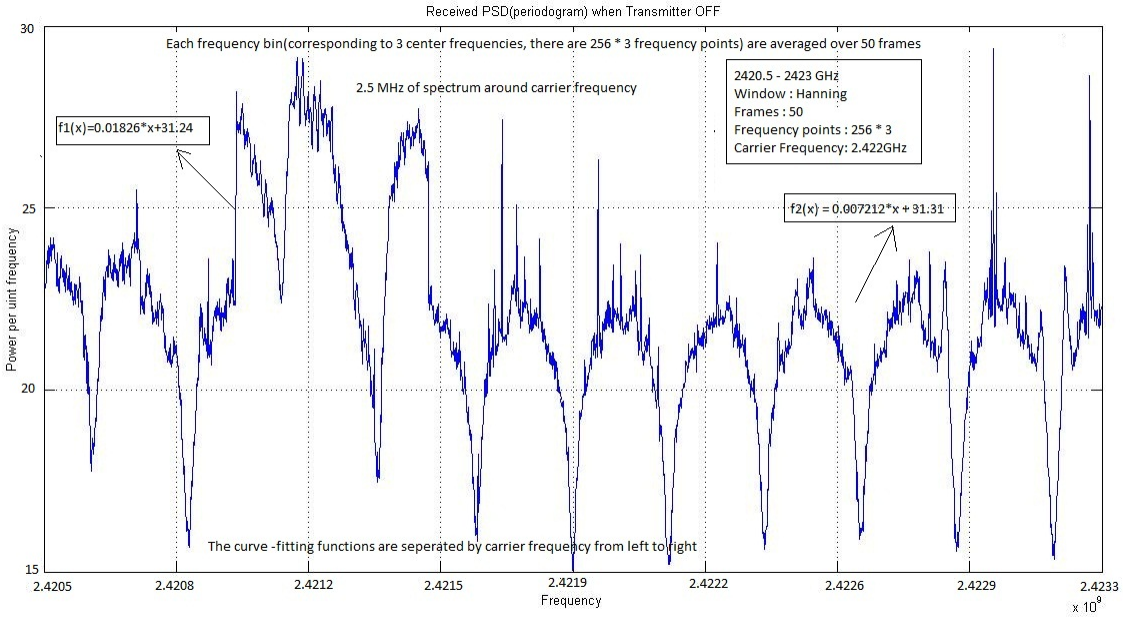

Figure 2.4: PSD and Curve-fitting plot under H_0

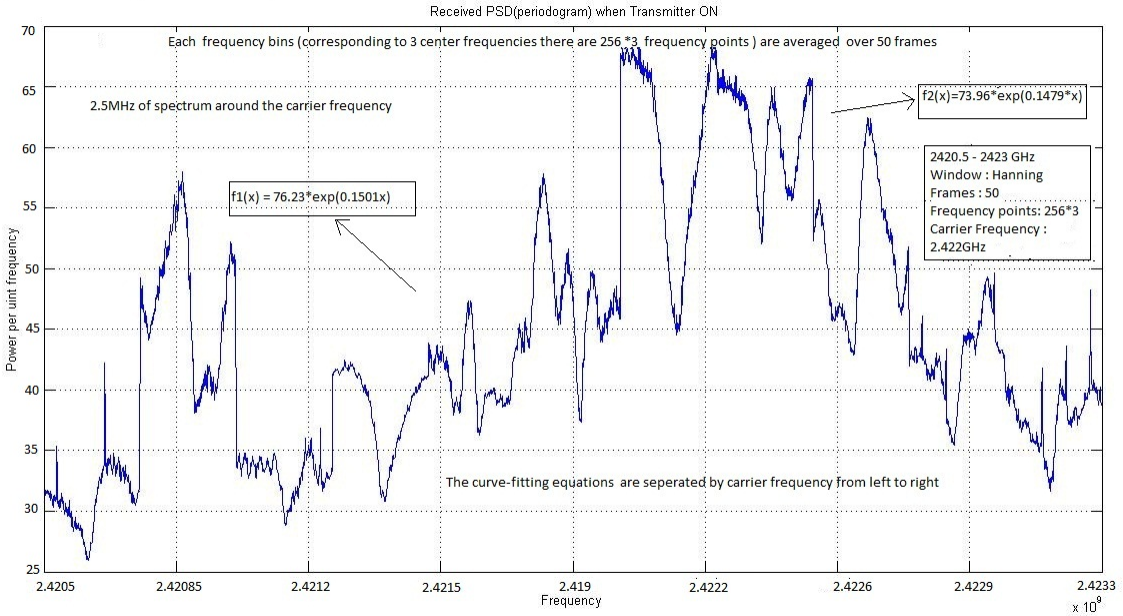

Figure 2.5: PSD and Curve-fitting plot under H_1

2.4.2 Histogram and Density Plots

The number of center frequencies is inversely related to USRP decimation rate. For instance, if the frequency range for scanning is δW and decimation rate is d then number of center frequencies(L) is given by $L = \frac{\delta W * d}{64MS/sec}$[3].
The average power for the spectrum band is the average of each average periodogram (obtained for each center frequency)(2.2) over the entire spectrum band. The Histogram and density plots are obtained for the average power collected over 300 frames. Histogram and density plots of average power under the two different hypotheses is shown in the Figure 2.6. The mean value in the Figure 2.6 indicates the average channel power when primary transmitter is ON (H_0) and OFF(H_1). In fact, the mean value of H_0 is the average noise power of the channel.

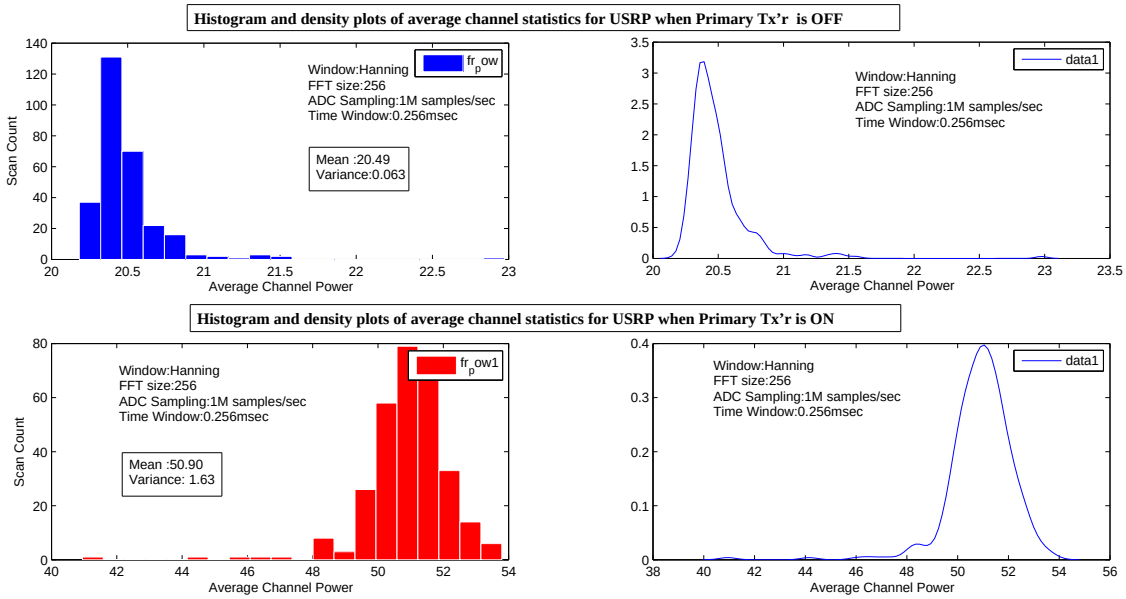

Figure 2.6: Histogram and density plots of average channel power for H_0 and H_1

2.4.3 Cases

The histogram and density plots of average channel power under H_1 are evaluated for different time windows and windowing functions.

[3]ADC Sampling rate is 64 MS/sec

2.4.3.1 Time Window

By changing the decimation rate and FFT length, the time window of measurement is varied. In USRP ADC clock runs at 64 MS/sec.
For example, the decimation rate is 8 and FFT length =256 and the samples at ADC are decimated by factor of 8 and sampled at 8 MS/sec i.e., Time window for 256 samples is 0.256 msec.
The Histogram and density plots are plotted for varying decimation rates(64,128,256) and with constant FFT length(256) is provided in the Figure 2.7.
There is decrease in the value of average channel power with the increase of time window

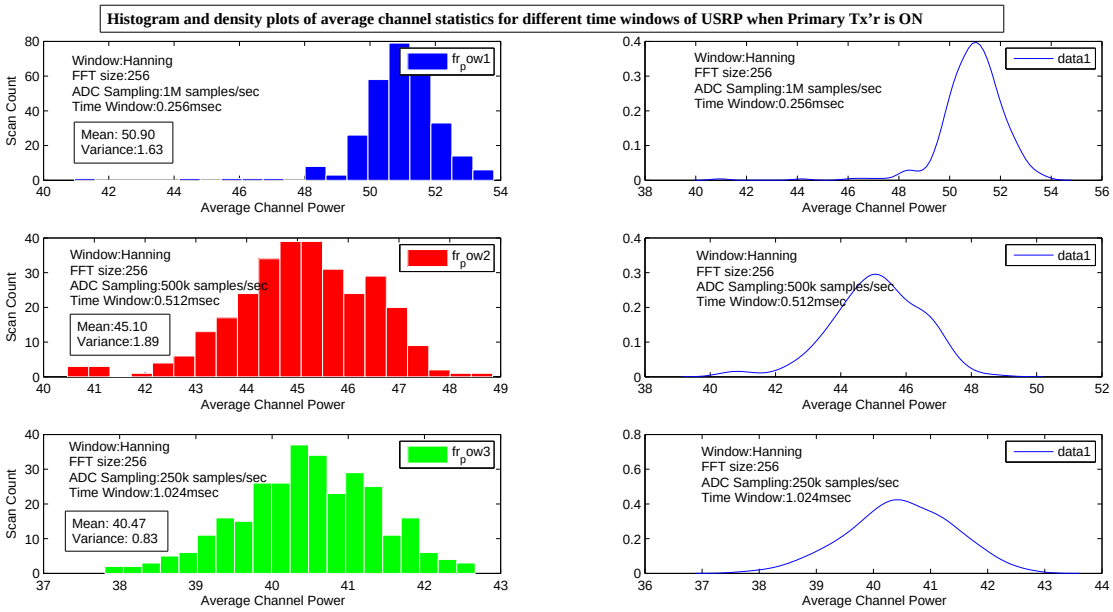

Figure 2.7: Histogram and density plots of average channel power for different time windows under H_1

where the increase of time window is achieved by decreasing the ADC sampling rate.

2.4.3.2 Windowing Function

Windowing is performed to reduce smearing, spectral leakage and to preserve the amplitude of signal. The different window function doesn't affect the shape of the power spectrum. But, there is a change in characteristics of the wave such as reduction in spectral leakage and side lobe power. Hanning window gives the best performance due to lower side lobes

[13].
The Histogram and density plots for Hanning, Blackmanharris and Rectangular window functions is shown in the Figure 2.8.
The average channel power for different windowing functions remain constant.

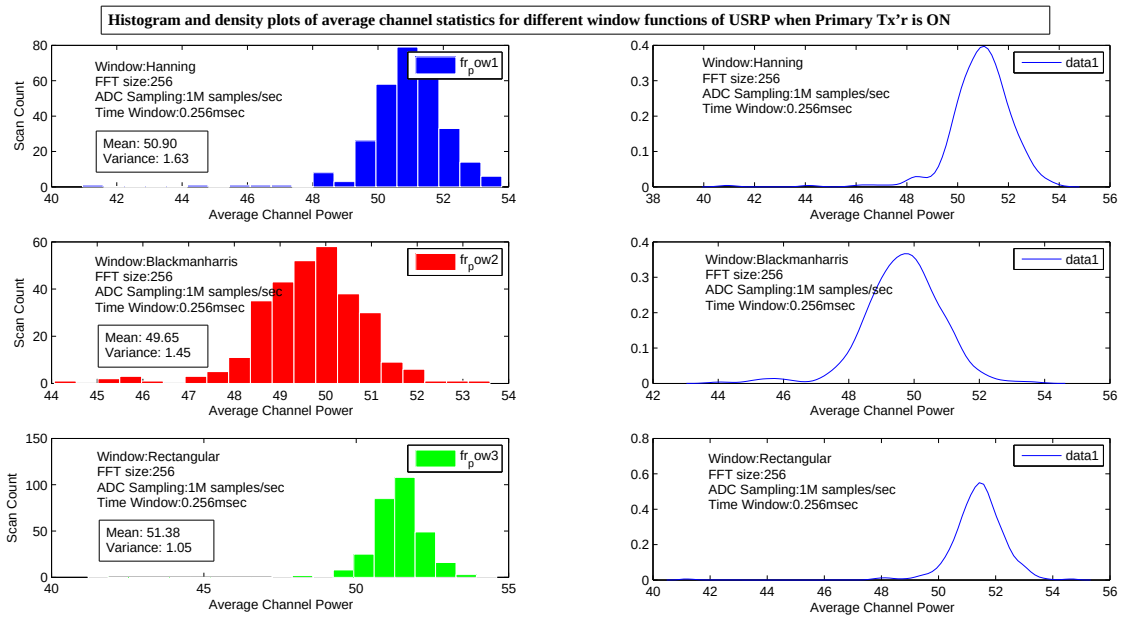

Figure 2.8: Histogram and density plots of average channel power for different windowing functions under H_1

2.5 Remarks

1. The bin_statistics functional block performs the maximum over the dwell time for the center frequency bins. We have implemented the average over the dwell time so as to evaluate the average periodogram analysis for each center frequency.

2. The shape of the PSD determines the probability density function (pdf) of the energy statistics under each hypothesis.

3. In order to implement periodic sensing involving sleep and sensing periods, we provide sleep_mode in the functional block of bin_statistics. It helps to discard the samples collected during the sleep mode. It is more robust and provides efficient sensing results compared to the sleep function in python.

Chapter 3

Markov Traffic Model

3.1 Motivation

In a Cognitive Radio Network(CRN), the Markov model for the primary user(PU) spectrum usage helps to study the dynamic behavior of spectrum access and traffic patterns of the PU. Also, primary channel is modeled in the form of alternative ON and OFF cycles with secondary users (SUs) utilizing the OFF periods for transmission. For instance, a simulation model of semi-Markov traffic is considered for PUs to enhance the efficient utilization of the OFF periods by SU in [10].

3.2 Objective

- To generate Markov traffic model involving ON and OFF periods on a cognitive test-bed. Also, to configure ON periods for transmission of long and short bursts of traffic, while the OFF periods vary depending on the former period traffic burst.
- Analyze the output spectrum for DBPSK modulation with spectrum analyzer and MATLAB plot for the PSD equation provided in [14].

3.3 Implementation

This section discuss the Markov process and the implementation of Algorithm on GNU Radio. Moreover, the PSD equation is analyzed theoretical and practical verification is provided on a spectrum analyzer.

3.3.1 Markov Process

Markov process or Markov chain is paradigm of states in which the current state depends on the past states. Homogeneous Markov chain [6] has finite number of states that are independent of time, and their transition matrix P indicates the probability of transitions from one state to another. The Figure 3.1demonstrates the two state Markov model.
S_o and S_1 are two states of Markov model with

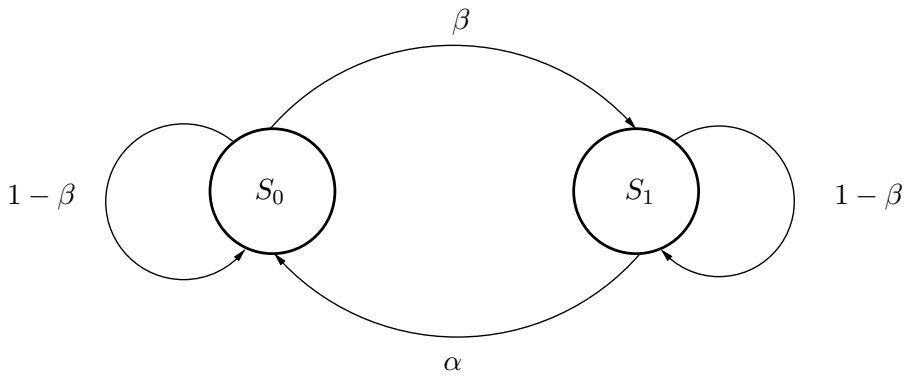

Figure 3.1: Block Diagram for two state Markov model

Probability of transition from stateS_1 to S_0 *i.e.*, $Pr[S_0/S_1] = S_{01} = \alpha$
Probability of transition from stateS_0 to S_1 *i.e.*, $Pr[S_1/S_0] = S_{10} = \beta$
Similarly,
Probability of transition from stateS_0 to S_0 andS_1 to S_1*i.e.*,
$Pr[S_0/S_0] = S_{00} = 1 - \alpha$ and $Pr[S_1/S_1] = S_{11} = 1 - \beta$
Transition matrix P for two state Markov model is given as

$$P = \begin{bmatrix} S_{00} & S_{01} \\ S_{10} & S_{11} \end{bmatrix} = \begin{bmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{bmatrix}$$

3.3.2 Explanation

The two state Markov model uses *benchmark_tx.py*(GNU Radio example[1]), which provides packet based modulation for continuous and discrete transmission. In the program, the two states OFF (S_0) and ON (S_1) indicate the presence and absence of primary traffic on the channel. The S_0 traffic is a period of no transmission that is produced on halting the main thread and S_1 traffic is similar to continuous traffic. The states of Markov model are generated from the predefined transition matrix P and based on the current state, the burst traffic of ON and OFF periods are produced. Also, the program provides the

flexibility for allocation of number of states and available packets for ON cycles.
In S_0 state, the time.sleep(txoff) function is utilized for halting the main thread of Python. The txoff represents the duration of OFF period, which depends on the former ON state i.e.,the length of previous ON burst determines the duration for the OFF period. The OFF periods are maintained longer than ON periods so as to provide sufficient transition time for switching between ON and OFF periods and to avoid the ON period creep into OFF period. Also, the longer ON periods requires longer OFF periods. For the run-time of the program, the indication of UuUu....[1] represents the S_0 state.
In S_1 state, the ON period is modeled with probability density function(p.d.f) $f_{T_{ON}}(x), x > 0$ that follow uniform distribution. The number of packets for ON period depends on available packets and their prior probabilities. For instance, the available packets for ON periods are x,y,z then the $\Pr(\mathrm{x}), \Pr(\mathrm{y}), \Pr(\mathrm{z})$ determines the packets for the current ON state. The transmission period depends on data rate, number of packets and packet size. Also, the random selection of packets contributes for short and long bursts of traffic. For the run-time of the program, the indication of dots represents the number of packets transmitted. The algorithm in Appendix A provides the Markov traffic for the current model.

3.3.3 PSD of DBPSK Modulation

PSD for DBPSK modulated wave is given by [14]. It is based on determining the PSD for a differentially encoded sequence. DBPSK can be considered as a combination of differential encoder and BPSK modulation. The basic structure of DBPSK modulation is provided in figure 3.2.

