An Investigation into Cognitive Radio System Performance

Courtney Hardy

Lucas F. Feather
University of Mississippi Main Campus

William S. Guy
University of Mississippi Main Campus

AN INVESTIGATION INTO COGNITIVE RADIO SYSTEM PERFORMANCE

by
Lucas Feather, William Guy, and Courtney Hardy

A thesis submitted to the faculty of The University of Mississippi in partial fulfillment of the requirements of the Sally McDonnell Barksdale Honors College

Oxford
May 2021

Approved by

Advisor: Professor John N. Daigle

Reader: Professor Paul M. Goggans

Reader: Professor Md Sakib Hasan

ACKNOWLEDGEMENTS

First, we would like to thank Dr. John N. Daigle for his advisory role in this project. For the duration of our research, Dr. Daigle graciously provided us with the materials and advice we needed, using his own personal time to make sure that we could always move forward with our project. We are grateful for his dedication to us and his interest in our continued education. As he had in our classes with him, he constantly challenged us, motivated us, encouraged us, and pushed us to accomplish our goals and to see this project through to the end. During our time working with him, he has provided our group with invaluable insight into engineering and life.

I would like to thank my parents, Dr. Timothy and Jacqueline Feather. Their support and encouragement has inspired me to pursue an engineering degree, and their lessons of patience and humility have helped me to stick with it. I would also like to thank Jack Reynolds Holiman, for sharing with me his relentless creativity, love, and innocent curiosity in the time that I was lucky enough to know him.

Lucas Feather

Firstly, I would like to acknowledge and thank my incredible parents, David and Lori Guy. I would not be who I am today without their constant love and support. Throughout my life, they have always encouraged me, always been there to celebrate and cry with, and always inspire me to be the best person I can be. I would also like to acknowledge and thank my Savior, Jesus Christ, for his love for me and for his faithfulness.

William Guy

I would like to thank my parents and my best friend for their support and unconditional love. Their presence in my life has shaped me into the person I am today.

Courtney Hardy

ABSTRACT

The objective of this thesis is to explore cognitive radio performance through an in-depth literature review and an implementation of a software-defined radio prototyping system. Specifically, this thesis investigates the spectrum-sensing aspect of cognitive radio by comparing two spectrum-sensing methods. It was found in the literature review that a system utilizing matched filter detection would provide higher probability of detection in low signal-to-noise ratio environments when compared to a system utilizing energy detection. These spectrum sensing methods were thus implemented and compared in the cognitive radio systems presented in this thesis. Additionally, experiments were conducted to determine the most efficient intervals for the spectrum sensing and cycle interval periods. Therefore, system performance was measured on the basis of probability of successful primary user signal detection and maximum throughput capabilities, quantified by bit error rate. It was found that a cognitive radio system based on matched filter detection was more robust, given that the transmitted signal of interest was previously known. However, compared to a system based on energy detection, the implementation of the matched filter required more complex algorithms and computational power. These results are consistent with the findings in the literature review.

Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

FCC	Federal Communications Commission
SDR	Software Defined Radio
CR	Cognitive Radio
ED	Energy Detection
MFD	Matched Filter Detection
CRN	Cognitive Radio Network
PU	Primary User
SU	Secondary User
RAT	Radio Access Technology
RAN	Radio Access Network
QoS	Quality of Service
TVWS	TV White Space
SAS	Spectrum Access System
MNO	Mobile Network Operator
GRC	GNU Radio Companion
USRP	Universal Software Radio Peripheral
UHD	USRP Hardware Driver
GR-MAC	GNU Radio MAC
DFT	Discrete Fourier Transform
IDFT	Inverse Discrete Fourier Transform
BER	Bit Error Rate

Chapter 1

Introduction

This thesis aims to evaluate cognitive radio (CR) system performance through the lens of spectrum sensing reliability. Two cognitive radio implementations are presented in this work, one of which employs blind spectrum sensing while the other uses signal-specific sensing. The blind sensing method and signal-specific sensing method used in this study are energy detection and matched filter detection, respectively. These sensing methods are responsible for detecting primary user transmission in specific frequency bands. The CR implementations are given in the form of GNU Radio flowgraphs and Python scripts. System performance is measured based on several criteria, including spectrum sensing reliability and communication throughput, quantified by bit error rate. A selection of experiments measuring the signal-to-noise ratio (SNR) sensitivity and throughput capabilities of each implementation were conducted with the goal of determining which sensing paradigm produces a more robust system. The following section provides some high level background information on cognitive radio as a preface to this thesis. Section 1.2 gives a summary of the remaining content in this thesis.

1.1 Background

The vast majority of the radio spectrum between 8.3 kHz and 275 GHz is divided into bands that are reserved for specific activities such as terrestrial broadcasting, satellite communication, space research, radionavigation, and mobile communication. The Federal Communications Commission (FCC) makes these fixed frequency band assignments in order to minimize interference between operating entities. However, studies show that between 15 and 85% of the licensed spectrum consists of vacant portions, also commonly referred to as 'white spaces' or spectrum holes [1]. The frequent presence of these temporarily unused portions of licensed bands suggest that the spectrum is underutilized at times [2].

Given that additional RF spectrum cannot be created, the only other means for improving spectral efficiency is through access to temporarily unoccupied portions of licensed bands. Cognitive radio was devised with this heightened spectral usage capability in mind. According to the FCC, "CR can be viewed as a combined application of software defined radio and intelligent signal processing with functional elements of radio flexibility, spectral awareness and the intelligence of decision-making" [3]. With these intelligent features, CRs are designed to better utilize the spectrum while avoiding interference with other users. In addition to addressing the spectral scarcity dilemma, CR has also emerged as an interesting solution to interoperability issues that have historically challenged emergency responders [3] [4]. Thus, CR technology has the potential to revolutionize the world of communications.