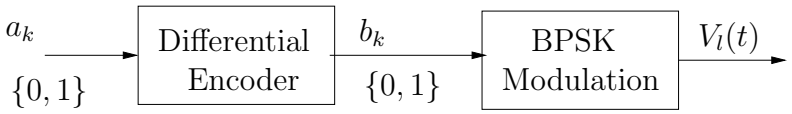

Figure 3.2: Block Diagram for DBPSK modulation

Let a_k be the i.i.d binary random variables with probabilities; $Pr[a_k = 1] = p$ and $Pr[a_k = 0] = 1 - p$; $0 < p < 1$. The sequence b_k is generated as $b_k = a_k \oplus b_{k-1}$. The PSD of differentially encoded sequence(b_k) and DBPSK modulated signal (V_l) is evaluated in [14]

$$S_b(f) = \frac{2p(1-p)}{1-(1-2p)\cos 2\pi fT - 2p(1-p)} + \frac{\mu_b^2}{T}\sum_{k=-\infty}^{\infty}\delta(f - K/T) \tag{3.1}$$

$$S_{v_l}(f) = \frac{1}{T}|G(f)|^2 \cdot S_b(f) \tag{3.2}$$

[1]USRP underrun [1] indicates that there are not enough samples for USRP transmission

where,
S_b is the PSD of differential encoded sequence.
μ_b is the mean of the b_k sequence.
$G(f)$ is the frequency response of the pulse shape.
T is the signaling interval.
If input symbols are equiprobable i.e,$Pr[a_k = 1] = 1/2$ and $Pr[a_k = 0] = 1/2$ and mean $\mu_b = 0$ then from equation (3.1)

$$S_b(f) = 1 \tag{3.3}$$

If, $g(t)$ is root raised cosine filter, frequency response $G(f)$ is provided as [6]

$$G(f) = \begin{cases} \sqrt{T} & \text{for} \quad 0 \leq |f| \leq \frac{1-\beta}{2T} \\ \sqrt{\frac{T}{2}[1 + \cos(\frac{\pi T}{\beta}(|f| - \frac{1-\beta}{2T}))]} & \text{for} \quad \frac{1-\beta}{2T} \leq |f| \leq \frac{1+\beta}{2T} \\ 0 & \text{for} \quad |f| \geq \frac{1+\beta}{2T} \end{cases} \tag{3.4}$$

From equation (3.3) and (3.4), if the input sequence are equiprobable, the mean of differential encoded sequence is zero and pulse shaping is raised cosine waveform, then PSD of DBPSK is given by (3.2)

$$S_{v_l}(f) = \begin{cases} 1 & \text{for} \quad 0 \leq |f| \leq \frac{1-\beta}{2T} \\ \frac{1}{2}[1 + \cos(\frac{\pi T}{\beta}(|f| - \frac{1-\beta}{2T}))] & \text{for} \quad \frac{1-\beta}{2T} \leq |f| \leq \frac{1+\beta}{2T} \\ 0 & \text{for} \quad |f| \geq \frac{1+\beta}{2T} \end{cases} \tag{3.5}$$

3.3.3.1 Plots

The basic parameters of DBPSK modulated wave

- Roll of factor for root-raised cosine filter (β) : 0.35
- Interpolation : 128
- Carrier Frequency (f_c) : 2.422 GHz
- Samples per symbol : 2

In USRP, the relation between Data rate,interpolation and samples per symbol is given as [2]

$$Datarate(R_b) = \frac{DACsamplingrate}{Interpolation * Samplespersymbol}$$

For DBPSK, the symbol rate(R_s) is equal to the data rate (R_b) and the symbol period $T = 1/R_s$

[2]DAC sampling rate : 128 MS/sec

Substituting the value of β and symbol period T in equation (3.5) provides PSD for DBPSK modulation, the MATLAB plot and the output of spectrum analyzer for DBPSK modulation are shown in the Figures 3.3 & 3.4

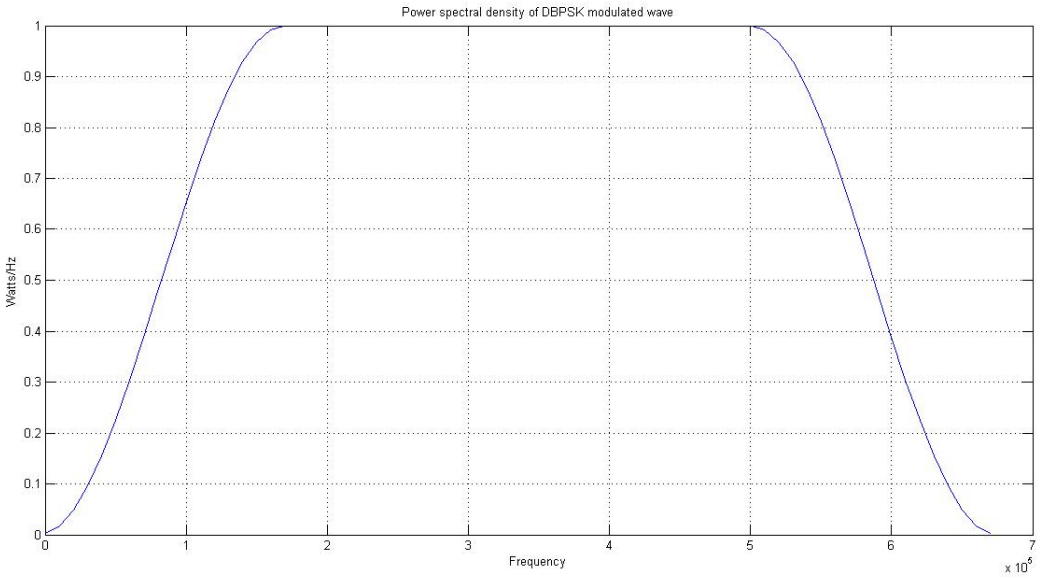

Figure 3.3: MATLAB plot for PSD of DBPSK modulated waveform using equation (3.5)

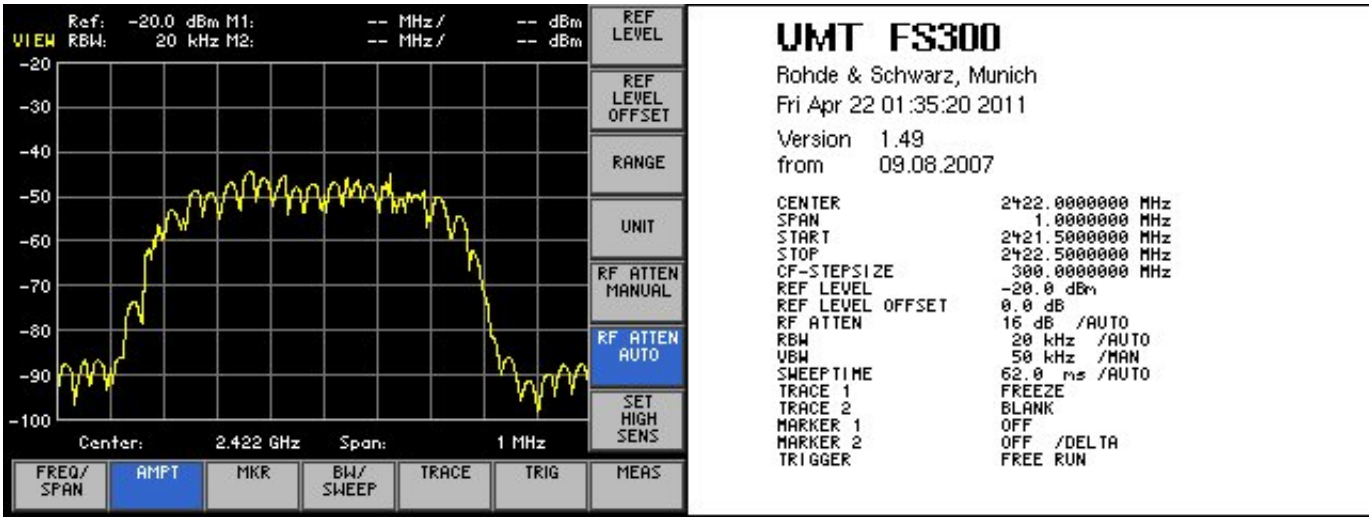

Figure 3.4: PSD of DBPSK modulated waveform on Spectrum Analyzer

3.4 Problems

Transition time is the time required to switch from one state to another. It is required to prevent the overlap between S_0 and S_1 periods. In fact, the longer S_0 periods provides the sufficient transition time to switch between S_0 and S_1 periods or vice-versa. Also, the transition time from continuous S_1 states to S_0 state is greater than the transition time from S_0 to S_1 or continuous S_0 states to S_1 state. For instance, the Markov chain sequence $S_0(1)$, $S_1(1)$, $S_0(2)$, $S_0(3)$, $S_1(2)$, $S_1(3)$, $S_1(4)$, $S_0(4)$, where $S_0(i); i = 1, 2, 3, 4$ represents OFF state and $S_1(i); i = 1, 2, 3, 4$ represents the ON state. The transition time between $S_0(1)$ and $S_1(1)$ or $S_0(3)$ and $S_1(2)$ is less than the transition time between $S_1(4)$ and $S_0(4)$. In fact, the $S_0(4)$ has longer OFF period than other OFF periods.

Chapter 4

Coded OFDM Transceiver

4.1 Motivation

Many Forward Error Correction(FEC) blocks are used with OFDM in fields such as WLAN /WMAN and DAB/DVB systems. The use of convolution codes in OFDM system helps to reduce the peak average ratio power(PARP) while improving the bit error rate (BER) [17]. In [22], a convolution code is implemented to the current OFDM model on GNU Radio that uses low code rate. However, the current Trellis Convolution Blocks on GNU Radio provide the flexibility for high code rates in addition to the Viterbi Algorithm with soft and hard decision decoding.

4.2 Objective

- Mathematical model for the Coded OFDM model
- Create Coded OFDM model using Trellis blocks in GNU Radio
- Packet structure and flow for the coded OFDM model
- Cope with big burst of errors in the coded OFDM model

4.3 Implementation

This section provides the background information of OFDM model, synchronization and equalization on GNU Radio. The channel coding technique and available blocks are also

discussed. Moreover, a mathematical model for the coded OFDM is explained for the implementation.

4.3.1 Background Information

4.3.1.1 OFDM

OFDM or Orthogonal Frequency Division Multiplexing is a type of multi-carrier(MC) technique that allows for high data rate transmission and reduces the effect of Inter Symbol Interference (ISI) caused by delay spread in wireless channels [17]. It divides the available channel bandwidth (W) into sub-bands(N) of narrow-width($\delta f = W/N$). Also, all the sub-bands are executed in parallel that are independently coded and modulated for high data rates.
In GNU Radio, the flow-graphs for OFDM transmitter and receiver are shown in the Figure4.1.

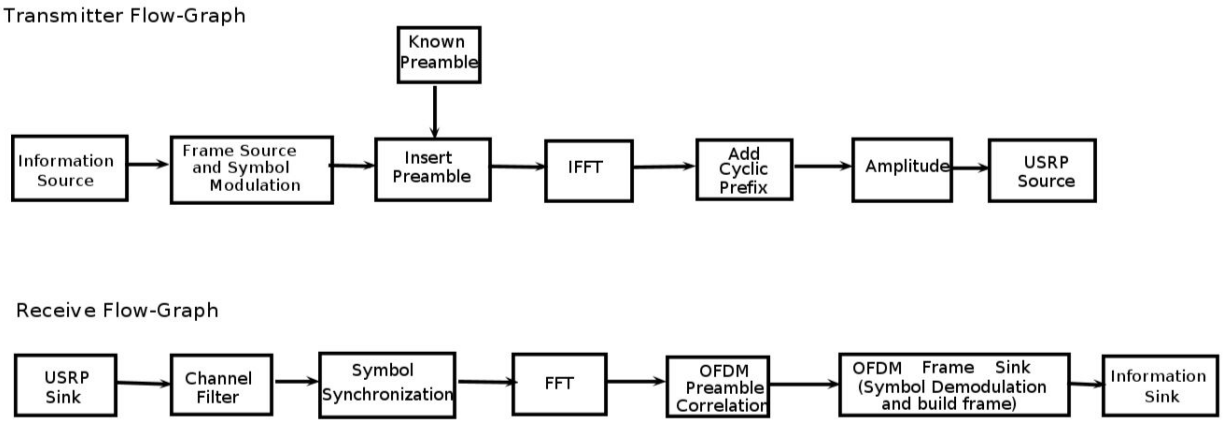

Figure 4.1: Flow-graph for OFDM model [12]

4.3.1.2 Synchronization and Equalization

Synchronization of an OFDM signal involves finding the symbol timing and the carrier frequency offset. They need to be efficiently determined for proper recovery of OFDM symbols. In the OFDM model of Figure 4.1, signal detection at the receiver is performed using Maximum likelihood [20] or PN -sequence correlation [16]. PN- sequence correlation involves transmitting known preamble symbol along with the OFDM symbols for synchronization. In fact, the preamble symbol contain one training sequence with two known

symbols of equal length in the time domain.
At the transmitter, the desired training sequence is obtained from the frequency domain sequence consisting of information on even frequencies and zeros on odd frequencies which are passed through an IFFT block [12]. The generation of training sequence is demonstrated in Figure 4.2. The "known preamble", "insert preamble" and "IFFT blocks" in Figure 4.1 are responsible for generation and encapsulation of the desired training sequence. It is appended before the start of OFDM symbols that are generated from each packet.

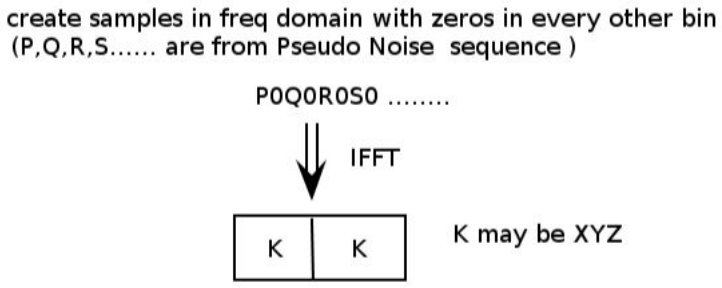

Figure 4.2: Training Sequence for Symbol Synchronization [12]

At the receiver, the generated training sequence that is placed at the start of the packet(frame) is used for symbol synchronization. The symbol timing is achieved by searching for the symbol in which the first half is similar to second half in time domain [12]. On the other hand, equalization is achieved by generating 1-tap equalizer from the received and known symbols (training sequence) that correct phase shifts and multi-path effects. The synchronized and equalized OFDM symbols are passed through OFDM demodulation. In Figure 4.1, the symbol synchronization and OFDM preamble correlation blocks are used for synchronization and equalization of OFDM symbols.

4.3.1.3 Channel Coding

Convolution coding: Channel coding contains channel encoding and decoding blocks that are used to improve the reliability of transmission by detecting and correcting errors that are introduced in the channel. Convolution coding is a type of channel coding technique that has memory and described in terms of finite state machine(FSM). In these codes, "each time instant k, x_k information is encoded to y_k information sequence and changing the state of encoded from s_k to s_{k-1} " [6]. The primary advantage of convolution codes is the natural Trellis structure that helps decoding based on Viterbi algorithm.

Trellis based convolution blocks in GNU Radio: Trellis convolution blocks are based on finite state machine(FSM). A FSM has a finite number of states,input and output symbols. It completely hides the information about input and output symbols. In GNU Radio,

the Trellis encoder, Metric calculator and Viterbi decoder are the three blocks that utilize FSM structure for convolution coding. The mathematical expressions for FSM and convolution blocks are explained in the section below.

1. Trellis Encoder : It is useful for generation of encoder sequence based on the initial state of FSM.
2. Metric Calculator: It provides proper metrics to the Viterbi algorithm. The metrics can be soft or hard decision decoding, where the soft decision decoding is based on Euclidean distance and the hard decision decoding is based on Hamming distance.
3. Viterbi Decoder : It performs the Viterbi decoding for K Trellis steps.

4.3.2 Mathematical Model for Coded OFDM Blocks

Let,

- U^p and V^q be the uncoded and coded bit stream of the transmitter
- $\underline{X}$ and $\underline{Y}$ are the input and output OFDM symbol.
- N and $\tilde{N}$ are the FFT length and sub-carriers of the OFDM symbol.
- K_s is the known preamble symbol
- $\hat{U^p}$ and $\hat{V^q}$ are the uncoded and coded bit stream of the receiver.
- Rate of the channel encoder, e.g. code rate $(\frac{p}{q}) = \frac{1}{2}$

4.3.2.1 Transmitter

The information source has data in the form of packets that are transformed into bit stream (U^p). The bit stream is channel encoded to produce the coded bit stream(V^q).

$$U^p = (U_0, U_1, ..., U_{p-1}) \quad \text{and} \quad V^q = (V_0, V_1, ..., V_{q-1}); \quad \text{Also, } |U^p| \neq |V^q| \tag{4.1}$$

where, $U_i \in I_d$ for $i = 0, 1, \ldots, p-1$ and $V_i \in O_d$ for $i = 0, 1, \ldots, q-1$; I_d and O_d are defined from the FSM that are explained below.
In the OFDM modulator, the symbol modulator maps the channel encoded bit stream (V^q)

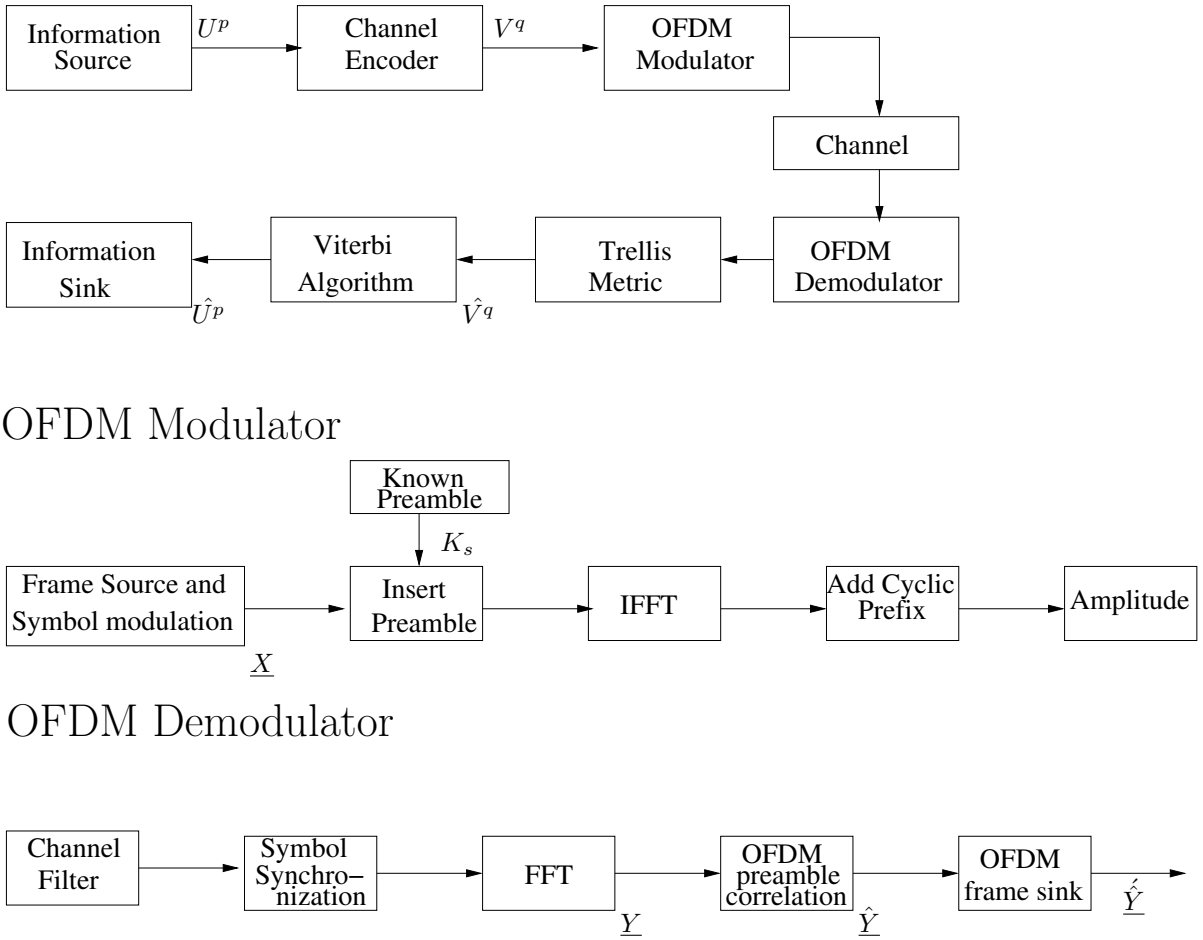

Figure 4.3: Block Diagram for coded OFDM

to sub-carriers based on the signal constellation. The output of the block is an OFDM symbol of length N which contain $\tilde{N}$ sub carriers and $(N - \tilde{N})$ zero sub-carriers.

$$\underline{X} = (l_0, l_1, \cdots, l_{z_l-1}, X_0, X_1, \cdots, X_{\tilde{N}-1}, l_{z_l} \cdots, l_{N-\tilde{N}-1}); \quad |\underline{X}| = N \tag{4.2}$$

for each sub-carrier, $X_i \in C = \{a_j \in R^D; 1 \leqslant j \leqslant M\}$
where, $l_i = 0$ for $i = 1, 2, \ldots, (N - \tilde{N})$, z_l is the number of zero sub-carriers to the left of $\tilde{N}$ sub-carriers, C is the constellation, a_j is the signal point in C, D is the number of dimension, M is the constellation size.
The known preamble (K_s) is appended to the OFDM symbols for synchronization and equalization. The number of OFDM symbols between the preambles depends on packet size($mbytes$),sub-carriers($\tilde{N}$) and number of bits per symbol(k).
The number of OFDM symbols generated for each packet $= \frac{m*8}{\tilde{N}*k} = 8\nu$, where $\nu = \frac{m}{\tilde{N}*k}$
In fact, the output of the insert preamble block corresponds to

$$K_s, \underline{X}^0, \underline{X}^1, \cdots\cdots\cdots, \underline{X}^{8\nu-1}, \; K_s, \underline{X}^{8\nu}, \underline{X}^{8\nu+1}, \cdots\cdots\cdots, \underline{X}^{16\nu-1} \cdots\cdots\cdots \tag{4.3}$$

where K_s = P0Q0R0 .....
The discrete OFDM symbols(time-domain) acquired from the IFFT block are appended with the cyclic prefix.

4.3.2.2 Receiver

The output OFDM symbols of the channel are passed through the channel filter, symbol synchronization and FFT blocks to acquire the synchronized OFDM symbols($\underline{Y}$) based on the PN sequence correlation [12]. They are corrupted by channel coefficients(fading channel parameters) and noise.

$$\underline{Y} = (m_0, m_1, \cdots, m_{z_l-1}, Y_0, Y_1, ...Y_{\tilde{N}-1}, m_{z_l}, \cdots, m_{N-\tilde{N}-1}); \quad |\underline{Y}| = N \tag{4.4}$$

where,
$Y_k = |h_k| * X_k + \eta_k$, for $k = 0, 1, \ldots, \tilde{N} - 1$
$m_k = |h_k| * l_k + \eta_k$, for $k = 0, 1, \ldots, N - \tilde{N}$
$|h_k|$ and η_k are channel and noise coefficients.

The OFDM preamble correlation block performs the correlation and equalization on the synchronized OFDM symbols($\underline{Y}$). It accepts the vector of complex constellation points from FFT that correlates with the known preamble to estimate the frequency offset in the FFT bins. It then performs the 1-tap equalization on all the sub-carriers to correct phase and amplitude distortion. In addition, the corrupted zero sub-carriers are removed from the OFDM symbol. The algorithm for the equalization and correlation is explained

in Appendix B. $\hat{\underline{Y}}$is the equalized OFDM symbol.

$$\hat{\underline{Y}} = (\hat{Y}_0, \hat{Y}_1, \cdots\cdots\cdots, \hat{Y}_{\tilde{N}-1}); \quad |\hat{\underline{Y}}| = \tilde{N} \tag{4.5}$$

where, $\hat{Y_k} = Y_k/\hat{h_k} = X_k + \hat{\eta_k}$, $\hat{h_k}$ are the estimates of channel coefficients and $\hat{\eta_k} = \eta_k/\hat{h_k}$. $\hat{\underline{Y}}$ are passed through OFDM demodulator to obtain the demodulated OFDM symbols($\hat{\underline{Y}}$)

$$\hat{\underline{Y}} = (\hat{Y}_0, \hat{Y}_1, \cdots\cdots\cdots, \hat{Y}_{\tilde{N}-1}); \quad |\hat{\underline{Y}}| = \tilde{N} \tag{4.6}$$

The algorithm for the demodulation using phase locked loop is explained in the Appendix C. The OFDM symbols are passed through the Trellis metric blocks to perform the soft or hard decision decoding so as to pass them to the Viterbi blocks.