While the potential applications of CR have been explored extensively in the literature, CR technology is still in the early stages of widespread deployment. This trend is largely attributed to the fact that the performance of cognitive radio networks in practical settings is not well known and is difficult to predict based on theoretical models alone [5]. As a result, the amount of throughput a cognitive radio network can realistically achieve and the ways in which the CR may impact the primary network require further investigation. The future of CR deployment will very likely depend on a clearer understanding of these performance issues. It is for this reason that this thesis focuses on CR performance.

1.2 Summary of Thesis Content

The remainder of this thesis is organized as follows. Chapter 2 provides a comprehensive literature review of cognitive radio, with a focus on the spectrum management process, prominent commercial use cases, and ongoing research challenges. Next, in chapter 3, our implementations of two cognitive radio systems are presented and explained. We will discuss equipment used, flowgraphs developed, and physical setups that will be relevant to our experiments. Background theory of energy detection and matched filter detection are also explained in the context of GNU Radio flowgraphs. Chapter 4 breaks down our experiments and each trial conducted, as well as the results of each experiment. In these experiments, the spectrum sensing methods are compared, and other aspects of the implementations are explored. Finally, chapter 5 presents conclusions drawn from the results of the experiments.

Chapter 2

Cognitive Radio Overview

The purpose of this chapter is to present the fundamentals of cognitive radio as described in the literature in order to contextualize the experimental implementation discussed in the subsequent chapters. Section 2.1 introduces software defined radio, which is one of the central components of cognitive radio. Section 2.2 formally defines cognitive radio and provides a high-level description of its theoretical capabilities. Section 2.3 outlines various CR network paradigms identified in the literature. Section 2.4 elaborates on cognitive radio network functionality within the context of spectrum management. With the various features of CR having been introduced from a theoretical perspective, section 2.5 then shifts the focus of the discussion to practical CR use cases as well as implementation challenges that hinder CR deployment. This chapter concludes with section 2.6, in which the key takeaways from the previous sections are summarized.

2.1 Software Defined Radio

Before formally defining cognitive radio, it is worth briefly discussing cognitive radio's key enabling technology, the software defined radio. Software defined radio (SDR) is a radio platform in which radio components such as filters, amplifiers, and modulators are implemented in software rather than hardware. This setup gives the SDR several key advantages over traditional radio. Whereas traditional radios can only be upgraded via hardware replacement, SDRs are highly reconfigurable and their software can be updated fairly easily. Furthermore, SDRs can perform a wide range of functions through relatively simple modification of software modules whereas hardware radios are application specific. To that end, SDRs are often said to be much more ‘future-proof’ given their ability to accommodate a diverse range of protocols, standards, frequency ranges, access technologies, and modulation formats within a single device [6], [7]. Thus, the SDR is an inherently flexible radio platform that has become an increasingly attractive option for many communications applications [8].

2.2 Cognitive Radio Definition

In essence, cognitive radio is a specific type of SDR enhanced by environmental awareness and intelligent adapting capabilities. The key features of CR most frequently cited in the literature include learning capabilities and dynamic reconfiguration of operating parameters [9]. The CR learns about the RF environment, the geographical environment, the SDR’s hardware resources and operating parameters, and federal policies. The CR obtains this knowledge through spectrum sensing and database access [1]. Based on the knowledge obtained, the CR autonomously reconfigures its internal operating parameters (output power, modulation type, center frequency, protocols) in accordance with predefined objectives such as efficient spectrum utilization. The reconfigurable SDR that the CR is built on allows for these internal modifications, which makes SDR an ideal platform for CR. Furthermore, unlike many other modern radios, cognitive radios learn from past actions in order to improve overall performance [9]. Thus, the integration of cognitive capabilities with SDR's intrinsic versatility gives cognitive radio diverse capabilities.

2.3 CR Network Architecture

Multiple CR nodes come together to form a cognitive radio network (also known as the dynamic spectrum access network). In practice, modern wireless network environments are heterogeneous; multiple radio access technologies (RATs) might be employed across multiple spectrum bands within a single network [10]. Such is the case for the network environment shown in Figure 2.1 [10], which illustrates how centralized and decentralized cognitive radio networks (CRNs) might operate alongside primary networks.

Primary networks are composed of primary users (PUs), who have licensed access to certain frequency bands, and primary base stations which facilitate communication between PUs. The CR users, or secondary users (SUs), on the other hand, must operate within unlicensed bands or access licensed spectrum opportunistically.

Interaction amongst secondary nodes varies depending on the network architecture. CR networks can be decentralized, consisting only of distributed CR nodes, or they can be infrastructure-based (centralized), in which case they might contain CR base stations and spectrum brokers in addition to the CR users themselves [7]. CR base stations provide single-hop connections from one CR user to another. They also serve as a central entity that coordinates spectrum sensing operations at multiple CR nodes and makes spectrum availability decisions for the network based on those local observations. Spectrum brokers, on

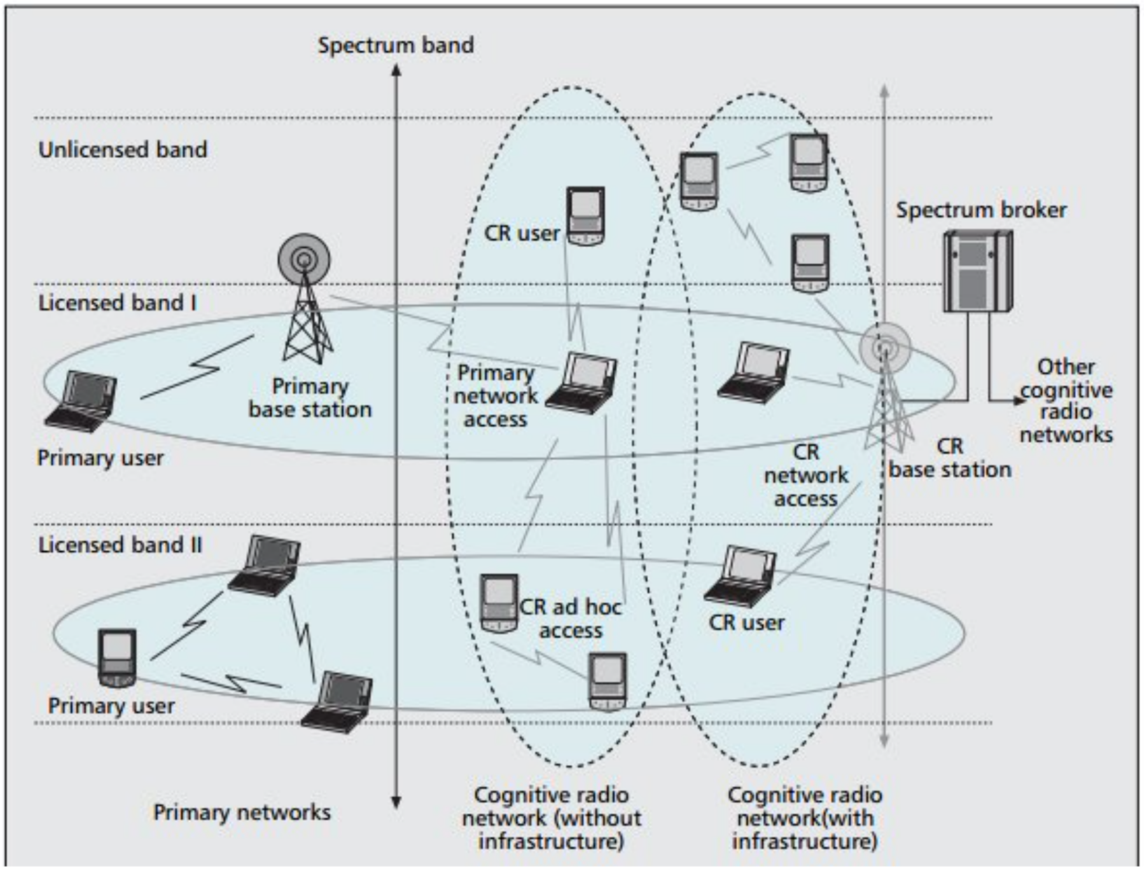


Figure 2.1: Cognitive Radio Network Architecture

the other hand, oversee the distribution of spectrum resources across multiple CR networks based on the sensing observations within each network [11]. In the case of decentralized CRNs, CR users communicate amongst themselves in an ad-hoc fashion, often over mixed frequency bands. Spectrum sensing in these CRNs is synchronized in a distributed way rather than through central infrastructure.

2.4 Spectrum Management Process

In order to enable opportunistic access to unused portions of the licensed spectrum while ensuring that CR users do not interfere with primary users, CRNs must make use of an effective

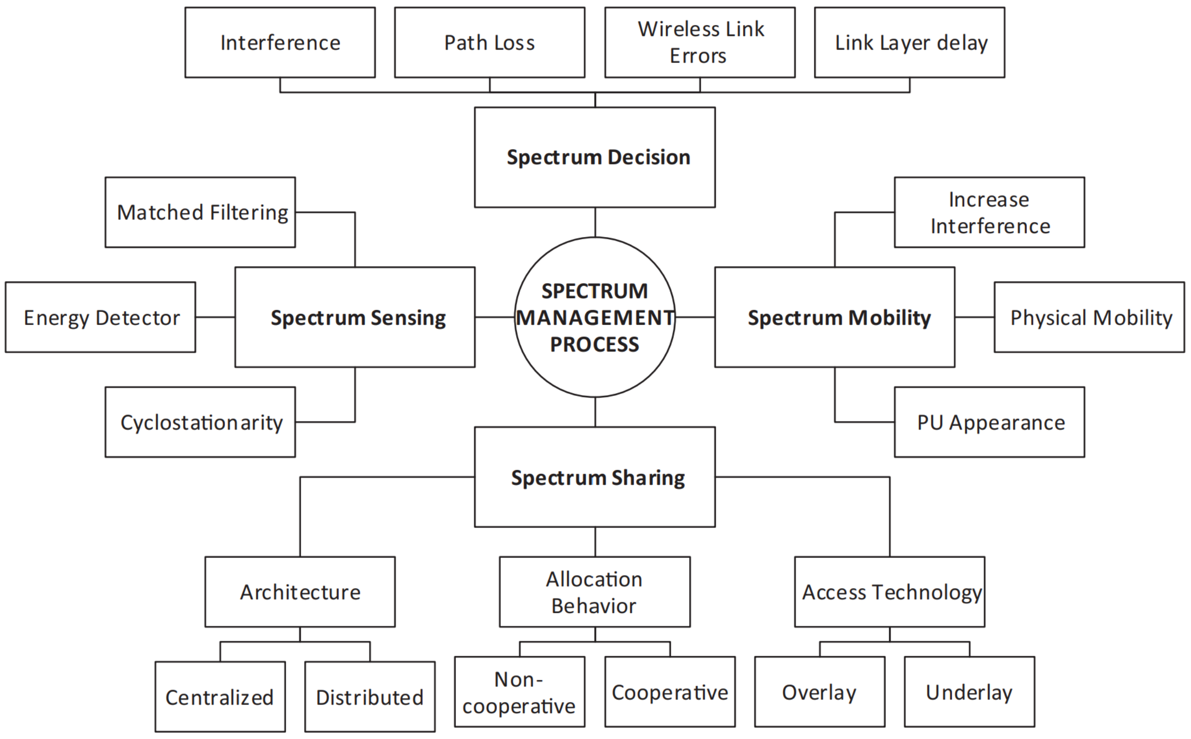


Figure 2.2: Spectrum management process

spectrum management scheme that coordinates activity amongst multiple CR nodes. This spectrum management process is categorized by four key steps: spectrum sensing, spectrum decision, spectrum mobility, and spectrum sharing. Figure 2.2 outlines the various factors that influence the outcome of each step. Each of these steps and their defining factors listed in Figure 2.2 [12] are further discussed in the following subsection.