FSM structure: All the Trellis convolution blocks are based on FSM. A FSM contains finite number of states, input and output values. A FSM class can be constructed by containing information of input(I), state(S), output(O), next state(NS), output symbol(OS). They are described as
Input denotation $I_d \in \{0, 1, 2 \cdots\cdots, I-1\}$,with cardinality I and x_k takes values from I_d
Output denotation $O_d \in \{0, 1, 2 \cdots\cdots, O-1\}$,with cardinality O and y_k takes values from O_d
State denotation $S_d \in \{0, 1, 2 \cdots\cdots, S-1\}$,with cardinality S and s_k takes values from S_d
Next state $NS : S_d X I_d - - > S_d$, meaning $s_{k+1} = NS(s_k, x_k)$
Output Symbol $OS : S_d X I_d - - > O_d$, meaning $y_k = OS(s_k, x_k)$
For instance, if the code rate is 1/2, then the input cardinality(I) is 2 and the output cardinality(O) is 4. for time instant k, input x_k, state s_k , output y_k and $y_k = OS(x_k, s_k)$. The state s_k moves to the next state s_{k+1}.

Soft Decision Decoding is based on the Euclidean Distance metric. The sub-carriers ($Y_k$for$k = 0, 1, \cdots \tilde{N}-1$) of synchronized OFDM symbol($\underline{Y}$) and their channel estimates($\hat{h}_k$) are used in the computation.
Euclidean Distance: For each sub-carrier of D dimension, O output symbols are produced. The metrics are obtained by

$$\|Y_k - \hat{h_k} * C_i\|^2 = \sum_{j=1}^{D} |Y_{k,j} - \hat{h_{k,j}} * C_{i,j}|^2 \tag{4.7}$$

where, $Y_k = (Y_{k,1}, Y_{k,2}....., Y_{k,D})$, $C_i = (C_{i,1}, C_{i,2}, ..., C_{i,D})$ is defined from the constellation, $\hat{h_k} = (\hat{h_{k,1}}, \hat{h_{k,2}}, ..., \hat{h_{k,D}})$ are the channel estimates for each sub-carrier and O depends on the output cardinality of the FSM channel coder.
For instance, if the code rate= 1/2 and QPSK modulation is utilized, then the channel encoder outputs 2 bits for each 1 bit and output cardinality of FSM is 4. So, for each

sub-carrier we obtain 4 metrics i.e., for Y_0, we get $||Y_0 - \hat{h_0} * C_0||^2, ||Y_0 - \hat{h_0} * C_1||^2, ||Y_0 - \hat{h_0} * C_2||^2, ||Y_0 - \hat{h_0} * C_3||^2$

Hard Decision Decoding is based on the Hamming Distance metric. The sub carriers ($\hat{Y_k}$ for $k = 0, 1, \cdots, \tilde{N} - 1$) of demodulated OFDM symbol($\underline{\hat{Y}}$) are used in the computation.
Hamming Distance: For each sub-carrier of D dimensional vector, O output metrics are produced.The metrics are obtained by

$$i_0 = \underset{i}{argmin} \|\hat{Y_k} - C_i\|^2 = \underset{i}{argmin} \sum_{j=1}^{D} |\hat{Y_{k,j}} - C_{i,j}|^2 \tag{4.8}$$

where $\hat{Y_k} = (\hat{Y_{k,1}}, \hat{Y_{k,2}}.....\hat{Y_{k,D}})$, $C_i = (C_{i,1}, C_{i,2}, ..., C_{i,D})$ are defined from the constellation and O depends on the output cardinality of the FSM channel encoder. For O output metrics, the position of i_0 is set to zero and the remaining metrics are taken as 1. For instance, if the code rate= 1/2 and QPSK modulation is utilized, then the channel encoder outputs 2 bits for each 1 bit and output cardinality of FSM is 4. So, for each sub-carrier we obtain 4 metrics i.e., for $\hat{Y_0}$, we get $||\hat{Y_0} - C_0||^2, ||\hat{Y_0} - C_1||^2, ||\hat{Y_0} - C_2||^2, ||\hat{Y_0} - C_3||^2$ and if the value of $||\hat{Y_0} - C_1||^2$ is minimum, then the output metrics are 1011. Since, the value of second metric is minimum, it is set to zero and the remaining three metrics are taken to be 1. These four float values corresponds to each sub-carrier that are passed through the Viterbi block.

Viterbi Algorithm Block: It instantiates the Viterbi decoder for a sequence of K trellis epochs, whose input is a sequence of $K \times O$ values and the output is a sequence of K values. Here, O is the output cardinality of the FSM. The output of the metric block that contains soft or hard symbols and the three metrics(PS , PI and OS) of the FSM are utilized in decoding the K epochs.
In a $\frac{p}{q}$ channel encoder, for each time instant k, input x_k information is encoded to output y_k information sequence while changing the state of encoder from s_k to s_{k-1}. The three matrices provide the previous state (PS), previous input (PI) and output symbol (OS) of a FSM and each state has 2^p incoming and outgoing paths. For instance, if $\frac{p}{q} = \frac{1}{2}$, then each state has 2 incoming and outgoing paths. The three matrices of the FSM are defined below.

$$s_{k-1} = PS(s_k, i); i = 0, 1 \cdots I - 1; \tag{4.9}$$
$$x_{k-1} = PI(s_k, i); i = 0, 1 \cdots I - 1; \tag{4.10}$$
$$y_k = OS(s_k, x_k) \tag{4.11}$$

where I is the input cardinality of the FSM.
For all the states in each epoch, the values of the minimum metric and previous states are

stored in two vectors. In fact, the ACS (Add compare and select) algorithm is performed at each state to compute the minimum metric for all the 2^p incoming paths.
At the end of K epochs, the two vectors containing the minimum metric and their previous state are utilized for extracting the previous input from PI matrix. The traced path obtained from the PI matrix provides the uncoded stream.

4.4 Implementation of Coded OFDM

In all the approaches, a (2,1) trellis code with 4 state i.e, 1 bit input and two bits output is considered for channel coding. The data stream is unpacked so that each input byte into 8 single bits (carried in a byte [1]) before the trellis encoding and pack every four 2-bit worth bytes into a single byte after trellis encoding. It implies that for every information bearing byte (before unpacking) we get two coded bytes (after packing).

4.4.1 Approach 1

4.4.1.1 Transmitter

In uncoded OFDM model, the data stream in information source is encapsulated with header and CRC, which are passed through the OFDM blocks. However, the coded OFDM model contains the channel encoder that is inserted between information source and OFDM blocks. In fact, the trellis blocks in GNU Radio which are based on FSM structure are utilized for the channel encoding. The Figure 4.4 demonstrates the flow-graph structure for the coded OFDM.
The encapsulated data stream containing header and CRC fields are traversed through the message source. The message source is a queue that contains a pointer on the top of the stack(queue). The output of the message source is connected to the "pack-and-unpack" block to send the stream of correct input bits to the trellis encoder. The number of bits to the trellis encoder depends on the code rate. The trellis encoded stream is packed into bytes before sending into the message sink. The packet from the message sink are passed onto the OFDM blocks. Also, the message sink is a queue similar to the message source block.
In order to pass the packets from message sink onto the OFDM blocks, a separate thread runs in parallel to the main program. The primary objective of the thread is to observe the packets in the message queue and upon the reception of the packet, they are transferred

[1] In GNU Radio, the data stream is in the form of bytes and to input single bit to the trellis encoder block, input byte is unpacked to 8 single bits and each input bit is stored in a byte

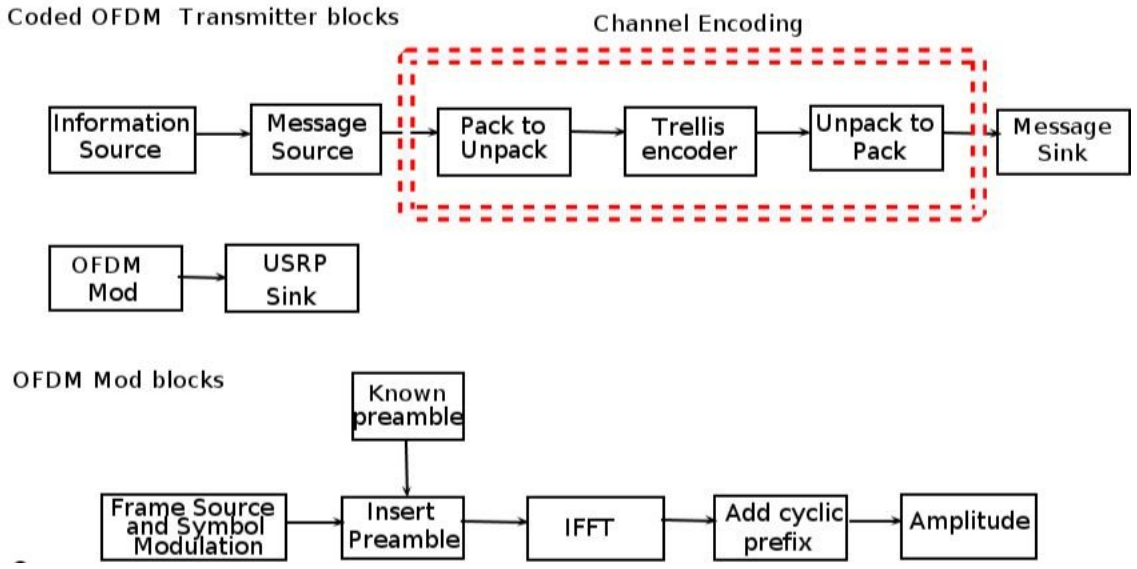

Figure 4.4: Approach 1: Flow-graph for coded OFDM transmitter

to the OFDM blocks.

4.4.1.2 Receiver

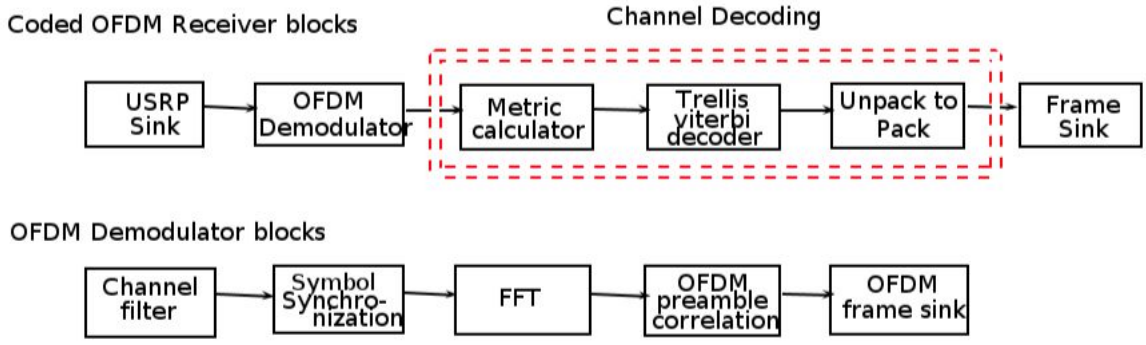

Figure 4.5: Approach 1: Flow-graph for coded OFDM receiver

In the uncoded OFDM model, the OFDM demodulator block performs the symbol synchronization, FFT, equalization and demodulation of the OFDM symbols.In fact, the demodulation and framing of the packets are performed in OFDM frame sink.
For the coded OFDM model, the output of the demodulated symbols (i.e., the output of OFDM frame sink block) is connected to the metrics calculator which produces the required input symbols to Viterbi decoder. The Viterbi decoder obtains the uncoded packet

based on trellis steps, initial and final states of the FSM. The decoded bytes are packed and sent to the frame sink. The frame sink looks for the start of the header and packs the payload and CRC in a message queue.

4.4.2 Approach 2

4.4.2.1 Problems in Approach 1

The program doesn't output any packets during the execution. The problem should be with the metric calculator and OFDM frame sink. In the receiver, the input to the metric calculator block should be synchronized OFDM symbols instead of demodulated symbols as soft decision decoding is implemented in the metric block. The possible approach is to output the soft symbols of the preamble correlation to the metric calculation and perform Viterbi decoding for the acquired soft symbols.

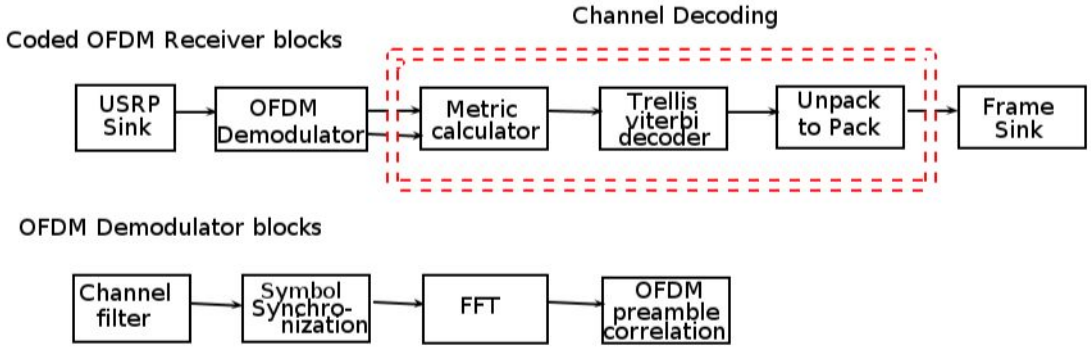

Figure 4.6: Approach 2: Flow-graph for coded OFDM receiver

4.4.2.2 Receiver

In the present approach, the output of the OFDM preamble correlation block, which provides the symbol and phase synchronized OFDM symbols are connected directly to the trellis metric block.The correlation block has two outputs. One to pass the streams of OFDM symbols, while the other contains the flag to indicate the start of OFDM symbols. The metric block is structured to perform soft decision decoding for the OFDM symbols when it receives the flag output. The corresponding metric values are passed to the Viterbi

block and then moved to the frame sink for determining the packet loss rate.
The approach has more than 90 % packet loss.

4.4.3 Approach 3 : Working Model

Problem:
The receiver and packet structure are the main problems of the former approach. The demodulated OFDM symbols which enter the trellis metric are in disarray with the transmitted OFDM symbols (i.e., OFDM symbols produced by each packet on the transmitter are not properly aligned at the receiver). Moreover, the header and CRC fields are incorrectly detected in the frame sink.
Approach:

- In order to design the coded OFDM model, the "equivalent inner channel" created between the trellis encoder and the Viterbi decoder, including the packing/message queuing/ofdm modulation/demodulation, etc, up until the input to the metric calculation block should be working perfectly. This "equivalent inner channel" has to be a byte-in byte-out channel, where each byte is carrying only 2 bits [2]. The metrics defined are "symbol-wise Hamming distance" i.e., to perform Hard Decision Decoding.
- The header and CRC fields are added to the coded data stream and the frame sink is replaced by message sink.

4.4.3.1 Transmitter

The flow-graph structure is similar to the Figure 4.4. The only difference is the encapsulation of header and CRC fields. The coded data stream in the message sink is appended with header and CRC fields that are sent to the OFDM modulator blocks.

4.4.3.2 Receiver

For coded OFDM system, the coded packets after the removal of header and CRC are passed on to a message source. They are passed through a unpack block to provide appropriate input bits to trellis metric (to perform Hard Decision Decoding) and produce required input symbols to Viterbi decoder to obtain the uncoded packet based on trellis

[2]In GNU Radio, the data stream is in form of bytes and here each byte for the inner channel contains the encoded bit-stream. Since, it is a (2,1) channel encoder, each byte contain 2 bits

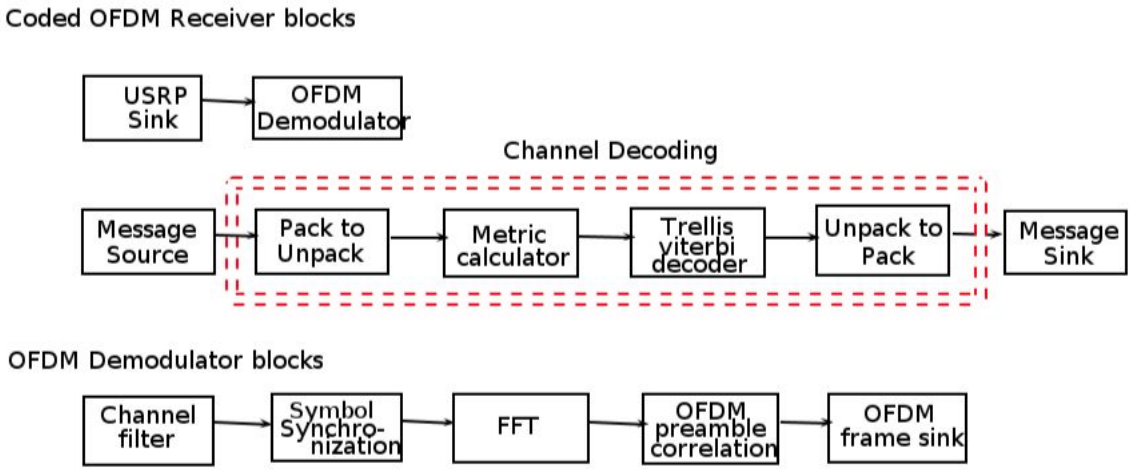

Figure 4.7: Working Model: Flow-graph for coded OFDM receiver

steps, finite state machine initial and final states. The decoded bytes consisting of payload and CRC are packed and sent to a message sink.
Two separate threads are created. One passes the coded packets from the output of message queue of OFDM frame sink onto the message source block. The other thread is used to perform the detection of received packets in the message sink.

4.5 Packet Structure and Packet Flow for Coded OFDM

4.5.1 Packet Structure

The three fields of the packet structure are header, coded_payload and CRC. The header contains two equal length fields that comprises offset and coded_payload length. The two equal length fields are used for identifying the packet at the receiver. The coded_payload is acquired through the channel encoder. The channel encoder in the figure 4.8 is a FSM convolution encoder, which is discussed in the section above. The CRC field is used for Forward error correction (FEC) of packets at the receiver. Moreover, the coded_payload and CRC fields are whitened, which involves bit-wise XOR operation with pseudo random stream of data. The packet structure for the coded OFDM model is shown in the Figure 4.8.

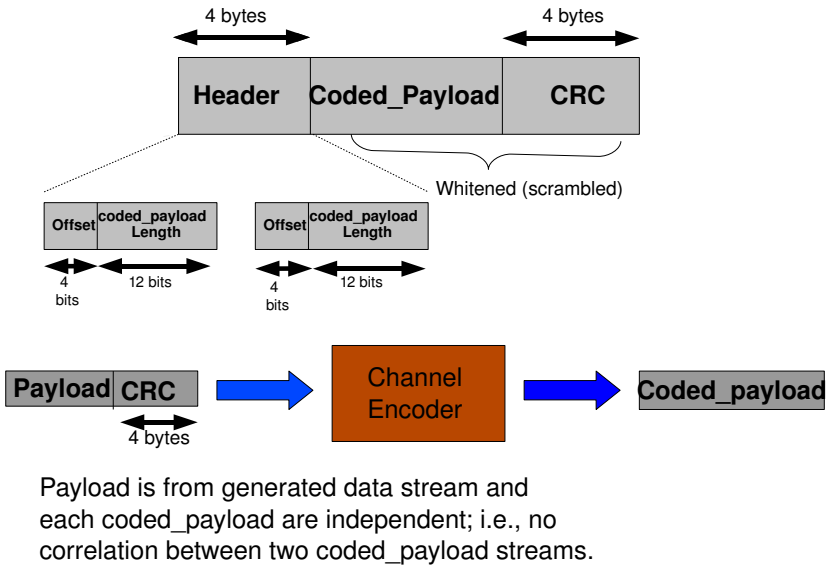

Figure 4.8: Packet structure for coded OFDM model

4.5.2 Packet Flow

4.5.2.1 Transmitter

- **Step 1**: Append CRC to the data stream payload. The payload is obtained from the information source(ex: binary or image file for transmission).
- **Step 2**: Channel encode the stream to acquire the coded payload. The encoded stream depends on the code rate.
- **Step 3**: Whiten(scramble) the data stream (coded_payload appended with CRC). It involves bit-wise XOR operation with known PN-sequence.
- **Step 4**: Generate header(4 bytes) of two equal 2 bytes consisting of offset(upper nibble) and coded_payload length(12 bits); Add header to the whitened data stream and pack them into packets, and send to the message queue.
- **Step 5**: Generate OFDM symbols from packets based on number of sub-carriers and signal constellation.
- **Step 6**: Insert known preamble; In the time domain it corresponds to one training symbol consisting of two equal length symbols.
- **Step 7**: Add cyclic prefix to the OFDM symbols

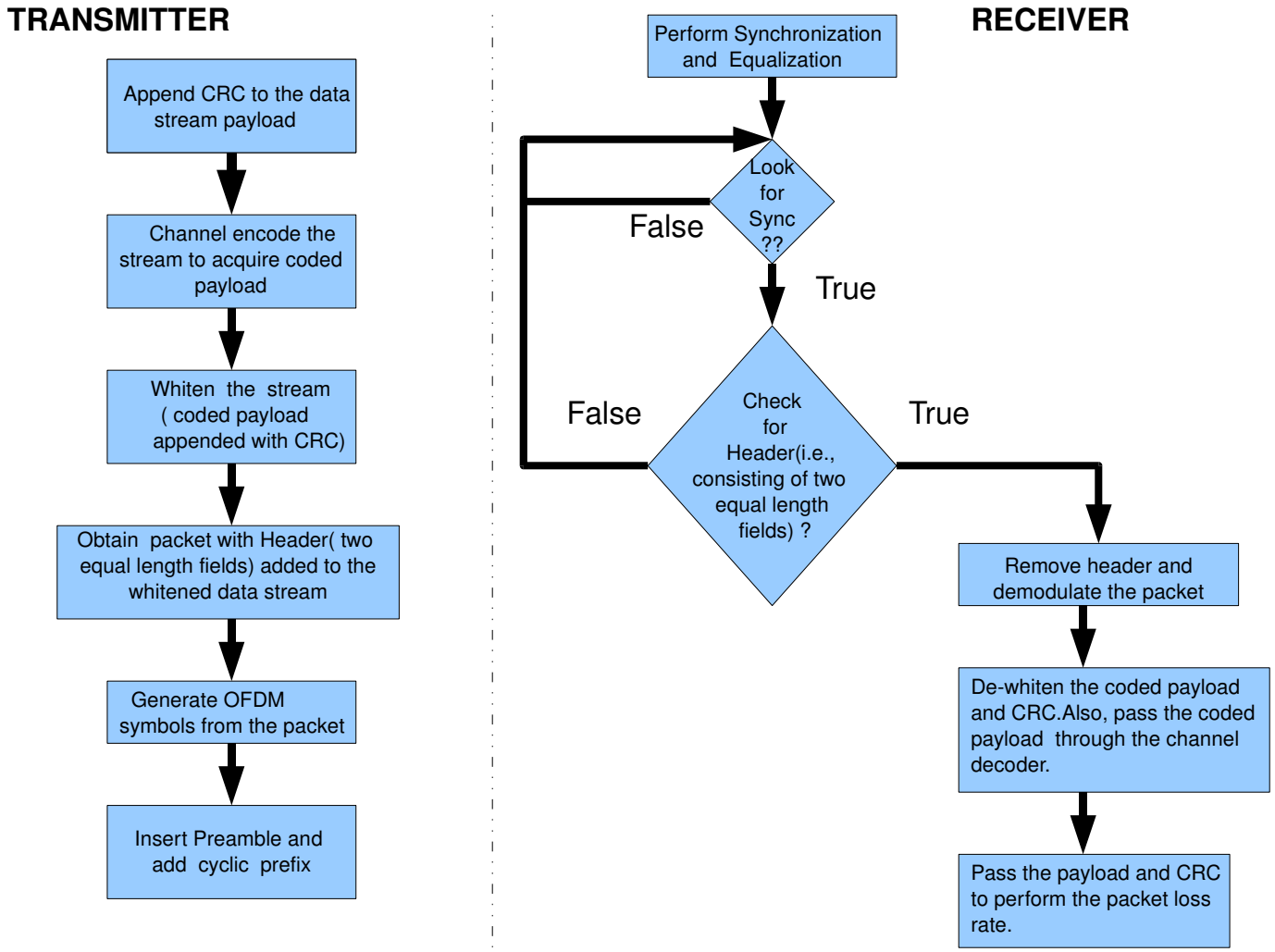

Figure 4.9: Packet flow for coded OFDM model

4.5.2.2 Receiver

- **Step 1**: Perform synchronization and equalization [16].
- **Step 2**: Look for the sync signal i.e., an indicator for start of the packet.
- **Step 3**: Check for the header with two equal length fields.
- **Step 4**: Remove the header and demodulate the OFDM symbols.
- **Step 5**: De-whiten the payload and CRC and pack coded payload into packets.
- **Step 6**: Pass the packet through a channel decoder to obtain payload and CRC.
- **Step 7**: Calculate the packet loss rate.