2.4.1 Spectrum Sensing

Spectrum sensing refers to the monitoring of RF activity within a target frequency band in order to detect unused portions of the licensed spectrum, or 'white spaces', which a secondary user may be able to use. CR users must continually sense the spectrum as PUs could resume transmission over a given spectrum hole at any moment.

Several notable choices of spectrum sensing are identified in the literature. These methods are categorized in Figure 2.3 [13]. The spectrum sensing methods can be divided into two main categories: narrowband sensing and wideband sensing. Narrowband sensing refers to signal detection over a relatively small frequency interval. The narrowband sensing techniques can be further categorized into blind sensing and signal-specific sensing [14]. Blind sensing methods do not rely on any prior knowledge of the PU signal features, whereas signal-specific sensing methods compare an incoming signal with a known signal template which may consist of pilot symbols, preambles, and other statistical features of the target signal. Three of the most frequently used narrowband sensing techniques are energy detection, matched filter (or pilot) detection, and cyclostationary feature detection, as listed in Figure 2.2. Energy detection is an example of a blind sensing method, while matched filter detection and cyclostationary detection are signal-specific methods. Wideband sensing, on the other hand, deals with sensing over a much larger frequency band. With wideband sensing, the large band might be sensed all at once, or it may be divided into individual subchannels that are sensed sequentially or simultaneously using one of the narrowband techniques. Nyquist-based wideband sensing samples signals at the Nyquist rate, whereas compressive wideband sensing samples signals below the Nyquist rate in order to achieve lower power consumption. In general, wideband sensing is a more effective method of detecting spectral holes across a wide frequency range, which enables CR nodes to choose from a greater selection of available channels. However, imperfect knowledge of PU sparsity levels, high-performance hardware requirements, long processing time, and high computational complexity often complicate the use of wideband sensing [13].

The objective of spectrum sensing is always to determine whether a given frequency

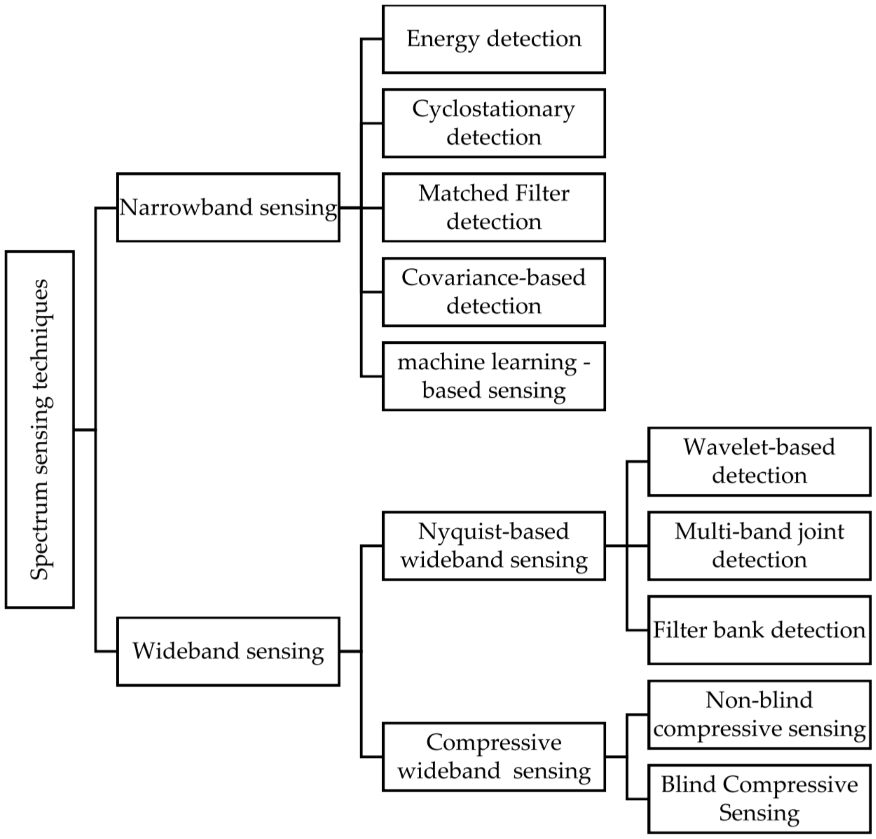

Spectrum sensing techniques
Narrowband sensing
Energy detection
Cyclostationary detection
Matched Filter detection
Covariance-based detection
machine learning - based sensing
Wideband sensing
Nyquist-based wideband sensing
Wavelet-based detection
Multi-band joint detection
Filter bank detection
Compressive wideband sensing
Non-blind compressive sensing
Blind Compressive Sensing

Figure 2.3: Spectrum sensing methods

channel is occupied by a primary user or not. This decision-making process is often modeled by a binary hypothesis test comprised of H_0 and H_1, in which H_0 indicates that the channel is empty and H_1 indicates that the PU is present;

$$H_0 : y(t) = \eta(t). \tag{2.1}$$

$$H_1 : y(t) = h(t)s(t) + \eta(t). \tag{2.2}$$

where $y(t)$ refers to the CR's received signal, $\eta(t)$ denotes an additive white Gaussian noise channel, $s(t)$ is the PU's transmitted signal, and $h(t)$ represents the amplitude variation due to channel fading [15]. N samples of y(t) are taken and used to select either H_0 or H_1 as the outcome of the hypothesis test. The way in which these samples are manipulated depends on the particular sensing method. The reliability of narrowband sensing techniques are frequently assessed using probability of detection and probability of false alarm. P_d refers to the probability that a primary signal was correctly detected, given that it was present. The probability of false alarm, P_f, is the probability of incorrectly detecting a primary user over a given frequency band while there is actually none in that band. These probabilities are defined as follows [16]:

$$P_d = \Pr(\text{Decide } H_1 | H_1). \tag{2.3}$$

$$P_f = \Pr(\text{Decide } H_1 | H_0). \tag{2.4}$$

The probability of missed detection, which is the complement of P_d, given by $P_{\text{md}} = 1 - P_d$, is also frequently used for evaluating the reliability of a given sensing method. A reliable sensing scheme should have a high probability of detection and low probability of false alarm in order to avoid interference with the primary user while still taking advantage of any empty channels available. An ideal sensing method would necessarily have $P_d = 1$ and $P_f = 0$. However, a sensing method with such high fidelity does not exist in practice, and those that approximate it often come at the price of very high computational complexity.

Many of these direct measurement techniques, which are known collectively as in-band sensing, have other major disadvantages, including the hidden terminal issue [1]. The hidden terminal issue refers to the situation wherein a CR node may be able to detect a primary transmitter, but not necessarily a primary receiver, and may therefore still interfere with the incumbent network despite the fact that no primary signal was detected. This issue may also arise when a CR node is unable to detect the PU due to shadowing and fading effects over the wireless channel. Several alternatives to traditional in-band sensing have been proposed as potential remedies to these issues, including out-of-band sensing, which typically involves a dedicated control channel for PU beacon signals, and cooperative sensing, which takes advantage of the spatial diversity of CR nodes by feeding local observations to a common decision-making receiver [1], [17].

2.4.2 Spectrum Decision

Spectrum decision involves accessing the most appropriate spectrum band from a set of multiple white spaces that have been identified. The selection depends on many different criteria, including potential interference with PUs and other CR users, channel holding time, transmit power, channel data rate, signal attenuation at certain high frequencies, and other QoS requirements [1]. Furthermore, spectrum decision-making can either be centralized, distributed, or cluster based, depending on the network architecture. Centralized spectrum decision-making refers to central controllers that make decisions on behalf of the network. Meanwhile, with distributed decision-making, the individual CR nodes make their own spectrum access decisions independent of any centralized network entity. Finally, cluster based decision-making involves a single CR user that makes decisions on behalf of other CR nodes within a local cluster.

2.4.3 Spectrum Mobility

Spectrum mobility, or spectrum handoff, occurs when a CR user must vacate its current channel and resume its transmission elsewhere. Several conditions may render a channel unsuitable, including reappearance of the PU, excess interference, or change in data demand [1]. Upon switching channels, CR nodes must adjust their internal parameters to the protocol suite required in the new operating frequency band, in order to ensure seamless communication throughout the transition [10].

2.4.4 Spectrum Sharing

Spectrum sharing coordinates CR user activity by allocating portions of the spectrum fairly amongst multiple CR devices. This is done in order to prevent users from interfering with each other in overlapping portions of the spectrum. The different methods of spectrum sharing are classified based on network architecture, spectrum allocation techniques, and spectrum access paradigms, as outlined in Figure 2.2. With centralized spectrum sharing, the network's central base station performs the spectrum allocations, whereas with distributed spectrum sharing, the individual CR nodes (typically operating in a decentralized network) access the spectrum according to local policy [12]. Spectrum allocation can be cooperative, wherein CR nodes use clustering techniques to send their interference measurements to a centralized entity to be analyzed. Non-cooperative spectrum allocation, on the other hand, does not require this information exchange between neighboring nodes as each CR user operates independently [18].

Finally, the three primary paradigms for spectrum access are interweave (not shown in Figure 2.2), underlay, and overlay. Interweave access refers to the classic technique of completely avoiding interference with PUs by only transmitting over unoccupied bands and ceasing transmission when PUs reappear. Underlay spectrum access is an interference control approach in which PUs and SUs can transmit simultaneously, provided that the SUs keep their transmit power below a certain threshold [10]. Overlay spectrum access similarly involves simultaneous transmission over a single band, but in this case the SUs cooperate with PUs by relaying PU data alongside their own messages in order to offset the effects of interference [18]. The power cost of relaying PU data is therefore exchanged for bandwidth. CR users must also have prior knowledge of the PUs' data encoding methods in order to properly employ this access method [1].

2.5 State of the Art

The concept of cognitive radio was first introduced in 1999 by Dr. Joseph Mitola at the Royal Institute of Technology in Stockholm. He envisioned cognitive radio as a "goal-driven framework in which the radio autonomously observes the radio environment, infers context, assesses alternatives, generates plans, supervises multimedia services, and learns from its mistakes" [19]. His vision was initially "a mere baby step in a potentially interesting research direction" [20], but the initiative quickly began to gain momentum in academics and regulatory agencies, especially with the rising urgency of the spectral scarcity dilemma. Since that time, cognitive radio has become one of the most heavily researched areas in communications [13], [21]. Over the course of the past two decades, several notable standards and use cases that apply CR principles have been realized. As is discussed in this section, these milestones have played a pivotal role in shaping the ongoing evolution of CR and could be strong indicators of what the future may hold for CR.

2.5.1 TVWS and Related Standards

One of the most important CR milestones came in 2008, when the FCC approved the unlicensed access to white spaces in the 54-72 MHz, 76-88 MHz, 174-216 MHz and 470-806 MHz television bands [10]. This ruling was the culmination of the long standing initiative to promote more efficient use of the spectrum. The decision also came in light of many television stations switching from analog to digital transmission systems, which freed up significant amounts of spectrum in the licensed television bands. The favorable propagation characteristics of these bands made them an especially attractive option for wireless service providers. Therefore, the FCC believed that the new rules would "allow wireless broadband providers that use unlicensed devices to reach new customers and to extend and improve their services in rural areas...without disrupting the incumbent television and other authorized services that operate in the TV bands" [22]. The ruling led to the emergence of several standardization initiatives and amendments, including IEEE 802.22, 802.11af, and 802.15.4m [23].