4.6 Problems

4.6.1 Cope with Big Burst of Errors in Coded OFDM Model

- **Modified Header**
  Problem: The data stream(payload) is passed through the channel encoder to acquire the coded payload. Each coded payload is independent i.e., there is no correlation between two coded payload streams. The coded payload is added with header and CRC before passing through the OFDM blocks. At the receiver, the demodulated OFDM symbols look for the header with two equal length fields. As a result, any false alarm of the header(incorrect header containing two equal length fields) leads to misalignment of the channel decoder with a complete loss of packets.
  Approach: The header field has two equal length fields. We included the two bit parity for each field that can be used for error detection. In fact, the header size is maintained to be 4 bytes by shortening the offset size to 2 bits. This reduces the overhead of additional bits in the header. Moreover, the receiver is designed for header verification of two equal length fields and parity check bit. This prevents the header mismatch and packet loss. The modified packet structure is shown in the Figure 4.10.
- **Interleaving**
  Appending convolution interleaving to the coded data stream reduce the burst of errors. The interleaver is attached to the output of the trellis encoder while the de-interleaver to the input of the trellis metric.
- **Packet size**
  Reducing the packet size of transmission. In fact, the trellis encoder and decoder perform better for smaller trellis steps. As a result, transmission of packets in smaller

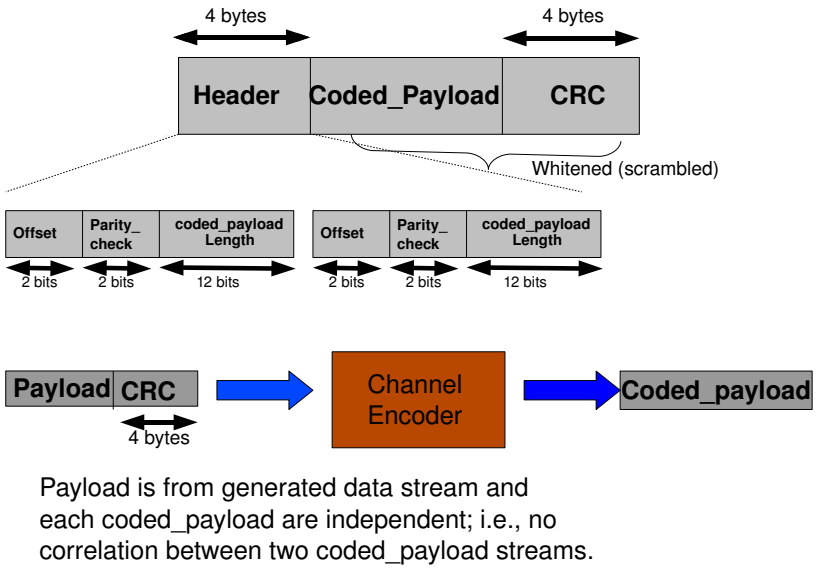

Figure 4.10: Modified Header Packet Structure

size decreases packet loss rate.
For instance, payload1 and payload2 are two equal length data stream, then payload1 + CRC through channel encoder has less than 1% packet loss and payload1+payload2+CRC stream through channel encoder has 50 to 60% packet loss. The packet loss rate is decreased with smaller packet size.

- **Sleep function**
  The use of sleep function(time.sleep()) in the queue watcher threads helps to protect the integrity of packets for trellis decoding. Two threads are watched at the receiver, one observes the message queue of the ofdm_frame_sink block to pass the data stream through channel decoder and the other thread is used to watch the output stream of the channel decoder to perform FEC(CRC check) and measure the packet loss rate. The packets for the second thread are obtained from the channel decoded stream of messages of first thread. So, the second thread is provided some transition time (time.sleep(0.01)) to reduce the packet loss rate.

4.6.2 Remarks

1. There is a significant packet loss in OFDM at the receiver due to mis-match on the preamble and miss the entire packet. The problem in capturing some of the packets seems to be at the ofdm_sync blocks in the receiver. As a result, the headers are

received incorrectly which result in misfire of the entire packet.

2. Misfire of the packets in coded OFDM model is sensitive to data rate i.e.,variations in decimation and interpolation result in better performance. For instance, decrease of interpolation and decimation rate by 2 results in 15-20 % improvement of packet loss.
   The relation between Decimation,Interpolation and Data rate for OFDM model[3]

$$\text{Data Rate}(R_b) = \frac{(\text{ADC Sampling Rate} * \text{Occupied Tones})}{(\text{No of FFT} * \text{Decimation rate})} * (\text{bits per tone})$$

$$\text{Data Rate}(R_b) = \frac{(\text{DAC Sampling Rate} * \text{Occupied Tones})}{(\text{No of FFT} * \text{Interpolation rate})} * (\text{bits per tone})$$

   Bits per tone depends on modulation type. For BPSK it is 1,QPSK it is 2.

3. In 2.4 GHz band,the receiver is prone to frequency deviation. In fact,the frequency offset(δ) of USRP is not the search range of the carrier frequency(f_c), especially for small bandwidth signals. If OFDM symbols are constantly transmitted on carrier frequency(f_c) then the receiver need to be tuned to various frequencies of $f_c \pm \delta$. Also, the value of δ varies by changing data rate.
   For instance, Interpolation:256, Decimation :128 and carrier frequency 2.2.422 GHz, then the frequency offset of the receiver is 10 kHz.

4. Without scrambling the data sequence (whitening the payload and CRC fields) all the packets are received incorrectly for both coded and uncoded system.

5. There is trade-off between overhead and system efficiency for the uncoded and coded OFDM. The overhead of a packet is the additional amount of bits/bytes appended to the payload for synchronization, equalization and error detection, whereas the system efficiency is defined as the ratio of raw bits/bytes sent to the total number of bits/bytes utilized for transmission. In uncoded OFDM, the overhead involves CRC, header, zero-padding, cyclic prefix and known preamble. However, the coded OFDM involves the corresponding fields in addition to the channel encoded bits/bytes and parity check bits of the header. For instance, if the payload length is 508 bytes, FFT length is 512, occupied tones is 200, length of cyclic prefix is 128, 32 bits of CRC, 32 bits of header, code rate of channel encoder is $\frac{1}{2}$ and QPSK modulation for each sub-carrier.
   In a uncoded OFDM, the payload(4016 bits) is appended to the 64 bits of header and CRC to generate each packet. The zero-padding bits and cyclic prefix bits are added to the OFDM symbols generated from each packet, while the preamble bits are added at the start of each packet. So, the total overhead for the uncoded OFDM is 9888 bits, while the system efficiency is provided as $\frac{4016}{4016+9888} = 0.288$.
   In a coded OFDM, the payload(4016 bits) is appended to 32 bits of CRC that are

[3]Important : Increase in data rate results in better performance as imposed to decrease its value.

passed through the channel encoder to obtain the coded payload. As the code rate is $\frac{1}{2}$, for each input bit it generates 2 output bits. The obtained coded payload is appended to 64 bits of header(including 2 bits of parity check) and CRC. The resultant coded packet are appended with zero-padding, cyclic prefix and preamble bits similar to the uncoded OFDM. The resultant overhead for the coded OFDM is 19632 bits, while the system efficiency is provided as $\frac{4016}{4016+19632} = 0.145$.
Therefore, for the coded OFDM
% Increase in overhead = 98.5 %
% Decrease in system efficiency = 49.65 %

Chapter 5

Four Node Test-bed Experiments

5.1 Motivation

To create the test-bed model for CR with 4 nodes. Also, to evaluate the co-existence of PUs and SUs under different conditions with a performance metric.

5.2 Objective

- Flow-graph model for sensing-transmission and sensing-reception-sensing-transmission cycles. Explaining the possible constraints for the implementation.
- To perform the test-bed experiments under three different scenarios for PUs and SUs and evaluating the discrepancies by examining with performance metric.
- Comparison of throughput metric for rendezvous protocols to evaluate the efficiency of communication.

5.3 Implementation

The sensing flow-graph is based on the discussion in the Chapter 2. The transmitter and receiver flow-graphs are based on packet modulator and demodulator blocks in GNU Radio.[1]

[1] digital folder in GNU Radio examples that provide continuous transmission and reception

5.3.1 Periodic Sensing-transmission Cycles

Objective: To generate alternate sensing-transmission cycles.
Approach: Two separate flow-graphs are generated as hierarchical blocks for sensing and packet transmission under the main block (the main source for all the blocks in the GNU Radio software). The flow-graphs are connected in parallel at the same time, while the set_enable(*ind*) function[2] is used to switch between the source and sink blocks of the USRP. If the *ind* is *True*, then USRP runs as transmitter and *ind* is *False*, then USRP runs as receiver. The Figure 5.1 indicates the flow-graph model for the sensing and transmission cycle.

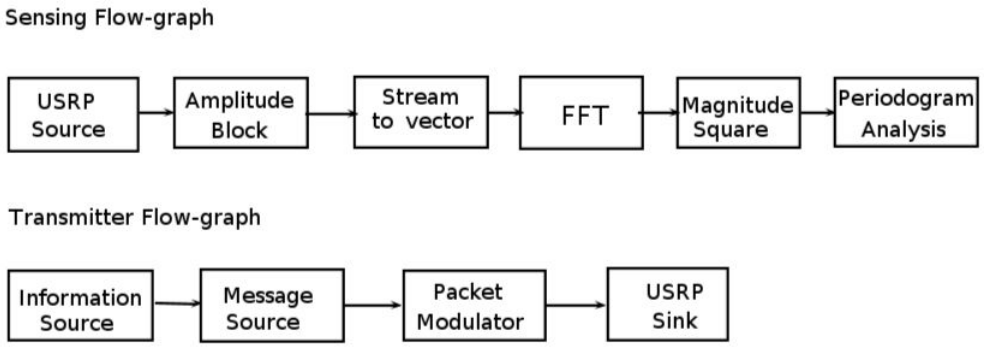

Figure 5.1: Block Diagram for sensing-transmission flow graphs

Initially, the USRP is switched to sensing mode by set_enable(*False*) and at the end of sensing cycle it is moved to transmission mode by switching set_enable(*True*). The cycles are repeated for periodic sensing and communication model. For proper operation of the cycles, transition time is inserted between the sensing and transmission cycles and vice versa(time.sleep() function is utilized). The Figure 5.2 provides the time division multiplexing of sensing and transmission periods.

5.3.2 Periodic Sensing-reception-sensing-transmission Cycles

Objective: To implement the period of sensing-reception-sensing-transmission. As USRP is a source in sensing and reception flow-graphs, only one flow-graph (sensing or reception) should be active.
Approach: All the three flow-graphs are implemented as hierarchical blocks. The transmitter flow-graph is connected all the time, but the dynamic flow graph model is implemented for sensing and reception flow-graph so that one flow graph is active for each time slot i.e.,

[2]Alternatively, set_auto_tr() function could be used for automatic switching between Tx'r and Rx'r

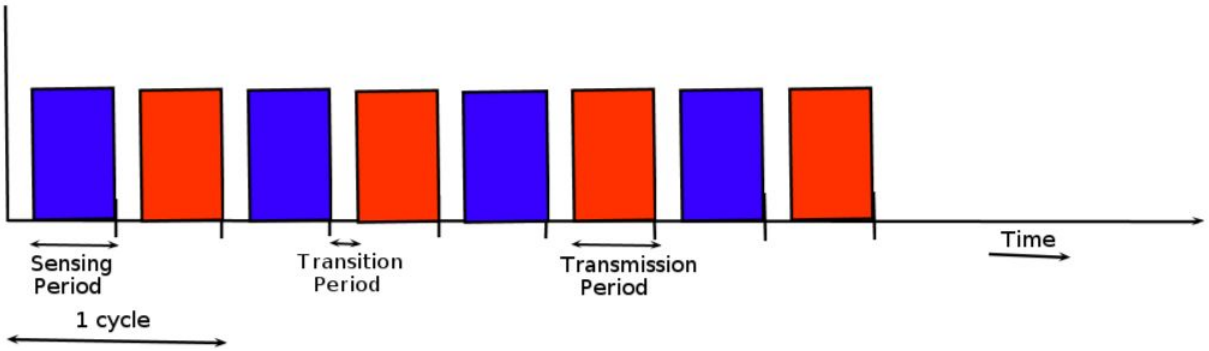

Figure 5.2: Time Division Multiplexing of sensing-transmission periods

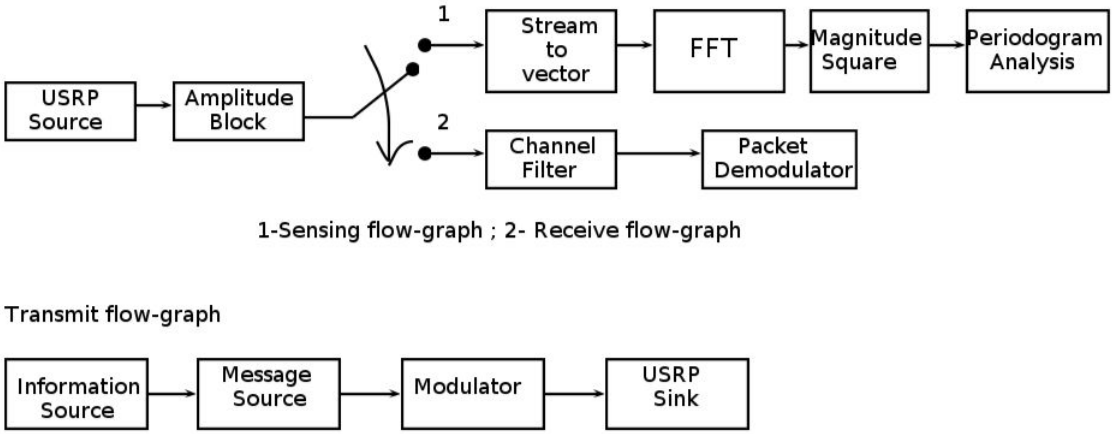

Figure 5.3: Block Diagram for sensing, reception and transmission flow graphs

during the sensing period the sensing flow-graph is active and in the reception period the reception flow-graph is active. Initially, the sensing flow-graph is connected and the lock() and unlock() function are used for dynamically switching between the two flow-graphs. The Figure 5.3 indicates the flow-graph model for the sensing, transmission and reception periods.
At the end of sensing period, the lock() method is called to disconnect the sensing blocks, and connect the receiver blocks, and the configuration ends by calling unlock() method to indicate the end of flow-graph transfer. Also, the sensing functional block(*bin_statistics_f.cc*) has been changed so as to initiate the scanning of center frequencies after the call from the python program. A method *scan_mode(ncenfreqs)* has been implemented in the functional block that passes the number of center frequencies(*ncenfreqs*) to be scanned and returns the control after the end of scanning period. This prevents the abrupt ion of the thread while using the lock() and unlock() methods to switch flow-graphs. In the reception mode, the packets are received for certain duration specified by the user. Similarly, the call blocks are repeated to switch for the sensing period.
The switching between the sensing and transmission or transmission or sensing periods is similar to section 5.3.1. The Figure 5.4 provides the time division multiplexing of sensing, reception, sensing, transmission periods.

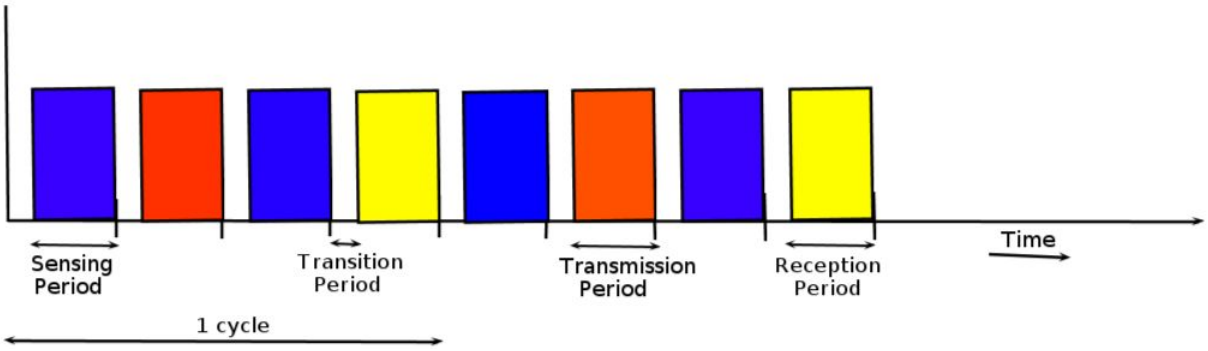

Figure 5.4: Time Division Multiplexing of sensing-reception-sensing-transmission periods

5.4 Experiments and Test-bed Results

The current test-bed model involves four USRPs, with two PUs communicating on primary channel and two opportunistic SUs that reconfigure their communication based on the primary traffic. The SUs communicate on the primary or secondary channel. The Figure 5.5 demonstrates the test-bed model of PUs and SUs.

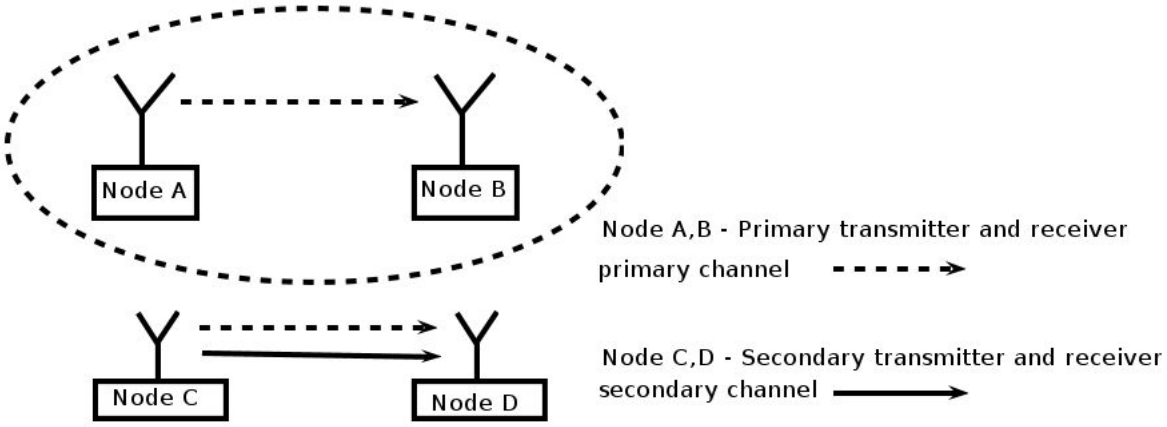

Figure 5.5: Test-bed Model for PU's and SU's

5.4.1 Scenarios and Performance Metric

Three scenarios are considered for primary and secondary traffic. The USRP is aware of the noise power of the primary channel based on the experiments for Hypothesis (H_0) discussed in the Chapter 2. Also, the sensing is performed for three center frequencies around the primary carrier frequency. For the three scenarios, the primary traffic corresponds to Markov model communication of ON and OFF cycles that are demonstrated in Chapter 3.

- scenario 1 : Single channel(primary channel) for primary and secondary traffic.
- scenario 2 : Two channels (primary and secondary channel) for communication. SUs communicate without mutual handshake.
- scenario 3 : Two channels (primary and secondary channel) for communication. SUs communicate through rendezvous protocol. We demonstrated the cases for two-way and three-way handshake. On the other hand, the method can be extended for DSA (dynamic spectrum access) of multiple channels.