IEEE 802.22, often known as "Wi-Far", is a standard that defines CR-based TV white space access techniques for wireless regional area networks (WRANs). This proposed radio interface, which could cover ranges of up to 100 km, intends to bring broadband services to remote areas while avoiding interference with TV band incumbents. Since its publication in 2011, 802.22 has further developed standards such as 802.22.1 ("Enhanced Interference

Protection of the Licensed Devices"), 802.22.2 ("Recommended Practice for Installation and Deployment of IEEE 802.22 Systems"), and 802.22.3 ("Standard for Spectrum Characterization and Occupancy Sensing") [23]. There have also been amendments to the baseline IEEE Std 802.22-2011, including 802.22a ("Management and Control Plane Interfaces and Procedures and enhancement to the Management Information Base (MIB)") and 802.22b ("Enhancement for Broadband Services and Monitoring Applications"). Overall, IEEE 802.22 is highly significant in that it is the first completed IEEE standard that specifies a standardized CR interface for white space access.

Another significant standard, 802.11af, was published in 2013 as an amendment to the IEEE 802.11 wireless local area networking (WLAN) standard. This amendment extends WLAN operability to TVWS, much in the same way that 802.22 does for WRANs [23]. Similarly, IEEE 802.15.4m builds on IEEE 802.15.4's specifications for low-rate wireless personal area networks (LR-WPANs) with guidelines for operation in TV white spaces. IEEE standards like these are an important aspect of the ongoing CR evolution in that they provide the mechanisms for practical implementations, in this case with respect to TVWS access. Significant contributions to the CR model for TVWS have also come from researchers outside of IEEE working groups. For example, [24] proposed a generalized enhanced detection algorithm (GEDA) that seeks to improve probability of detection and address the hidden node issue. Other researchers have proposed optimized sensing techniques and test statistics as well [13]. There have also been a variety of CR paradigms proposed for the use of TVWS in Neighborhood Area Networks (NANs) [13]. Research contributions of this sort will likely play a key role in the gradual adoption of CR use cases for the TV band.

While the recommendations for CR implementations for TVWS are laid out by these standards and research contributions, it is worth noting that there are little to no concrete examples to date of real-world CR deployment in the TV band [25], [26]. The authors of [24] explain that “the main obstacles to TVWS adoption are reliable detection of primary users i.e., TV operators and consumers, allied with specifically, the hidden node problem,” and the authors of [26] add that since “radio regulations, such as spectrum mask and maximum transmission power, are different depending on countries, it is not so simple to implement the standard.” Therefore, further investigation into the performance capacity of potential CR use cases for TVWS is necessary. This trend of performance uncertainty stalling CR deployment is not unique to the TV band; indeed, virtually all CR deployment initiatives are met with this challenge, which will be discussed further in section 2.5.3.

2.5.2 Practical Applications of CR Techniques

Mobile communication networks have historically experienced exponential growth in demand for wireless services since the deployment of the 2G GSM standard [21]. In light of this trend, mobile network operators (MNOs) have had to come up with innovative ways to use spectrum resources more efficiently in order to increase network capacity, including, for example, higher bit transmission and small cell densification. While these physical layer solutions are aimed towards maximizing the use of existing spectrum resources, there is a theoretical upper bound to how efficiently this finite pool of resources can be used. For this reason, the prospect of accessing unused spectrum resources, made possible by cognitive radio technology, has emerged as another way in which mobile network operators might optimize their networks [21].

There are several instances to date of CR principles being applied to mobile communication networks [21]. One notable example includes the spectrum access system (SAS). Initially approved by the FCC in 2015, SAS is a cloud-based service that coordinates shared access to the Citizen's Broadband Radio Service (CBRS) band (3.55 GHz to 3.7 GHz) amongst three tiers of users [27]. The first tier is populated by incumbents, including military radar, fixed satellite systems, radio location services, and terrestrial wireless systems operating primarily near coastal areas. Members of this tier transmit freely and are protected from interference by lower tier users. The second tier consists of Priority Access (PA) users. Prospective PA users bid for Priority Access Licenses (PALs), which give them access to a 10 MHz portion of the CBRS band. PALs are 10 year renewable and are issued on a county-by-county basis [28]. PA users frequently consist of mobile network operators, utility companies, and hospitals; some of the winning bidders of the latest auction (finalized as of October 2020) include Alabama Power Company, Aeronet Wireless Broadband LLC, and Hawaiian Electric Company, inc [28]. Tier 2 users must vacate a given channel should an incumbent need to transmit there, for which reason their reserved 10MHz portion may span the CBRS band dynamically rather than existing at a fixed frequency. The third tier is composed of General Authorized Access (GAA) users. These users are guaranteed opportunistic access to at least 80 MHz of CBRS spectrum, but they receive no protection from tier 1 and tier 2 transmission. GAA users do not require a license for operation, but they must be registered with the SAS.

Figure 2.4 shows a high-level layout of the CBRS network architecture. Coordination between the three classes of users is facilitated by central SAS entities. Tier 2 and 3 users are often referred to collectively as CBRS Service Devices (CBSDs). The central SAS entities