Performance Metric: Interference for PUs and SUs are studied by analyzing the percentage of packet loss and error rate. They are examined for without interference, empirical threshold, false alarm (PU is absent, SU sense channel is busy) and miss detection (PU is present, SU sense channel is idle) that are obtained by varying the threshold for the energy detector(periodogram analysis).
For the three scenarios, the performance metrics provides to evaluate the interference free communication and coexistence of PUs and SUs. Also, it is used as QoS for the establish-

ment of various rendezvous protocols for SUs traffic.

$$\text{Percentage of packet loss } (\%pl) \quad = \frac{\text{No of transmitted} - \text{No of received}}{\text{No of transmitted}} \tag{5.1}$$

$$\text{Percentage of error rate } (\%er) \quad = \frac{\text{No of received} - \text{No of correct}}{\text{No of received}} \tag{5.2}$$

5.4.2 Scenario 1: Single (Primary) Channel for Communication

PUs and SUs communicate on single channel, but SUs utilizes the channel only when it senses that there is no PU communication.
Secondary Traffic: At any given time, the secondary transmitter operates in three modes; sensing, idle or transmission mode. In sensing mode, it senses the primary channel to evaluate the average power of the channel. In idle mode, it remains silent for the length of the duration specified by the user. In transmission mode, it transmits a fixed number of packets. Initially, it senses the primary channel and before returning to the sensing mode, it moves to idle or transmission mode based on the sensing decision. Usually, if the channel is busy, it remain for a short period in idle mode and switch back to sensing mode, but if the channel is deemed free, then it transmit data packets over the primary channel. The program uses the periodic sensing and transmission that are explained in the section 5.3.1 to alternatively switch between sensing and transmission mode. Similarly, the secondary receiver remain in idle or reception mode. The idle mode is similar to the secondary transmitter and for the reception mode it receives packets for a certain duration.[3]

5.4.2.1 Primary Traffic : Parameters and Assumptions

The primary traffic uses DBPSK modulation for its transmission. Assume the number of packets available for transmission are 10,15,20. The ON periods are calculated based on data rate, packet size and packet overhead. Also, the ON periods has uniform distribution and their selection depends on prior probabilities. The corresponding OFF periods are 0.51, 1.32, 2.55 seconds that are determined from algorithm in Appendix A. The parameters of primary traffic are

- Modulation : DBPSK ; Excess Bandwidth = 0.35
- Interpolation rate : 512 ; Samples per symbol : 2
- Carrier Frequency(f_c) : 2.422 GHz

[3]The program is based on the packet demodulator block in GNU Radio.

- Packet size=4000 bytes ; Packet overhead = 20 bytes
- Data Rate[4] =125Kb/sec
- Transition Matrix(P) = $\begin{bmatrix} 0.25 & 0.75 \\ 0.35 & 0.65 \end{bmatrix}$
- Pr(10) = 0.45; Pr(15) = 0.3; Pr(20) = 0.25
- $T_{ON10} = 0.25728secs$; $T_{ON15} = 0.38592secs$; $T_{ON20} = 0.51456secs$
- $T_{OFF10} = 0.51secs$; $T_{OFF15} = 1.32secs$; $T_{OFF20} = 2.55secs$

5.4.2.2 Secondary Traffic :Parameters and Assumptions

The secondary traffic uses Coded OFDM model discussed in Chapter 4. We evaluate the performance metrics for short and long burst. The transmission periods for number of packets depends on data rate, packet size and packet overhead. The sensing period depends on dwell delay and tune delay for each of the center frequencies.
We considered the case of 30 and 100 packets for short and long burst. There are only three center frequencies and the sensing period is $3 * (t_d + t_n); t_d = 10msec$ and $t_n = 1msec$. The parameters for secondary traffic are

- Modulation : Coded OFDM (QPSK); Code Rate = 1/2
- Interpolation : 256; Samples per symbol=2
- Occupied tones: 200; FFT length =512
- Scanning Frequency Range :2.5 MHz i.e., considering three center frequencies around carrier (2.4205 GHz-2.423 GHz)
- Packet size =508 bytes ; Packet Overhead = 525 bytes
- Data Rate [5]=390.625 Kb/sec
- Transmission time for short burst(30 packets) $T_{t\_30} = 0.634secs$
- Transmission time for long burst(100 packets) $T_{t\_100} = 2.115secs$
- Sensing period for three center frequencies $T_s = 33msecs$

[4] $DataRate(R_b) = \frac{(ADCSamplingRate)}{(Samplespersymbol * Interpolationrate)} * (bitspersymbol); ADCSamplingRate : 128MS/sec$

[5] $DataRate(R_b) = \frac{(ADCSamplingRate * OccupiedTones)}{(FFTsize * Interpolationrate)} * (bitspertone)$

We evaluate the performance for short and long burst of traffic from SU. The packet loss and error rate are calculated based on equations (5.1), (5.2)

- Total number of packets for transmission(PU): 200
- Total number of packets for transmission (SU): 1000
- Threshold for Energy Detector : τ
- Average Power Statistics for PU :$P_{avg} = 54.5$
  where the average power statistics are similar to the mean of average power for H_1 hypothesis in Chapter 2.

5.4.2.3 Performance of PU

case(i): Short Burst of Traffic from SU
The Table 5.1 provides the percentage of packet loss and error rate of PU with short bust of traffic from SU.

Table 5.1: Performance of PU: short burst of traffic from SU

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No SU Traffic	200	197	0	1.5
Empirical Threshold($\tau(55) \cong P_{avg}$)	171	165	14.5	3.5
Miss Detection ($\tau(70) > P_{avg}$)	87	63	56.5	27.5
False Alarm ($\tau(45) < P_{avg}$)	135	120	32.5	11.11

case (ii):Long Burst of Traffic from SU
The Table 5.2 provides the percentage of packet loss and error rate of PU with long bust of traffic from SU.

Table 5.2: Performance of PU: long burst of traffic from SU

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No SU Traffic	200	197	0	1.5
Empirical Threshold($\tau(55) \cong P_{avg}$)	85	82	57.5	3.5
Miss Detection ($\tau(70) > P_{avg}$	40	35	80	12.5
False Alarm ($\tau(45) < P_{avg}$	60	55	70	8.33

5.4.2.4 Performance of SU

case(i): Short Burst of Traffic from SU
The Table 5.3 provides the percentage of packet loss and error rate of SU with short bust of traffic from SU.

Table 5.3: Performance of SU: short burst of traffic from SU

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No PU Traffic	768	745	20.2	2.55
Empirical Threshold($\tau(55) \cong P_{avg}$)	727	698	27.3	3.98
Miss Detection ($\tau(70) > P_{avg}$)	717	687	28.3	4.18
False Alarm ($\tau(45) < P_{avg}$)	715	682	28.5	4.615

case (ii) : Long Burst of Traffic from SU
The Table 5.4 provides the percentage of packet loss and error rate of SU with long bust of traffic from SU.

Table 5.4: Performance of SU: long burst of traffic from SU

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No PU Traffic	840	830	16.0	1.19
Empirical Threshold($\tau(55) \cong P_{avg}$)	827	815	17.3	1.25
Miss Detection ($\tau(70) > P_{avg}$)	726	713	27.4	1.79
False Alarm ($\tau(45) < P_{avg}$)	749	738	25.1	1.468

5.4.2.5 Special Case:Long Burst of Traffic from PU and SU

From the Table 5.2, there is a complete loss of packets due to secondary interference (big burst of data). In order to avoid the interference we need to increase the number of packets during ON cycle i.e., transmission time of PU is longer and enabling SU to detect PU. The PU traffic are changed as
Transmitting 40,60,80 packets during ON cycle.
OFF periods ; 5 ,6 and 8 seconds.

Performance of Primary User Traffic.
The Table 5.5 provides the percentage of packet loss and error rate of PU with long bust of traffic from PU and SU.
Performance of Secondary User Traffic

Table 5.5: Performance of PU:long burst of traffic from PU and SU

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No SU Traffic	200	200	0	0
Empirical Threshold($\tau(55) \cong P_{avg}$)	168	167	16	0.5
Miss Detection ($\tau(70) > P_{avg}$)	102	92	49	9.8
False Alarm ($\tau(45) < P_{avg}$)	138	128	31	7.24

The Table 5.6 provides the percentage of packet loss and error rate of SU with long bust of traffic from PU and SU.

Table 5.6: Performance of SU: long burst of traffic from PU and SU

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No PU Traffic	847	836	15.3	1.29
Empirical Threshold($\tau(55) \cong P_{avg}$)	800	786	20	1.75
Miss Detection ($\tau(70) > P_{avg}$)	719	694	28.1	3.47
False Alarm ($\tau(45) < P_{avg}$)	743	733	25.7	1.34

Remarks:

1. From the Table 5.2, there is a large percentage of packet loss and error rate for the PU with long burst of traffic from SU.
2. The large percentage of packet loss and error rate for the PU can be reduced by lengthening the transmission periods of the ON traffic for the PU. As it is shown in Table 5.5 and 5.2, there is decrease of 40 % packet loss and error rate, when long burst of traffic is performed for the PU
3. Shorter idle periods are used for the secondary transmitter. Secondary transmitter senses the primary channel and switches to idle or transmission mode based on the sensing decision. At the end of idle or transmission mode, it switch back to sensing period. So, the shorter idle periods increases the probability of detection of the PUs.

5.4.3 Scenario 2: Two Channels (Without Handshaking)

We assume, there are two channels for SU's communication and the secondary channel is free all the time. There is no synchronization of SU's for the data traffic exchange.
Protocol for synchronization of secondary traffic:
There is no mutual handshake and the exchange of communication takes place with the

initiation of a beacon signal. The secondary transmitter sends a beacon signal before the data transmission on either channel(primary or secondary) based on sensing decision for primary traffic. The secondary receiver remains on the channel if there is a beacon signal or hops between the primary and secondary channel. For the secondary receiver, the length of the duration it remains for the beacon signal on a particular channel is known as contention period. The flow-graph model of sensing transmission cycle in section 5.3.1 is used for the secondary transmitter and packet demodulator blocks are modified for the secondary receiver. The duty cycle for SUs and secondary receiver channel usage is demonstrated in the Figure 5.6.

We tried to use the same parameters as discussed in scenario 1, but Coded OFDM is very sensitive to packet loss and there is high probability for loss of beacon signal in transmission. Thus, we intend to use Coded OFDM for primary traffic and DBPSK modulation for secondary traffic. Moreover, the two carrier frequencies are considered as 2.457GHz (channel 10) and 2.462GHz (channel 11) in 802.11 channels which are less suspectable to interference compared to 2.422GHz (channel 3).

5.4.3.1 Primary Traffic: Parameters and Assumptions

The primary traffic uses coded OFDM modulation. Assume, the number of packets available for transmission are 40,60,80. The ON periods are calculated based on data rate, packet size and packet overhead. Also, the ON periods has uniform distribution and their selection depends on prior probabilities. The corresponding OFF periods are considered as 1.692, 5.072, 8.46 secs that are determined from algorithm in Appendix A. The parameters for primary traffic are

- Modulation : Coded OFDM (QPSK) ; Code Rate = 1/2
- Interpolation = 128 ; Samples per symbol=2
- Occupied tones: 200; FFT length =512
- Packet size =508 bytes ; Packet Overhead = 525 bytes
- Primary Carrier Frequency(f_{c1}) : 2.457 GHz
- Data Rate [6] = 780.25 Kb/sec
- Transition Matrix(P) = $\begin{bmatrix} 0.25 & 0.75 \\ 0.35 & 0.65 \end{bmatrix}$
- Pr(40) = 0.25; Pr(15) = 0.35; Pr(20) = 0.40

[6] $DataRate(R_b) = \frac{(ADCSamplingRate * OccupiedTones)}{(FFTsize * Interpolationrate)} * (bitspertone)$; $ADCSamplingRate : 128MS/sec$;

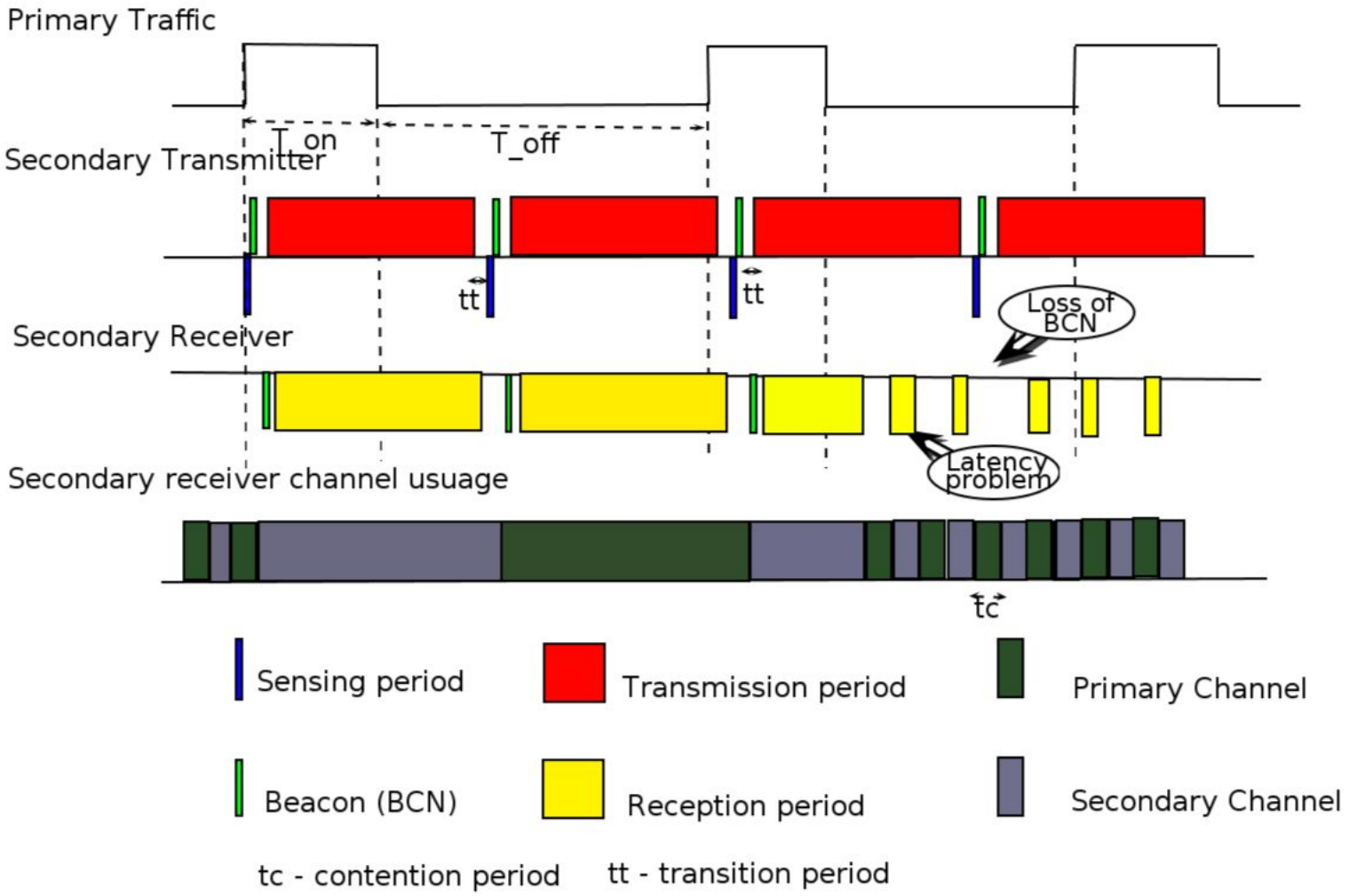

Figure 5.6: Duty cycle of SU's for two channel and without mutual handshake

- $T_{ON40} = 0.846secs;\ T_{ON60} = 1.268secs;\ T_{ON80} = 1.692secs$
- $T_{OFF40} = 1.692secs;\ T_{OFF60} = 5.072secs;\ T_{OFF80} = 8.46secs$

5.4.3.2 Secondary Traffic: Parameters and Assumptions

Transmission period depends on data rate, packet size and packet overhead. The sensing period is same as the scenario 1.

- Modulation : DBPSK ; Excess Bandwidth = 0.35
- Interpolation rate : 256 ;Samples per symbol : 2
- Scanning Frequency Range : 2.5 MHz i.e., three center frequencies around carrier (2.4205-2.423 GHz)
- Secondary Carrier Frequency(f_{c2}) : 2.462 GHz
- Packet size=4000 bytes;Packet overhead = 20 bytes
- Data Rate [7] = 250 Kb/sec
- Tran mission time for long burst(100 packets) $T_{t\_100} = 4.864secs$
- Sensing period for three center frequencies $T_s = 33msecs$

We evaluate the performance of PUs and SUs besides mentioning the synchronization problems of secondary traffic. The packet loss and error rate are calculated based on equations (5.1), (5.2)

- Total number of packets for transmission(PU): 1000
- Total number of packets for transmission(SU): 1000
- Threshold for Energy Detector : τ
- Average Power Statistics for PU :$P_{avg} = 46.5$
  where, the average power statistics are obtained by performing the average power analysis for Coded OFDM parameters under hypothesis H_1 as discussed in section 2.4, Chapter 2.

[7] $DataRate(R_b) = \frac{(ADCSamplingRate)}{(Samplespersymbol * Interpolationrate)} * (bitspersymbol)$; $ADCSamplingRate : 128MS/sec$

5.4.3.3 Performance of PU

The Table 5.7 provides the percentage of packet loss and error rate of PU for two channel without handshaking.

Table 5.7: Performance of PU for two-channel without hand-sake

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No SU Traffic	991	989	0.9	0.2
Empirical Threshold($\tau(45) \approxeq P_{avg}$)	864	832	13.6	3.7
Miss Detection ($\tau(60) > P_{avg}$)	665	625	33.5	6.01
False Alarm ($\tau(35) < P_{avg}$)	824	786	17.6	4.61

5.4.3.4 Performance of SU

The Table 5.8 provides the percentage of packet loss and error rate of SU for two channel without handshaking.

Table 5.8: Performance of SU for two-channel without hand-sake

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No PU Traffic	509	432	49.1	15.12
Empirical Threshold($\tau(45) \approxeq P_{avg}$)	506	463	49.4	8.49
Miss Detection ($\tau(60) > P_{avg}$)	517	488	48.3	5.60
False Alarm ($\tau(35) < P_{avg}$)	523	496	47.7	5.16

Remarks:

1. There is no mutual handshake between the two SUs before the start of data communication. It results in large percentage of packet loss and error rate which is demonstrated in the Table 5.8.

2. In the synchronous protocol, the secondary receiver hops between primary and secondary channel if there is no beacon signal for the contention period. For the implementation on GNU Radio, the main thread in python awaits for the packet(beacon signal) in the message sink. The message sink acquires the packets from the demodulated blocks (implemented on C++) of the receiver. The latency for the packet to pass through the blocks and return the call to python abrupt the main thread which results in switching between the two channels. The problem causes the receiver to

hop between the channels even if there is continuous burst of traffic. The pictorial representation of latency problem is shown in the Figure 5.6.

3. Interference of PU's and SU's is accountable for the packet loss. The loss of control packet results in complete loss of data packets for the SU. The Figure 5.6 demonstrates the loss of beacon signal and the interference of PUs and SUs.

5.4.4 Scenario 3 : Rendezvous Communication (Two Channels)

5.4.4.1 Rendezvous Communication: Two-way Handshaking

The rendezvous protocol [18] uses mutual handshake in advance to actual data traffic. The secondary nodes accomplish handshaking by transmitting periodic beacon signals on free channels and awaits for an acknowledgment signal. The handshaking completes, when a node receives an acknowledgment signal and start communicating for data traffic. We assume that there are two channels for SU's communication.
Protocol for synchronization of secondary traffic:
There are two modes of operation; synchronous and data traffic modes. In the synchronous mode, the nodes meet on a control channel with an exchange of beacon and acknowledgment signal i.e., mutual handshaking between the nodes. To achieve handshaking, nodes send a periodic beacon signal(containing channel information) on free channels. If there is a node that listens to a beacon, then it responds immediately with an acknowledgment signal. The reception of an acknowledgment signal completes synchronous mode and both switch to the data traffic mode.
In data traffic mode, the node that receives acknowledgment signal acts as the secondary transmitter to send data traffic (predefined no of packets), the other node the one who receives beacon signal acts as the secondary receiver to listen to the data traffic. At the end of data traffic, each node sense communicating channel to switch to synchronous mode(channel is busy) or remain in data traffic mode(channel is idle) i.e.,the exchange of communication takes place until the channel is occupied by the PU. The flow-graph and duty cycle of SU's for two-way handshake is shown in the Figure 5.7 and 5.8.

5.4.4.2 PU and SU Traffic

All the parameters for PUs and SUs are same as scenario 2.
Remarks
There are two cases that might result in disruption of communication.