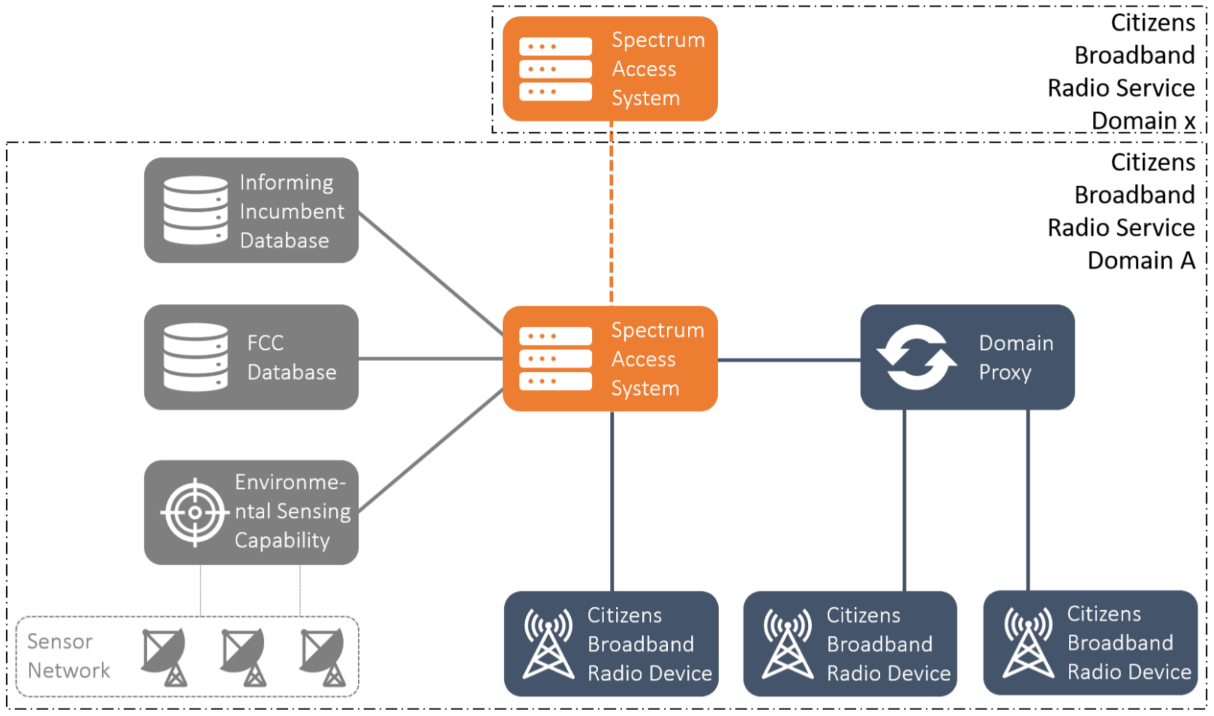

Spectrum Access System
Citizens Broadband Radio Service Domain x
Citizens Broadband Radio Service Domain A
Informing Incumbent Database
FCC Database
Environmental Sensing Capability
Spectrum Access System
Domain Proxy
Sensor Network
Citizens Broadband Radio Device
Citizens Broadband Radio Device
Citizens Broadband Radio Device

Figure 2.4: Spectrum Access System Network Architecture

are responsible for dynamically assigning frequency channels and maximum transmit power to the different CBSDs in the network in order to protect incumbents and license holders while ensuring seamless transmission for all users. The SAS makes these assignments based on information coming from various databases and sensing networks, as seen in Figure 2.4 [29]. In this sense, the CBSDs act as cognitive nodes in a centralized, infrastructure based network. Registered CBSDs must provide certain information, such as geographical location and tier status, to the SAS database. The SAS may interact with a CBSD directly or through a domain proxy (such as in the case of interfacing with legacy radio equipment or networks of CBSDs). In any case, it is the CBSD that initiates communication with the SAS. In order to begin transmission, CBSDs must submit a formal request to the SAS for spectrum access in a certain geographic area. In response, the SAS will grant this request with an assignment to an unused channel. When the transmission is complete, that channel will be added back to the SAS' list of unoccupied frequency bands available for other users.

Unlike other statically managed frequency bands, SAS' hybrid licensing scheme strikes an important balance between interference mitigation and efficient use of spectrum resources. For this reason, the deployment of SAS in the CBRS band opens the door for many innovative use cases, particularly in regards to new business models for 4G and 5G service delivery [29]. This serves as a testament to the extensive benefits to be reaped through the application of CR principles.

Additional applications of CR principles in the realm of mobile communications can be found in the LTE Unlicensed (LTE-U) initiative, licensed assisted access (LAA), MuLTEfire, cognitive femtocells, device-to-device (D2D) communications, and narrowband Internet of Things (NB-IoT) [21]. In general, the incorporation of CR techniques enhances existing systems with more efficient spectrum access solutions and interference mitigation [30].

There are instances of CR features being explored for other wireless communication systems as well. For example, CR has been considered for military communications, in which case the system could intelligently locate vacant spectrum bands even if communication is obstructed by an enemy [21]. CR based solutions for public safety networks have also received significant attention [4], [3], [31]. CR techniques could allow emergency responders to communicate on an ad-hoc basis in the event of a natural disaster that damages network infrastructure [3], [32], [21]. CR technology could also play an important role in bridging the gap between incompatible radio standards in heterogenous networks, encouraging interoperability [4], [31]. Overall, these initiatives are a reflection of the wide range of possibilities that cognitive radio has to offer.

2.5.3 Ongoing Challenges

Despite the fact that cognitive radio is such a heavily researched topic, its commercial use remains rather limited. It is worth reiterating that many of the CR features and network paradigms discussed in sections 2.2 through 2.4.4 have been proposed and examined extensively by researchers but have not necessarily been adopted in real-world applications. To that end, there are several challenges to be overcome before CR can be employed to its full potential.

At its core, CR is a means for opportunistic spectrum access amongst various users. For this reason, the prospect of practical CR realization ultimately begins with a regulatory model that promotes dynamic use of the spectrum. However, that is far from the case for the static, auction-based spectrum assignment policies that are currently in place [25], [1]. Before government bodies will be willing to modify existing regulatory policy, CR must present a viable business case, otherwise entrepreneurs will not be compelled to invest in the new technology [25]. This situation is further complicated by the fact that modern cellular networks already offer stupendous data rates at low costs in areas with modest populations; CR will likely have to offer a highly sought-after service that cellular networks do not already provide in order to present a compelling business case. It is possible that CR's potential lies primarily in bringing connectivity to rural areas, a goal which was notably part of the FCC's rationale for opening up the TV band for CR use [22].