1. In data traffic mode, if a node stagnates on a channel as the other node vacates the channel at the end of data traffic (advances to synchronous mode; the node sense the

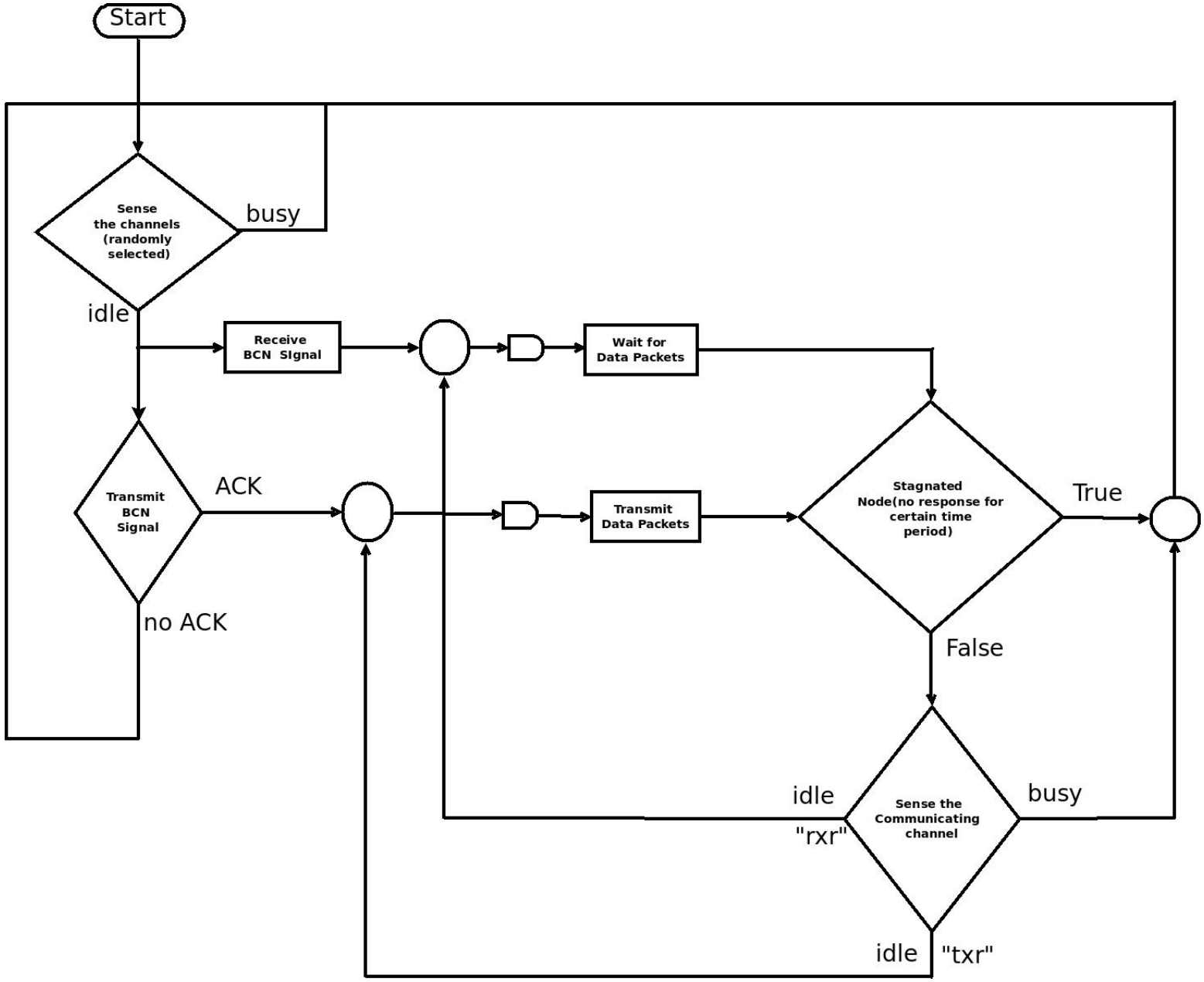

Figure 5.7: Flow Chart for two-way handshake

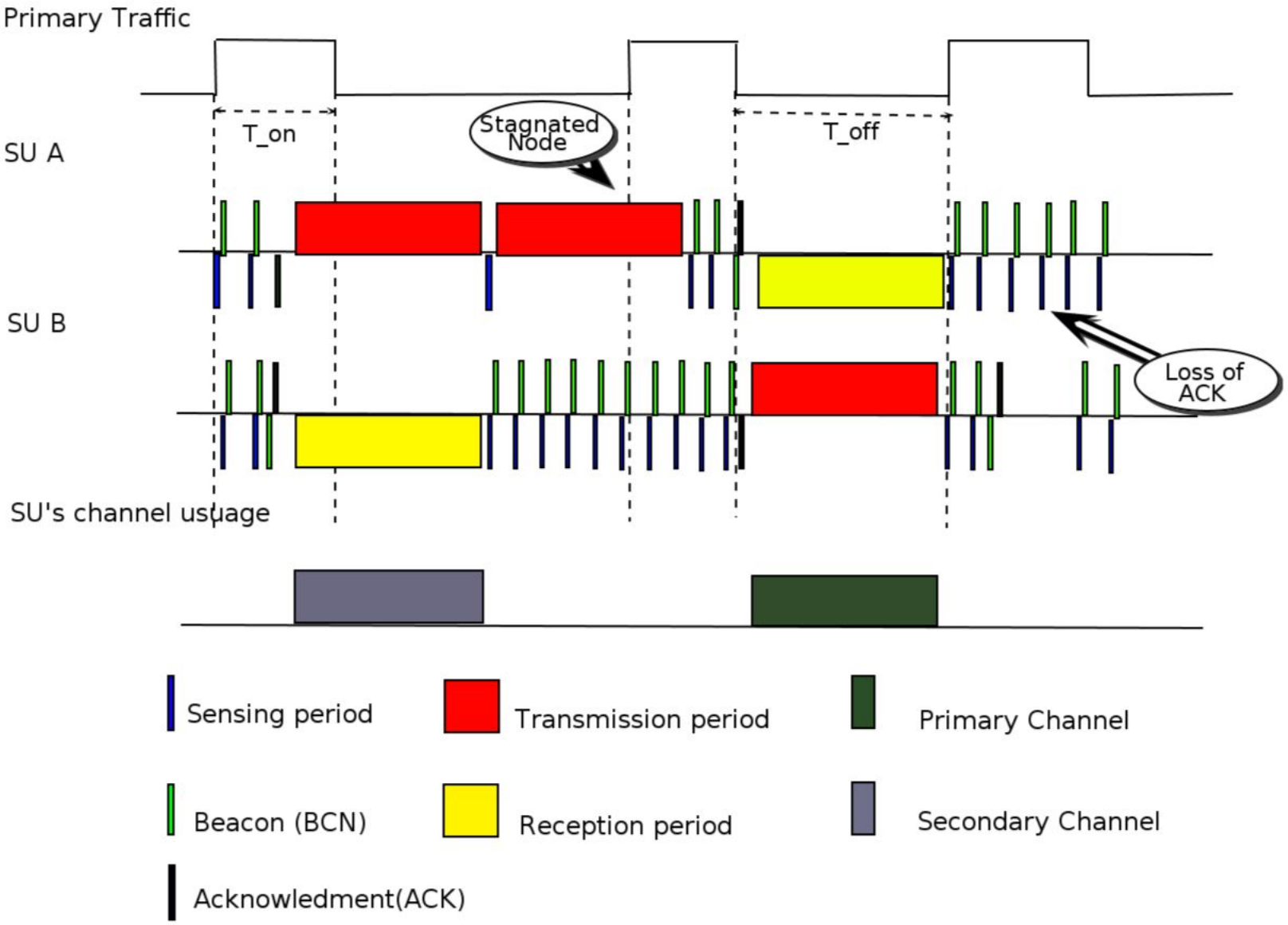

Figure 5.8: Duty cycle of SU's for two channel and with two-way handshake

Table 5.9: Performance of PU for two-channel with two-way handshake

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No SU Traffic	970	969	0.03	0
Empirical Threshold($\tau(45) \cong P_{avg}$)	974	958	0.26	16.42

Table 5.10: Performance of SU for two-channel with two-way handshake

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No PU Traffic (Node A)	620	384	38.0	59.03
Empirical Threshold (Node A)	634	298	36.6	52.99
No PU Traffic (Node B)	468	204	53.2	56.41
Empirical Threshold (Node B)	710	208	29.0	50.2

channel is busy). The stagnated nodes tries to communicate for a certain period. It switches to synchronous mode if there is lack of response from the other node. It stagnated node problem causes packet loss for the SUs. The Figure 5.8 demonstrates the stagnated node problem.

2. Loss of an acknowledgment signal might cause one node to be in synchronous mode, while the other is still in data traffic mode. Similar to case (i) the node in data traffic mode remains in that mode for a certain duration and returns to synchronous mode due to lack of response on the channel. The Figure 5.8 shows the loss of acknowledgment signal.

The Table 5.10 demonstrates the percentage of packet loss and error rate for the SU due to stagnated node and loss of acknowledgment signal.

5.4.4.3 Rendezvous Protocol : Three - way Handshake

Motivation for three-way protocol
1. Duty cycles for two nodes are asynchronous.i.e., lack of time synchronization.
2. Loss of beacon and acknowledgment signal.
3. Stagnated Node Problem.
It is based on the lines of two-way handshaking protocol.
Protocol for synchronization of secondary traffic: We assume there are two channels for communication. For the protocol, there are two modes of operation; synchronous and data traffic mode. In synchronous mode, each node transmits a beacon signal and waits for an acknowledgment signal, with the reception of acknowledgment signal, the node transmits ack-ack(to convey the reception of acknowledgment signal so as to reduce stagnated node problem) signal to establish communication and traverses to data traffic mode, while the

other node on reception of ack-ack signal moves to data traffic mode. In addition, synchronous traffic signals contains channel information, node representation (Tx'r or Rx'r) and packet number.
In data traffic mode, the exchange of data traffic is performed with the transmitter and receiver rendezvous communication on agreed channel. At the end of data traffic, each node traverse to the synchronous mode to establish the communication. The flow-graph model and duty cycle of SU's for the demonstration of three-way handshake are shown in Figures 5.9 and 5.10.

5.4.4.4 MAP Testing

It is a decision rule to distinguish between the two hypothesis based on the collected statistics. The density functions of statistics and prior probabilities for each hypothesis are used in MAP testing. Consider, the collected statistics ($\underline{z}$) for the two hypothesis H_0 and H_1. The MAP testing is defined from [15].

$$\frac{P(\underline{z}/H_0)}{P(\underline{z}/H_1)} \underset{H_1}{\overset{H_0}{\gtrless}} \frac{\pi_1}{\pi_0}; \quad \text{where, } P(H_0) = \pi_0 \text{ and } P(H_1) = \pi_1 \tag{5.3}$$

For our experiments, the average power statistics (P_{avg}) are used to distinguish between the two hypothesis H_0 (Primary traffic is OFF) and H_1 (Primary traffic is ON). The density function of P_{avg} for H_0 is obtained from section 2.4 in Chapter 2, while the density function of P_{avg} for H_1 is acquired by performing the similar experiment with Coded OFDM modulation for PU. The experiments provide that both statistics follow Gaussian distribution i.e.,

$$P(P_{avg}/H_0) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp^{-\frac{(z-\mu_0)^2}{2\sigma_0^2}}; \quad \mu_0 = 20.49 \text{ and } \sigma_0 = 0.063 \tag{5.4}$$

$$P(P_{avg}/H_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp^{-\frac{(z-\mu_1)^2}{2\sigma_1^2}}; \quad \mu_1 = 44.040 \text{ and } \sigma_1 = 2.897 \tag{5.5}$$

The prior probabilities for the two hypothesis are evaluated from transition matrix (P). In fact, the transition matrix(P) for the Markov traffic of PUs is chosen as $P = \begin{bmatrix} 0.25 & 0.75 \\ 0.35 & 0.65 \end{bmatrix}$ which is used to obtain the ratio of prior probabilities from Appendix A. So, the ratio is given as

$$\frac{\pi_1}{\pi_0} = \frac{0.75}{0.35} \tag{5.6}$$

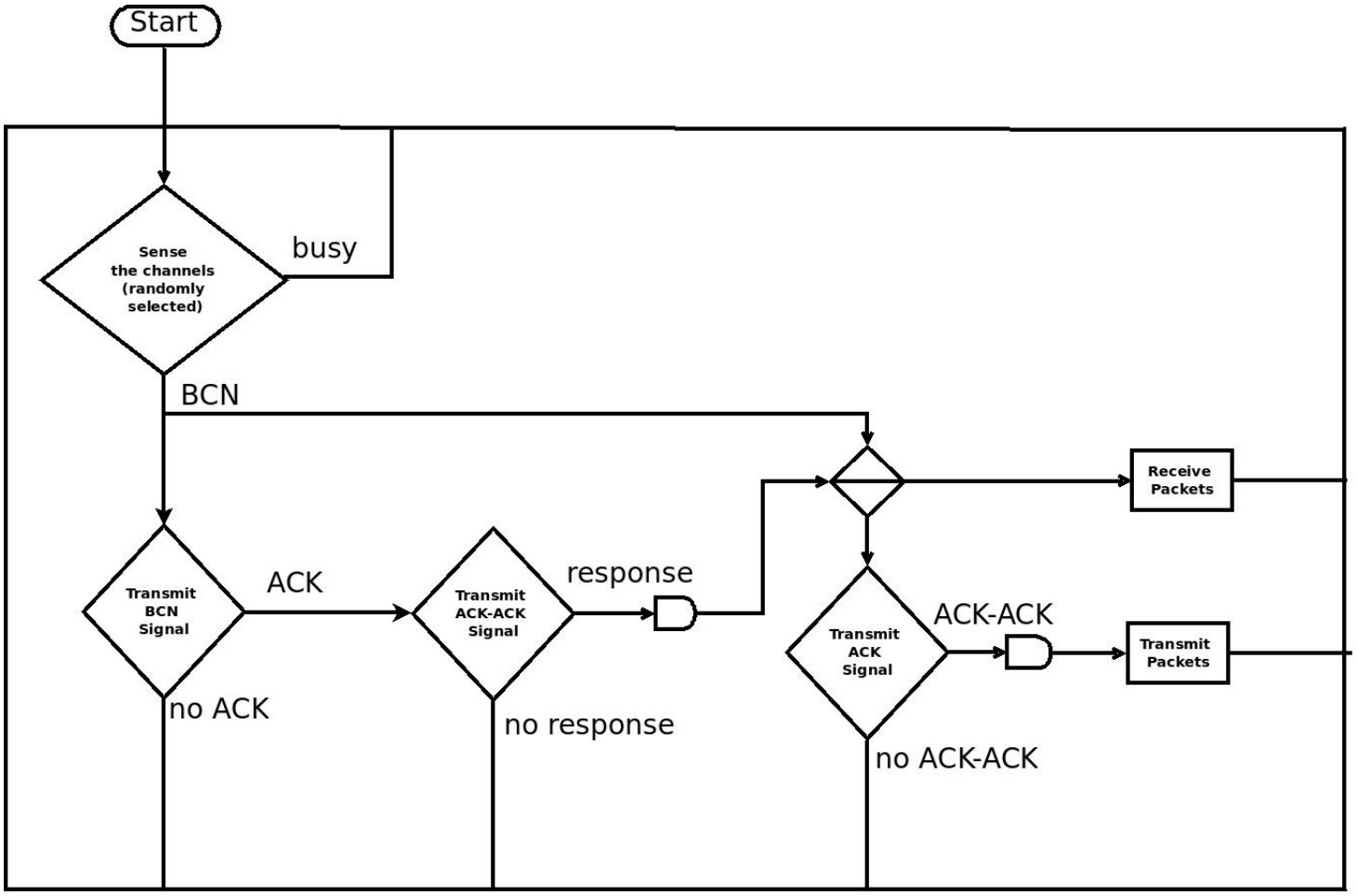

Figure 5.9: Flow Chart for three-way handshake

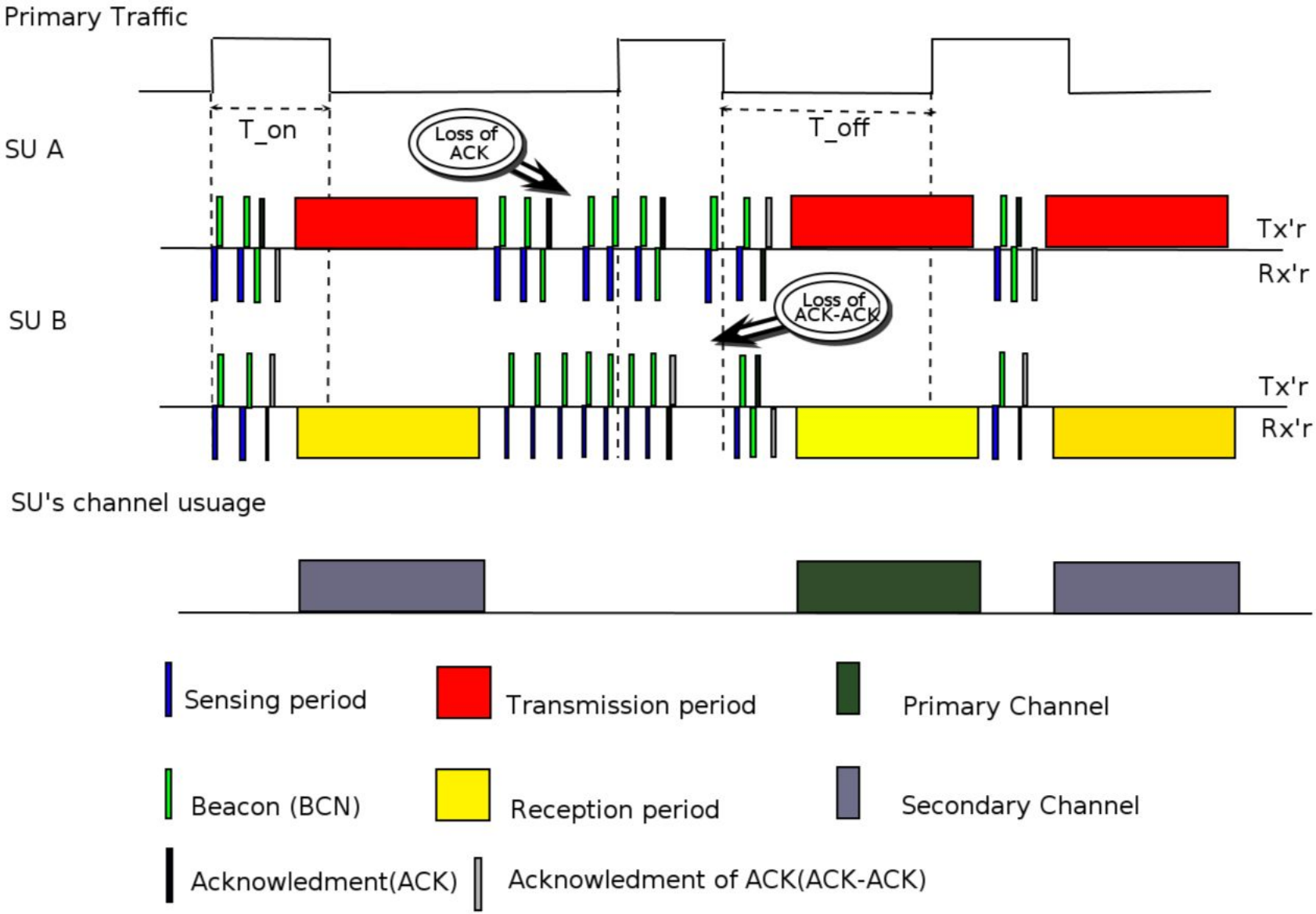

Figure 5.10: Duty cycle of SU's for two channel and with three-way handshake

From equation 5.3 to 5.6, the resultant equation is provided as

$$\frac{P(P_{avg}/H_0)}{P(P_{avg}/H_1)} \underset{H_1}{\overset{H_0}{\gtrless}} \frac{\pi_1}{\pi_0}; \tag{5.7}$$

The collected statistics for each time instant are substituted in equation 5.7 to determine the presence or absence of primary traffic. The tables 5.11 to 5.14 provides the percentage packet loss and error rate for MAP testing.

5.4.4.5 Primary and Secondary Traffic Performance

The performance of PUs and SUs are evaluated for two different modulations(DBPSK, GMSK) of secondary traffic. Also, the MAP testing is performed in each case. The remaining parameters are same as scenario 2.
Average power statistics of the channel(P_{avg}) = 44.05.
where, the average power statistics are obtained by performing the average power analysis for Coded OFDM parameters under hypothesis H_1 as discussed in section 2.4, Chapter 2.

case(i): Secondary Traffic Modulation (DBPSK)

Table 5.11: Performance of PU for two-channel with three-way handshake(DBPSK)

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No SU Traffic	985	960	1.5	2.53
Empirical Threshold($\tau(45) \approxeq P_{avg}$)	958	943	4.2	1.56
Miss Detection ($\tau(60) > P_{avg}$)	885	870	11.5	1.69
False Alarm ($\tau(35) < P_{avg}$)	951	937	4.9	1.47
MAP testing	970	958	3.0	1.28

Table 5.12: Performance of SU for two-channel with three-way handshake(DBPSK)

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No PU Traffic	956	879	4.4	8.05
Empirical Threshold($\tau(45) \approxeq P_{avg}$)	946	861	5.4	8.98
Miss Detection ($\tau(60) > P_{avg}$)	915	821	8.5	10.27
False Alarm ($\tau(35) < P_{avg}$)	923	872	7.7	5.52
MAP testing	934	858	6.6	5.48

case(ii): Secondary Traffic Modulation(GMSK)

Table 5.13: Performance of PU for two-channel with three-way handshake(GMSK)

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No SU Traffic	985	960	1.5	2.53
Empirical Threshold($\tau(45) \cong P_{avg}$)	968	951	3.2	1.75
Miss Detection ($\tau(60) > P_{avg}$)	901	867	9.9	3.77
False Alarm ($\tau(35) < P_{avg}$)	959	948	4.1	1.14
MAP testing	952	940	4.8	1.26

Table 5.14: Performance of SU for two-channel with three-way handshake(GMSK)

Case	No of received	No of Correct	%Packet Loss	%Packet Error
No PU Traffic	992	988	0.8	0.40
Empirical Threshold($\tau(45) \cong P_{avg}$)	974	965	2.6	0.92
Miss Detection ($\tau(60) > P_{avg}$)	936	919	6.4	1.81
False Alarm ($\tau(35) < P_{avg}$)	948	920	5.2	2.95
MAP testing	958	946	4.2	1.25

Remarks

1. The major problem for the two-way handshake is the stagnated node where the node remain in data traffic mode while the other node sends out beacon signal for handshaking. We address the problem by providing the mutual handshake after the completion of each data traffic burst. Also, we modeled the protocol so that one node is a receiver while the other remain as a transmitter for the data traffic.

2. From the Tables 5.12 and 5.14, it provides that using GMSK for SUs results in less percentage of packet loss and error rate compared to DBPSK for all the cases(empirical threshold, false alarm, miss detection and MAP testing). Especially, for the empirical threshold there is decrement of 8% in error rate for GMSK and DBPSK. So, it is better to use GMSK for the three-way rendezvous protocol of SUs.

3. Although there loss of acknowledgment and ack of acknowledgment in the three-way handshake. It provides better performance rather than two-way handshake by eliminating the stagnated node problem. The Tables 5.10 and 5.14 demonstrates that there is reduction of 30 % of packet loss and 50 % of packet error for empirical threshold in a three-way handshake protocol.

4. The three-way handshake provides better performance compared to two-channel model without handshaking. The Tables 5.14 and 5.8 demonstrates that the percentage of packet loss and error rate are decreased by more than 40 % for the SUs.

5.4.4.6 Comparison of Throughput Metric of SUs for Handshaking Protocols

Throughput metric is defined as the average rate of successful messages over the communication channel. It is measured in terms of correctly received bits/packets/ messages per unit time. We define the metric as the number of correctly received packets per unit time i.e.,

$$\text{Throughput Metric } = \frac{\text{Number of correctly received packets}}{\text{Transmission time for all packets}} \, packets/sec.$$

The multimedia image file is used for transmission of packets. The throughput metric is compared for the two-way and three-way handshaking of the SUs under no PU traffic, empirical threshold and MAP testing.

Table 5.15: Throughput metric of SUs for rendezvous protocols for three conditions

Handshaking protocol	No PU traffic	Empirical Threshold	MAP testing
Scenario 3 (Two-way)	3.759	2.486	3.011
Scenario 3 (Three-way with DBPSK)	5.498	5.445	5.029
Scenario 3 (Three-way with GMSK)	6.066	5.939	5.933

Remarks

1. From the Table 5.15, the throughput metric for the three-way with DBPSK and GMSK is more than the two-way handshaking. It indicates the three-way handshaking provides the effective way for transmission of packets over the communication channel.

2. From the Table 5.15, the throughput metric for the three way handshaking for DBPSK is less than the three-way handshaking for GMSK. It indicates the GMSK modulation for SUs provides high efficieny (throughput) compared to DBPSK for SUs.

5.4.4.7 Multimedia Traffic

The Figure 5.11 and 5.12 provides the multimedia traffic of primary and secondary users for various threshold. Here, Coded OFDM is used for primary traffic and GMSK modulation is used for secondary traffic. The figure provides the image files for four cases that are discussed in the former tables.

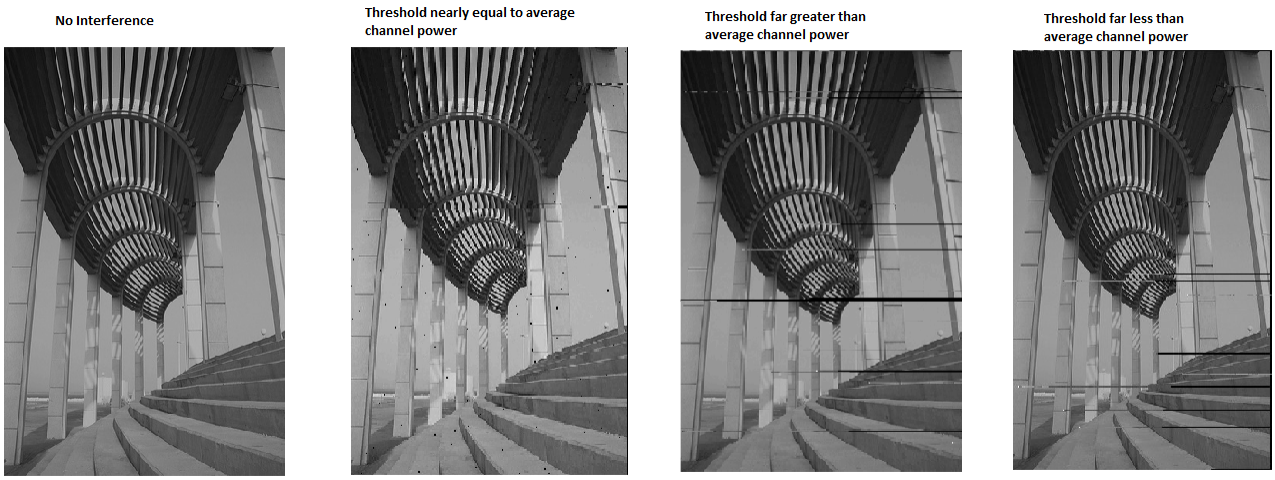

Figure 5.11: Multimedia Traffic for Primary Users

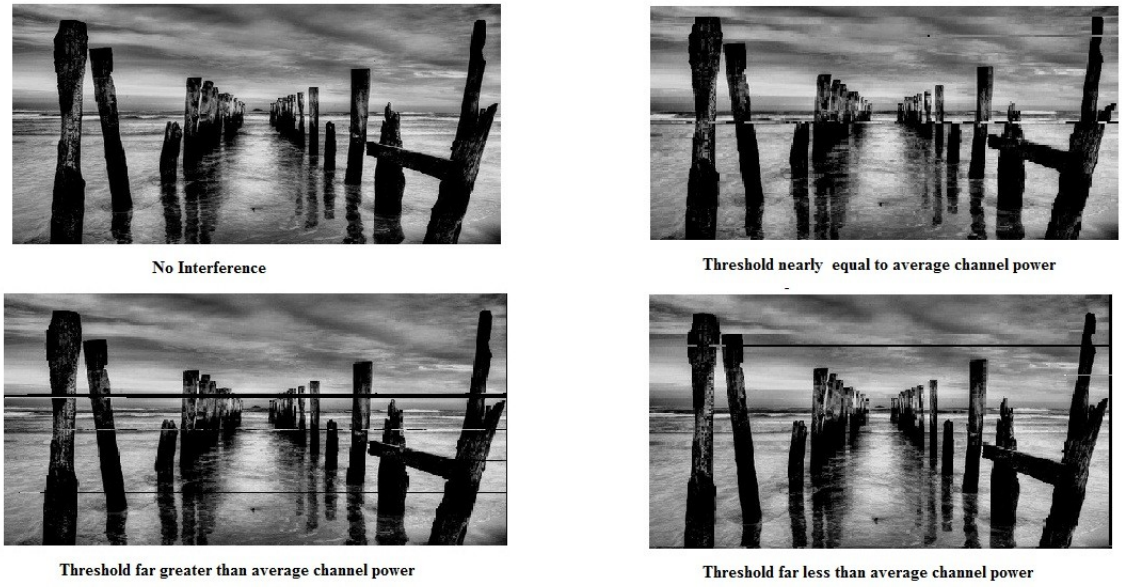

Figure 5.12: Multimedia Traffic for Secondary Users

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The ability of SDR is used to develop the CR test-bed. We proposed our own test-bed model to achieve spectrum sensing and co-existence of the PUs and SUs in an interference channel. We worked on implementing the energy detection of PU using the average periodogram technique to provide the PSD, curve-fitting functions, histogram and density plots for two different hypothesis (busy or idle) in order to determine the statistical variations of primary traffic and the density functions of average channel power. Moreover, the histogram plots for varying time windows and windowing functions are provided to determine the variations in average channel power.

We implemented the Markov traffic model for the primary users and the need for transition time due to practical limitations are addressed. Also, the Coded OFDM transceivers are developed and the problems for the big burst of errors are mitigated by modifying the header structure, transition time between threads, interleaving blocks and reducing the packet size.

Finally, we extended the work to a four-node test-bed model with two PUs and two SUs. The interference metrics of the PUs and SUs are evaluated for the three scenarios with empirical, false alarm and miss detection thresholds in each case. Also, the throughput metrics provide that three-way handshaking for SU with GMSK modulation provide high throughput in our experiments.

6.2 Future Work

There is a great scope for improving the test-bed model of Cognitive radio. The effective area of research work includes

1. We have considered energy detection using average periodogram analysis for spectrum sensing of PUs. In a large network, there are different PUs and SUs, so we need an efficient detector to differentiate between different PUs and SUs. Thus, we could extend the work to implement other detection methods such as cyclo-stationary and matched filter detection that helps differentiate between different users.

2. The optimality for the detection of the threshold is to be determined. Also, to optimize the SU sensing and transmission parameters.

3. The statistical information of the Markov traffic model for the PUs can be utilized for the SUs to efficiently utilize the OFF period and configure their transmission parameters including transmission power, duty cycles and modulation.

4. To Develop and implement more effective and efficient MAC protocols for the SU and perform MAC layer sensing.

Bibliography

[1] Open source gnu-radio. "`http://gnuradio.org/`".

[2] BBN Technologies Corp. *"GNU Radio Architectural Changes"*.

[3] Danijela Cabric, Artem Tkachenko, and Robert W. Brodersen. "Experimental study of spectrum sensing based on energy detection and network cooperation". In *Proceedings of the first international workshop on Technology and policy for accessing spectrum*, TAPAS '06, New York, NY, USA, 2006. ACM.

[4] Matt Ettus. *"USRP Users and Developers Guide"*. Ettus Research.

[5] Spectrum Policy Task Force. "Spectrum policy task force report et docket no. 02-135". Technical report, U. S. Federal Communications Commission, 2002.

[6] Proaksis John G. and Salehi Masoud. *"Digital Communication"*. Mc Graw Hill, 5 th edition edition, 2008.

[7] Firas Abbas Hamza. *"The USRP under 1.5X Magnifying lens"*. GNU Radio community, June 2008.

[8] S. Haykin. "Cognitive Radio: Brain-empowered wireless ommunications". *Selected Areas in Communications, IEEE Journal on*, 23(2):201 – 220, February 2005.

[9] J. Mitola III. *"Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio"*. PhD thesis, Department of Teleinformatics,Royal Institute of Technology (KTH), Stockholm, Sweden, May 2000.

[10] Hyoil Kim and K.G. Shin. "Efficient discovery of spectrum opportunities with MAC-layer sensing in cognitive radio networks". *Mobile Computing, IEEE Transactions on*, 7(5):533 –545, may 2008.

[11] Alexe E. Leu, Mark McHenry, and Brian L. Mark. "Modeling and analysis of interference in listen-before-talk spectrum access schemes". *International Journal of Network Management*, 16(2):131–147, 2006.

[12] Ettus Matt, W. Rondeau Thomas, and McGwier Robert. "OFDM implementation in GNU Radio". Wireless@VT Symposium, 2007.

[13] Alan V. Oppenhiem and Ronald W. Schafer. *"Discrete-Time Signal Processing,"*, chapter 10. Prentice Hall, 1999.

[14] C. Pimentel and V.C. da Rocha. "On the power spectral density of constrained sequences". *Communications, IEEE Transactions on*, 55(3):409 –416, march 2007.

[15] H. Vincent Poor. *"An Introduction to Signal Detection and Estimation"*. Springer-Verlag New York, Inc., 1994.

[16] T.M. Schmidl and D.C. Cox. "Robust frequency and timing synchronization for OFDM". *Communications, IEEE Transactions on*, 45(12):1613 –1621, dec 1997.

[17] P.K. Sharma and A. Basu. "Performance analysis of peak-to-average power ratio reduction techniques for wireless communication using OFDM signals". In *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on*, pages 89 –95, oct. 2010.

[18] M.D. Silvius, A.B. MacKenzie, and C.W. Bostian. "Rendezvous MAC protocols for use in cognitive radio networks". In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1 –7, oct. 2009.

[19] Captain Ryan W. Thomas. *"Cognitive Networks"*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg,Virginia, 2007.

[20] J.J. van de Beek, M. Sandell, and P.O. Borjesson. "ML estimation of time and frequency offset in OFDM systems". *Signal Processing, IEEE Transactions on*, 45(7):1800 –1805, jul 1997.

[21] P. Welch. "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms". *Audio and Electroacoustics, IEEE Transactions on*, 15(2):70 – 73, jun 1967.

[22] Zhi Yan, Zhangchao Ma, Hanwen Cao, Gang Li, and Wenbo Wang. "Spectrum sensing, access and coexistence testbed for cognitive radio using USRP". In *Circuits and Systems for Communications, 2008. ICCSC 2008. 4th IEEE International Conference on*, pages 270 –274, may 2008.

[23] T. Yucek and H. Arslan. "A survey of spectrum sensing algorithms for cognitive radio applications". *Communications Surveys Tutorials, IEEE*, 11(1):116 –130, quarter 2009.

Appendix A

Algorithm for Markov Traffic Model

Initialization
The two states S_0 and S_1 represents OFF and ON periods, while the transition matrix P provides the transition probabilities.

- $S_0 = 0$ and $S_1 = 1$
- $P = \begin{bmatrix} S_{00} & S_{01} \\ S_{10} & S_{11} \end{bmatrix} = \begin{bmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{bmatrix}$

The ON period follow an uniform distribution and the OFF period depend on the former ON period. The number of packets available for ON periods are x, y, z and their corresponding probabilities, OFF periods and default period are related as

- $\Pr(x) = p1;\ \Pr(y) = p2;\ \Pr(z) = p3$
- $T_{OFFx} = 2 * T_{ONx};\ T_{OFFy} = 4 * T_{ONy};\ T_{OFFz} = 5 * T_{ONz}$
- $T_{def} = 2$

Assumptions
Let,

- N is the number of Markov states
- Mk_st is a vector containing the Markov states
- T_{OFF} is the current OFF period that depends on former ON period

Algorithm

Step 1 : Generate $(N-1)$ uniform random variables between [0,1] and store in a vector U_rv

Step 2 : Store the transition probabilities to a matrix P

Step 3 : Obtain the stationary probabilities of S_0 and S_1 using AP=A;
where, A= $[\Pr(S_0 = 0) \quad \Pr(S_1 = 1)]$

Step 4 : For initial state, generate uniform random variable u.

if $u < \Pr(S_0 = 0)$ **then**
  $Mk\_st[0] \leftarrow 0$
else
  $Mk\_st[0] \leftarrow 1$
end if

Step 5 :

for $i = 1 \rightarrow N-1$ **do**
  if $Mk\_st[i-1] = 1$ **then**
    Generate uniform random variable u
    if $u < p1$ **then**
      $No\_pkts \leftarrow x$
    else
      if $u > p1 \quad and \quad u < p1 + p2$ **then**
        $No\_pkts \leftarrow y$
      else
        if $u > p1 + p2 \quad and \quad u < 1$ **then**
          $No\_pkts \leftarrow z$
        end if
      end if
    end if
    Store No_pkts to vector ON_pkts
    if $U\_rv[i] < P[1][0]$ **then**
      $Mk\_st[i] \leftarrow 1$
    else
      $Mk\_st[i] \leftarrow 0$
    end if
  else
    if $ON\_pkts[i-1] = x$ **then**
      $T_{OFF} \leftarrow T_{OFFx}$
    else
      if $ON\_pkts[i-1] = y$ **then**

```
            T_OFF ← T_OFFy
          else
            if ON_pkts[i − 1] = z then
              T_OFF ← T_OFFz
            else
              T_OFF ← T_def
            end if
          end if
        end if
        if U_rv[i] < P[0][1] then
          Mk_st[i] ← 0
        else
          Mk_st[i] ← 1
        end if
      end if
    end for
```

Appendix B

Description of Channel Estimation in OFDM Model

The channel estimation for the OFDM symbols is performed in two steps. Initially, the known preamble symbol is used to correlate and estimate the coarse frequency offset. In the estimation of channel coefficients, the even taps are set based on known preamble and coarse frequency offset, while the odd taps are linearly interpolated between set carriers and zero-filled carriers.

Assumptions

Let,

- L is the cyclic prefix length.
- n is the OFDM symbol count.
- Δf_c is the coarse frequency offset.
- n_c is the maximum bins for frequency correlation.
- $\Delta\phi_{K_s}$ and $\Delta\phi_s$ are used in the correlation from known preamble and received symbol.
- $\hat{h}$ is the channel coefficient of OFDM symbol.

Algorithm

Step 1 : Initialize $\Delta\phi_{K_s} = [0, 0, \cdots, 0]$, $\Delta\phi_s = [0, 0, \cdots, 0]$ and $\hat{h} = [0, 0, \cdots, 0]$; where, $|\Delta\phi_{K_s}| = |\hat{h}| = \tilde{N}$, $|\Delta\phi_s| = N$

Step 2 : Estimation of coarse frequency offset

for $i = 0 \rightarrow |\Delta\phi_{K_s}| - 2; \quad i \leftarrow i + 2$ **do**
$\Delta\phi_{K_s}[i] \leftarrow ||K_s[i] - K_s[i+2]||^2$
end for
for $i = 0 \rightarrow |\Delta\phi_s| - 2; \quad i \leftarrow i + 2$ **do**
$\Delta\phi_s[i] \leftarrow ||Y[i] - Y[i+2]||^2$
end for
for $i = z_l - n_c \rightarrow z_l + n_c$ **do**
$sum \leftarrow 0;\ max \leftarrow 0;\ index \leftarrow 0$
for $j = 0 \rightarrow j < \tilde{N} - 1$ **do**
$sum \leftarrow sum + (\Delta\phi_{K_s}[j] * \Delta\phi_s[i+j])$
end for
if $sum > max$ **then**
$max \leftarrow sum;\ index \leftarrow i$
end if
end for
$\Delta f_c \leftarrow index - z_l;\ n = 0$

Step 3 : Estimation of channel coefficients
$\hat{h}[0] \leftarrow K_s[0] \star e^{-\frac{2\pi\Delta f_c L}{N*1}} \star Y[z_l + \Delta f_c]$
for $i = 0 \rightarrow \tilde{N} - 1; \quad i \leftarrow i + 2$ **do**
$\hat{h}[i] \leftarrow K_s[i] \star e^{-\frac{2\pi\Delta f_c L}{N*1}} \star Y[i + z_l + \Delta f_c]$
$\hat{h}[i-1] \leftarrow \frac{h[i]+h[i-2]}{2}$
end for
for $i = 0 \rightarrow \tilde{N} - 1$ **do**
$\hat{Y}[i] \leftarrow \hat{h}[i] \star e^{-\frac{2\pi\Delta f_c L}{N*n}} \star Y[i + z_l + \Delta f_c]$
end for
$n \leftarrow n + 1$

Appendix C

Description of Demodulation of OFDM Symbols

The demodulation of OFDM symbols are performed using PLL (phase locked loop). Let,

- p_k is the phase locked loop coefficient; $p_k = |p_k|e^{j\phi}$.
- ϵ is the accumulated error
- f is the frequency shift
- ϕ_g is the phase gain
- f_g is the frequency gain
- e_g is the equalizer gain
- $\hat{X_k}$ be the closet constellation point to $\hat{Y_k}$. Also, $\hat{X_k} = C_i$, when euclidean distance metric is minimum i.e., $\min_{1 \leqslant i \leqslant M} \|\hat{Y_k} - C_i\|$, where C_i are from the constellation points of size M.

Algorithm

Step 1 : Initialize, $\phi \leftarrow 0;\ \epsilon \leftarrow 0;\ f \leftarrow 0;\ \phi_g \leftarrow 0.25;\ f_g \leftarrow \phi_g^2/4;\ e_g \leftarrow 0.05$

Step 2 : Initialize, $p = [p_1, p_2, \cdots, p_{\tilde{N}}];\ where,\ p_1 = p_2 = \cdots = p_{\tilde{N}} \leftarrow 1.0 + 0.0j;$

Step 3 :

for $k = 0 \rightarrow \tilde{N} - 1$ **do**
$\hat{Y_k} \leftarrow \hat{Y_k} \times e^{j\phi} \times p_k$
$\epsilon \leftarrow \epsilon + \hat{Y_k} \times \hat{X_k}^*; \epsilon \leftarrow |\epsilon| e^{j\theta}$
if $(\|\hat{Y_k}\| > 0.001)$ **then**
$p_k \leftarrow p_k + e_g \times (\hat{X_k}/\hat{Y_k} - p_k)$
end if
end for

Step 4 : $\theta \leftarrow \arg(\epsilon)$

Step 5 : $f \leftarrow f - f_g \times \theta$

Step 6 : $\phi \leftarrow \phi + f - \phi_g \times \theta$

Step 7 :
if $(\phi \geqslant 2\pi)$ **then**
$\phi \leftarrow \phi - 2\pi$
end if

Step 8 :
if $(\phi < 0)$ **then**
$\phi \leftarrow \phi + 2\pi$
end if

Appendix D

Programs for Primary and Secondary Users

D.1 Primary Users: Markov Traffic Model with Coded OFDM

```
#!/usr/bin/env python
#
# Copyright 2005, 2006 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#
```

```
########################################################################
#usage = "usage: %prog [options] -f cen_freq -m modulation -s packet-size
          --tx-amplitude=value"
#Example:  python benchmark_coded_ofdm_tx.py -f 2.422G -m qpsk -s 508
          --tx-amplitude=8000
########################################################################


from gnuradio import gr, blks2,ofdm_packet_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import random, time, struct, sys, math, os
import gnuradio.gr.gr_threading as _threading

# from current dir
from coded_transmit_path import coded_transmit_path
from pick_bitrate import pick_tx_bitrate
import fusb_options


class my_top_block(gr.top_block):
    def __init__(self, options):
      gr.top_block.__init__(self)

      self._tx_freq            = options.tx_freq    # tranmitter's center frequency
      self._tx_subdev_spec     = options.tx_subdev_spec  # daughterboard to use
      self._interp             = options.interp     # interpolating rate
      self._fusb_block_size    = options.fusb_block_size # usb info for USRP
      self._fusb_nblocks       = options.fusb_nblocks    # usb info for USRP

      if self._tx_freq is None:
        sys.stderr.write("-f FREQ or --freq FREQ must be specified\n")
        raise SystemExit

      # Set up USRP sink; also adjusts interp, and bitrate
      self._setup_usrp_sink()

      self.txpath = coded_transmit_path(options)
      self.rxpath = coded_receive_path(options)
```

```
      self.connect(self.txpath, self.u)


# ////////////////////////////////////////////////////////////////
#  The Remaining methods for declaration of USRP and
# other methods for changing parameters of USRP are similiar to
# OFDM (GNU radio examples)
# ////////////////////////////////////////////////////////////////

# ////////////////////////////////////////////////////////////////////////
#    main ( Markov traffic generation and packet transmission)
# ////////////////////////////////////////////////////////////////////////

def main():

    def send_pkt(payload='', eof=False):
        return tb.txpath.send_pkt(payload, eof)

    ######## Function to generate the random Markov OFF(0)
    ## and ON(1) states based on Transition Matrix P ############
    def gen_markov_states(no_of_sts, trans_matx):
        global Mk_states
        Mk_states = []
        # generates uniform r.v of length equal to markov states
        unif_rv=[ random.random() for i in xrange(no_of_sts-1)]
        ## Generate Initial state based on prior probabilites of ON and OFF
        #  that are obtained by solving AP = A; where  A = [ Pr(S_0 = 0) Pr(S_1 =1]
        y=random.random()
        if y < 0.464:
          MK_states.append(0)
        else:
          Mk_states.append(1)
        ## Generate states based on algorithm discussed in Appendix A
        for i in range(no_of_sts):
          if(Mk_states[i]==1):
             if (unif_rv[i]<trans_matx[1][0]):
                Mk_states.append(0)
             else:
                Mk_states.append(1)
          if(Mk_states[i]==0):
             if (unif_rv[i]<trans_matx[0][1]):
                Mk_states.append(1)
```

```
          else:
             Mk_states.append(0)


#### Function to return the packets for
##  ON traffic that follow uniform distribution
 def gen_ON_pkts(ON_traffic_pkts,ON_traffic_prs):
     uni_rv =random.random()
     if uni_rv < ON_traffic_prs[0] :
        no_of_pkts = ON_traffic_pkts[0]
     if uni_rv > ON_traffic_prs[0] and uni_rv < ON_traffic_prs[0] \
                              + ON_traffic_prs[1]:
        no_of_pkts = ON_traffic_prs[1]
     if uni_rv > ON_traffic_prs[0] + ON_traffic_prs[1] and uni_rv < 1 :
        no_of_pkts = ON_traffic_prs[2]


 ## Function for determining the no of packets of former ON
 ## state to evaluate the OFF time for the current OFF state
 def tym_OFF_pkts(no_of_pkts):
     arr=[]
     arr.append(no_of_pkts)
     tym_oFF=arr[len(arr)-1]
     return tym_OFF


 parser = OptionParser(option_class=eng_option, conflict_handler="resolve")
 expert_grp = parser.add_option_group("Expert")
 parser.add_option("-s", "--size", type="eng_float", default=508,
    help="set packet size [default=\%default]")
 parser.add_option("-M", "--megabytes", type="eng_float", default=1.0,
    help="set megabytes to transmit [default=\%default]")

 parser.add_option("", "--no-of-sts", type="eng_float", default=1000,
    help="set no of states of Markov traffic [default=\%default]")
 parser.add_option("-P", "--transition-matrix", type="eng_float", \
    default = [[0.25, 0.75],[0.35,0.65]],
    help="set transition matrix for Markov traffic")
 parser.add_option("", "--pkts-ON-traffic", type="eng_float", \
    default = [40,60,80],
    help = " set the packets for ON traffic (supports three)")
 parser.add_option("", "--prs-ON-traffic", type="eng_float", \
    default = [0.45,0.3,0.25],
```

```
   help = "set the probabilities for packets (corresponds to ON traffic)")

parser.add_option("","--from-file", default=None,
   help="use file for packet contents")
parser.add_option("", "--fsm-file", default =None,
   help= " use for setting the FSM parameters of channel encoder")


my_top_block.add_options(parser, expert_grp)
coded_transmit_path.add_options(parser, expert_grp)
blks2.ofdm_mod.add_options(parser, expert_grp)
blks2.ofdm_demod.add_options(parser, expert_grp)
fusb_options.add_options(expert_grp)

(options, args) = parser.parse_args ()

if options.from_file is not None:
    source_file = open(options.from_file, 'r')

pkt_size = int(options.size)

## set the Markov states, transition matrix and ON traffic parameters
Mk_states = options.no_of_sts
Mk_tran_matx = options.transition_matrix
ON_traffic_pkts = options.pkts_ON_traffic
ON_traffic_prs =options.prs_ON_traffic


# build the graph
tb = my_top_block(options)

r = gr.enable_realtime_scheduling()
if r != gr.RT_OK:
    print "Warning: failed to enable realtime scheduling"
tb.start()                       # start flow graph

# Start the thread to watch the coded payload that are acquired
# from messge queue so as to pass through OFDM modulator blocks.
txr_watcher = _queue_watcher_thread_txr(tb.txpath.coded_pkt_msgq,\
                     tb.txpath.ofdm_tx._pkt_input)

# Call the Function to generate Markov states
# based on transition matrix and number of states
```

```
gen_markov_states(Mk_states, Mk_tran_matx)

# Initialize the parameters for the loop
count_state=0
current_state= 0
no_of_pkts = 0
pktinfo = 1

while (count_state<Mk_states):
    if Mk_states[current_state]==1:
       print "\n ON state"
       pktno= 1
       no_of_pkts=gen_ON_pkts(ON_traffic_pkts, ON_traffic_prs)
       print " No of packets that are transmitted", no_of_pkts
       while pktno<=no_of_pkts:
          if options.from_file is None:
             data = (pkt_size - 2) * chr(pktno & 0xff)
          else:
             data = source_file.read(pkt_size - 2)
             if data == '':
               break;
          payload = struct.pack('!H', pktinfo & 0xffff) + data
          send_pkt(payload)
          time.sleep(0.01)   # transition time between packets
          txr_watcher.pkt_run()   # Start the other thread
          sys.stderr.write('.')
          pktno+=1
          pktinfo+=1
    else:
       print "\n OFF state"
       ###  Determining the OFF period depending on the former ON period.
       #The value of OFF cycles are selected based on alogirthm in Appendix A
       tymoff=tym_OFF_pkts(no_of_pkts)
       if (tymoff==ON_traffic_pkts[0]):
          tm=5.0
       elif (tymoff==ON_traffic_pkts[1]):
          tm=6.0
       elif (tymoff==ON_traffic_pkts[2]):
          tm=8.0
       else:
          tm=2.0
       print "Time of Sleep for OFF state is \%d secs" \% (tm)
       time.sleep(tm)     ## OFF period  that includes the additional time
```

```
        count_state+=1        ## for switching ON and OFF or OFF and ON states
        current_state+=1
        txr_watcher.pkt_run(eof=True) #Indicate the thread for the finish of packets.
        send_pkt(eof=True)
    tb.wait()                       # wait for it to finish



## Thread to transmit the coded payload stream to OFDM modulator blocks.
class _queue_watcher_thread_txr(_threading.Thread):
    def __init__(self, pkt_msgq,pkt_input):
        _threading.Thread.__init__(self)
        self.setDaemon(1)
        self.pkt_msgq = pkt_msgq
        self.pkt_input = pkt_input
        self.start()

    def pkt_run(self,eof= False):
          if eof:
            coded_msg=gr.message(1)
          else:
            msg = self.pkt_msgq.delete_head()
            time.sleep(0.01)   ### transition time between the threads.
            msg_string=msg.to_string()
            coded_pkt=ofdm_packet_utils.make_packet(msg_string, 1, \
                          1, pad_for_usrp=False, whitening=True)
            coded_msg = gr.message_from_string(coded_pkt)
          self.pkt_input.msgq().insert_tail(coded_msg)



# ////////////////////////////////////////////////////////////////////////////
#                        Coded OFDM transmit flow-graph
# ////////////////////////////////////////////////////////////////////////////

class coded_transmit_path(gr.hier_block2):
    def __init__(self, options):

gr.hier_block2.__init__(self, "coded_transmit_path",
gr.io_signature(0, 0, 0), # Input signature
gr.io_signature(1, 1, gr.sizeof_gr_complex))
```

```
self._verbose      = options.verbose   # turn verbose mode on/off
self._tx_amplitude = options.tx_amplitude  # digital amplitude sent to USRP


#####Channel Coding blocks ########
self.uncoded_pkt_input=gr.message_source(gr.sizeof_char,4)
### The FSM file contains the FSM parameters
fsm_st=trellis.fsm(options.fsm_file)
## determine the input and output  bits
bitspersymbol = int(round(math.log(fsm_st.I())/math.log(2)))
codedsymbolperbits=int(round(math.log(fsm_st.O())/math.log(2)))
K= ((int(options.size)+4)*8)/bitspersymbol
interleaver=trellis.interleaver(K,666)
## pack and unpack to pass the required bits based on channel encoder
self.pack_unpack=gr.packed_to_unpacked_bb(bitspersymbol,gr.GR_MSB_FIRST)
self.trell_enc= trellis.encoder_bb(f,0)
self.inter = trellis.permutation(interleaver.K(),interleaver.INTER(),\
                                 1,gr.sizeof_char)
self.unpack_pack=gr.unpacked_to_packed_bb(codedsymbolperbits,gr.GR_MSB_FIRST)
self.coded_pkt_msgq=gr.msg_queue()
self.coded_msg_sink=gr.message_sink((gr.sizeof_char),self.coded_pkt_msgq,False)

##############################


self.ofdm_tx = \
             blks2.ofdm_mod(options,msgq_limit=4, pad_for_usrp=False)

self.amp = gr.multiply_const_cc(1)
self.set_tx_amplitude(self._tx_amplitude)

# Display some information about the setup
if self._verbose:
    self._print_verbage()

# Create and setup transmit path flow graph
self.connect(self.uncoded_pkt_input,self.pack_unpack,self.trell_enc,self.inter)
self.connect(self.inter,self.unpack_pack,self.coded_msg_sink)
self.connect(self.ofdm_tx, self.amp, self)

# ////////////////////////////////////////////////////////////////
#  The Remaining methods for declaration of OFDM parameters and
# other methods for changing them are similiar to
```

```
        # OFDM (GNU radio examples)
        # ////////////////////////////////////////////////////////////////////
```

```
if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

D.2 Secondary Users: Three-way Handshaking

```
#!/usr/bin/env python
#
# Copyright 2005,2006,2007,2009 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.

from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
```

```
from optparse import OptionParser
import random,struct,sys,math,time

# Import from current directory
import usrp_dynamic_str_path


############# Three-way handshaking protocol ##########################
# Parameters and functional spec
#Two modes : Synchronous and Data traffic ; default = sync_traffic
#  sync_traffic : Exhange of BCN, ACK, ACK-ACK signal for mutual handshake
#  data_traffic : Actual data traffic b/w SUs
#  sync_str provide the representation for BCN('b'),ACK('a') and ACK-ACK('o') signal
#  node_st provides the SUs node transmitter ('t') and receiver ('r')

#usage = "usage: \%prog [options] -f cen_freq -m modulation  -r data rate"
#Example:  python benchmark_coded_ofdm_tx.py -f 2.422G -m gmsk -r 250k
##############################################################################
# ////////////////////////////////////////////////////////
#                       Top block
# ////////////////////////////////////////////////////////

class my_top_block(gr.top_block):
    def __init__(self, demodulator, modulator,rx_callback,options):
      gr.top_block.__init__(self)

      # Set up the dynamic flow graph (receive,transmit or sense path)
      self.dynamic_str_path = usrp_dynamic_str_path.usrp_dynamic_str_path(demodulator,\
                                modulator, rx_callback, options)
      self.connect(self.dynamic_str_path)

# Global Variables
global n_rcvd,n_right,mode, data_pktno
mode = "sync_traffic"

# ////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////

def main():
    global n_rcvd,n_right,frame_power,mode,channel
    global traffic_status
    n_rcvd = 0
```

```
n_right = 0
frame_power=0


###### Receiver Info ##############
def rx_callback(ok, payload):  # Callback method for reception of packets ##
 global n_rcvd, n_right,mode
 global channel,traffic_status

 if (mode == "sync_traffic") :
   if ok:
      (pktno,) = struct.unpack('!H', payload[0:2])
      (sync,)  = struct.unpack('s',  payload[2])
      (state,) = struct.unpack ('s',  payload[3])
      (ch_no,) = struct.unpack('!H',  payload[4:6])
      if (str(sync) == 'a' and str(ch_no) == str(channel)):
         if (str(state) == 'r'):
           traffic_status = "txr"
         mode = "data_traffic"
         sync_str = 'o'
         node_st = 't'
         ack_ack_payload = struct.pack('!HssH', pktno & 0xffff,\
                             sync_str,node_st,channel & 0xffff)
         send_pkt(ack_ack_payload)
         print "Received Ack Signal,Transmit Acknowlegment for Ack Signal"
         print " ok = %5s pktno = %4d" %(ok,pktno)

      if (str(sync) == 'b' and str(ch_no) == str(channel)):
         sync_str = 'a'
         node_st = 't'
         ack_payload = struct.pack('!HssH', pktno & 0xffff,\
                         sync_str,node_st,channel & 0xffff)
         send_pkt(ack_payload)
         print " Recevied Beacon Signal,Transmit Ack Signal"
         print "ok = %5s pktno = %4d" %(ok,pktno)


      if (str(sync) == 'o' and str(ch_no) == str(channel)):
         mode = "data_traffic"
         if (str(state) == 'r'):
           traffic_status = "txr"
         print "Received Acknowledgment for Ack Signal"
         print "ok =%5s pktno =%4d"  %(ok,pktno)
```

```
 if mode == "data_traffic":
   if (len(str(payload[0:2])) == 2):
      (data_pktno,) = struct.unpack('!H', payload[0:2])
      (hdr, ) = struct.unpack('!s' , payload[2])
      if(str(hdr) == 'h' ) :
          n_rcvd += 1
          if ok:
            n_right += 1
          print "ok = %5s  data_pktno = %4d n_rcvd = %4d n_right = %4d" \
                  % (ok, data_pktno,n_rcvd,n_right)



######  Transmitter Info ###############
def send_pkt(payload='', eof=False):
  return tb.dynamic_str_path.send_pkt(payload, eof)

def tx_packets():
  nbytes = 15
  pkt_size = options.size
  n = 0
  while n < nbytes:
     hdr = 'h'
     if options.from_file is None:
       data=(pkt_size - 3) * chr(data_pktno & 0xff)
     else:
       data=source_file.read(pkt_size-2)
     if data == '':
       break
     payload = struct.pack('!Hs', data_pktno & 0xffff,hdr) + data
     send_pkt(payload)
     time.sleep(0.02)
     n += 1
     sys.stderr.write('.')
     data_pktno += 1


######  Sensing Info ###############

def set_sensing(freqs):
```

```
    global frame_power
    tb.dynamic_str_path.setup_scan_info(freqs[0],freqs[1])
     # no of center frequencies to be scanned
    nsteps = tb.dynamic_str_path.str_path.nsteps
    ## scan_mode is method implemented in gr.bin_statistics.cc to
    # start the scanning for primary channel
    tb.dynamic_str_path.stats.scan_mode(int(nsteps))
    tb.dynamic_str_path.sensing()
    frame_power = tb.dynamic_str_path.str_path.avg_frame_power


######  Channel Info ##############

def chan_info(ch_no):
  dict_cen={}
  dict_minmax={}

  cen_freq=["2.422G","2.452G", "2.457G","2.462G"]
  channel_no=[3,9,10,11]

  min_freq=["2.421G","2.451G","2.456G","2.461G"]
  max_freq=["2.423G","2.453G","2.458G","2.463G"]

  for i in range(len(channel_no)):
    dict_minmax[channel_no[i]]=[min_freq[i],max_freq[i]]

  ky=dict_minmax.keys()
  val=dict_minmax.values()

  for i in range(len(ky)):
    if (ky[i]==ch_no):
      freqs=val[i]

  for i in range(len(channel_no)):
    dict_cen[channel_no[i]]=cen_freq[i]

  ky1=dict_cen.keys()
  val1=dict_cen.values()

  for i in range(len(ky1)):
    if (ky1[i]==ch_no):
      cen_freq=val1[i]
  cen_freq=eng_notation.str_to_num(cen_freq)
```

```
  freqs[0]=eng_notation.str_to_num(freqs[0])
  freqs[1]=eng_notation.str_to_num(freqs[1])
  return freqs,cen_freq

############
##### Command Line arguments for transmission
## reception and sensing Flowgraphs ######
demods = modulation_utils.type_1_demods()
mods   = modulation_utils.type_1_mods()

# Create Options Parser:
parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
expert_grp = parser.add_option_group("Expert")
parser.add_option("-m", "--modulation", type="choice", choices=mods.keys(),
  default='dbpsk',
  help="Select modulation from: %s [default=%%default]"
  % (', '.join(mods.keys()),))
parser.add_option("-s", "--size", type="eng_float", default=1500,
  help="set packet size [default=%default]")
parser.add_option("-M", "--megabytes", type="eng_float", default=1.0,
  help="set megabytes to transmit [default=%default]")
parser.add_option("","--from-file", default=None,
  help="use file for packet contents")


usrp_dynamic_str_path.add_options(parser, expert_grp)

for mod in mods.values():
    mod.add_options(expert_grp)

for mod in demods.values():
    mod.add_options(expert_grp)

(options, args) = parser.parse_args()


# build the graph
tb = my_top_block(demods[options.modulation],\
            mods[options.modulation], rx_callback, options)

if options.from_file is not None:
    source_file = open(options.from_file, 'r')
```

```
 r = gr.enable_realtime_scheduling()
 if r != gr.RT_OK:
     print "Warning: Failed to enable realtime scheduling."

 tb.start()        # start flow graph
 running = True
 pktno=0
 data_pktno = 0
 print "Sensing Mode(Sense the primary channel)"

 while running:
     if (mode == "sync_traffic"):
        # random selection of channels
channel=int(random.choice([10,11]))
        # Dynamic change of flow-graph to connect only sensing flow-graph
        tb.dynamic_str_path.configure_rxr_sense()
        scan_freqs,cen_freq=chan_info(channel)
        set_sensing(scan_freqs)
        #Dynamic change of flow-graph to disconnect sensing flow-graph
        # and connect transmission and reception flow-graphs
        tb.dynamic_str_path.configure_sense_rxr()

        ########Decision Statistics based on MAP testing #########
        # stats follow gaussian distribution (each mean and variance are known)
        dec_stats_OFF = (1/(math.sqrt(2*math.pi*1.53))) * \
                         (math.exp(-((frame_power - 27.168)**2)/(2*1.53)))
        dec_stats_ON  = (1/(math.sqrt(2*math.pi*2.89))) * \
                         (math.exp(-((frame_power - 44.040)**2)/(2*2.89)))

        # The ON and OFF probability ratio is obtained from Appendix A
        dec_stats = (dec_stats_OFF/dec_stats_ON)
        mk_ON_OFF_ratio=(0.35/0.75)

        if ( dec_stats > mk_ON_OFF_ratio):
           print "Primary Channel %d is Busy" %(channel)
           continue
        ######
        # set the center frequency for tx'r and rx'r
        tb.dynamic_str_path.usrp_source.set_center_freq(cen_freq)
        tb.dynamic_str_path.usrp_sink.set_center_freq(cen_freq)

        nbytes = 6
        pkt_size = 6
```

```
            n = 0
            while n < nbytes:
              sync_str = 'b'
              node_st = 't'
              payload = struct.pack('!HssH', pktno & 0xffff,sync_str,\
              node_st, channel & 0xffff)
              send_pkt(payload)
              n += len(payload)
              sys.stderr.write('.')
              pktno += 1
              time.sleep(0.1)
          ######################
         if (mode == "data_traffic" ):
            if (traffic_status == "txr"):
                 tx_packet()
                 time.sleep(0.4) # transition time to switch to sensing cycle
            if (traffic_status == "rxr") :
                 time.sleep(2.0)
            mode ="sync_traffic"
            time.sleep(0.1)  # transition time to switch to sensing cycle



class dynamic_str_path(gr.hier_block2):
    def __init__(self, demod_class, mod_class,rx_callback,\
                       options,usrp_rate,usrp_source,usrp_sink):
gr.hier_block2.__init__(self, "dynamic_str_path",
gr.io_signature(0, 0, 0),
gr.io_signature(0, 0, 0))


         ######################################################################
         # Receiver and Transmitter flow-graphs : Packet modulator blocks
         #  Sensing Flow-graph : usrp_spectrum_sense.py

         # Connect the  initial Sensing flowgraph
         self.connect(self.packet_transmitter, self.amp, self.usrp_sink)
         self.connect(self.usrp_source,self.ampl)
         self.connect(self.ampl,self.channel_filter,self.packet_receiver)
         self.connect(self.ampl,self.s2v,self.fft,self.c2mag,self.stats)

         ################################################
```

```
# Sensing for channel in piece-wise average periodogram analysis
def sensing(self):
    frame_count=0
    self.frame_power=0
    mid_stats = zeros(self.nsteps*self.fft_size)
    psd_stats = zeros ((self.nsteps * self.fft_size) - ((self.nsteps-1)*64))
    count=0

    while (frame_count!=1):
      m = parse_msg(self.msgq.delete_head())
      fshift=array(m.data)
      f_shift=fftshift(fshift)
      freq_stats=zeros(len(m.data))

      for i in xrange(len(m.data)):
          freq_stats[i]= 10*math.log10(f_shift[i]/(self.norm_win_pow*len(m.data)))
          mid_stats[count+i] = freq_stats[i]



      if (m.center_freq == self.max_center_freq):
          # Evaluating the periodogram stats for all centre
           #frequencies (including averaging the overlapping bins)
          for i in range(0,64):
             psd_stats[i] = mid_stats[i]

          for i in range(0,self.nsteps):
             for j in range( (i*self.fft_size+64),((i+1)*self.fft_size - 64)):
                psd_stats[j-64*i] = mid_stats[j]

          for i in range((self.nsteps*self.fft_size)-64,(self.nsteps*self.fft_size)):
             psd_stats [i -(64*(self.nsteps-1))] = mid_stats[i]

          for i in range(1,self.nsteps): #  Averaging the overlapping bins
             for j in range(0,64):
                psd_stats[i*self.fft_size-64*i + j] = \
               (mid_stats[i*self.fft_size-64+j]+ mid_stats[i*self.fft_size + j])/2

          self.avg_frame_power = (psd_stats.sum()/(self.nsteps*self.fft_size))
          #print "The average power of frame" ,self.avg_frame_power
          frame_count += 1
```

```
            else:
                count = count + self.fft_size


if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

Vita

Venkat Vinod Patcha was born in July, 1987, in Kurnool, Andhra Pradesh, India. He graduated with his Bachelor of Technology in Electronics and Communication Engineering degree from Padmasri Dr. B.V.Raju Institute of Technology (Affiliated to JNTU), Hyderabad, India, in the year 2008. He is presently pursing his Master of Science in Electrical Engineering degree at Louisiana State University and is expected to graduate in August 2011. His research interests include digital/wireless communications, cognitive radio and software radios and his present focus is on design and implementation of cognitive radio test-bed using software radios.

He was awarded the Certificate of Academic Excellence in his bachelor degree studies for the academic year 2004-2007 from former Indian President A.P.J Abdul Kalam. He worked as Research Assistant in the Department of Electrical and Computer Engineering, Louisiana State University. He is a graduate student member of IEEE.