CR's potential as a viable business case also depends on its performance capacity, namely its ability to guarantee high QoS to secondary users without imposing on primary users. It follows that at the root of many of the dilemmas that account for CR's relatively slow

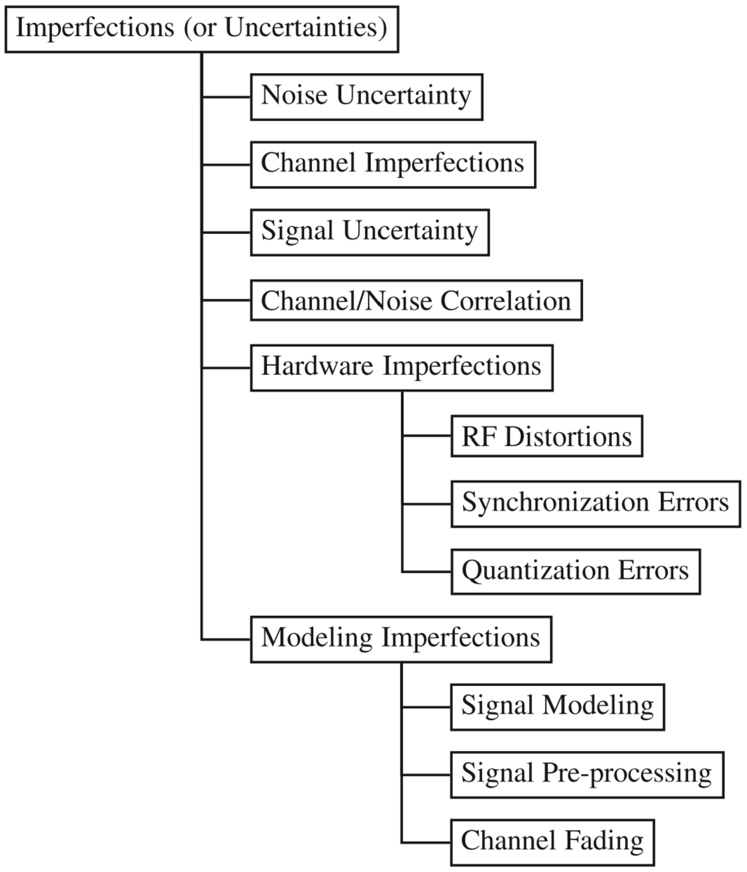


Figure 2.5: Uncertainties of cognitive radio systems

deployment is a shortage of models that can accurately predict CRN performance in practical settings. Due to the inherent uncertainty associated with wireless systems, these models are notoriously difficult to construct and decipher. Some notable instances of uncertainties in the context of CR system modeling are listed in Figure 2.5 [5]. While there are ample models for CR proposed in the literature, many of these models are academically oriented and tend to omit these uncertainties for the sake of analytical readability. As a result, the conclusions drawn from models of this nature may not necessarily be applicable to real world CR implementations. This trend has contributed significantly to the uncertainty surrounding CRN performance and in turn to the limited practical use of CR.

CR's future as a commercially applicable technology will very likely depend on a better understanding of its performance capacity [5]. The lack of commercial use cases for TVWS is compelling proof of this conjecture; even with a relatively flexible regulatory model and several established standards in place, performance uncertainty has hindered deployment [24]. CRN performance is jointly characterized by interference mitigation and secondary throughput. There are a number of factors that may affect either one of these performance aspects, including hardware specifications, network architecture organization, and, most importantly, the methods for CR knowledge acquisition and primary user detection.

The various approaches to spectrum sensing and CR learning in general account for significant ongoing research challenges in the CR community. While each sensing technique comes with inherent advantages and disadvantages, there tends to be a recurring trade-off between reliability and computational complexity [13]. In general, CR system designers must strike a crucial balance between a detection method that is both accurate and suitable for real time hardware applications. More sophisticated sensing methods such as cooperative sensing, out-of-band sensing, and the use of geolocation databases have been proposed to remedy some of the unreliability issues that arise with in-band spectrum sensing, as mentioned in section 2.4.1 [1]. While these methods allow for more accurate primary user detection, they tend to introduce additional delay, overhead, energy consumption, and security risks that must be taken into account [1], [9], [33]. Furthermore, PUs must actively agree to work with CR users in order to realize setups such as out-of-band sensing, database access, and overlay spectrum access, which is something frequency band incumbents will not be inclined to opt for so long as the presence of CR users in their licensed band could cause harmful interference. This attitude is readily exemplified by the TV broadcasters' opposition to the FCC's TVWS decision, as well as by the controversy surrounding LTE-U's potential impact on Wifi devices in the unlicensed 5GHz band [22], [21]. In this regard, the mindsets of researchers, regulatory overseers, frequency band incumbents, and entrepreneurs must come together in order for the deployment of CR to be logistically feasible [25]. This collaboration will necessarily entail further investigations into CR performance from a deployment perspective as well as additional research initiatives aimed towards improving primary user detection methods.

2.6 Conclusions

In this chapter, an overview of the theoretical capabilities and practical implementation challenges of cognitive radio as outlined by the literature has been presented. From this discussion it can be concluded that while CR has many powerful features in theory, in practice its performance in real-world settings is not well known and commercial deployment has been scarce as a result. Ongoing research challenges for CR frequently relate to improving the ways in which the CR obtains knowledge, namely through spectrum sensing and possible database access. Overall, CR has a ways to go before it can be implemented on a large scale, but the wealth of research and existing standards are a promising sign for this technology.

Chapter 3

Experimental Background

Given the importance of reliable PU detection to the performance of the CR system and feasibility of CR deployment, there is a great deal of research dedicated to perfecting the various CR sensing methods [13]. In keeping with this theme, our experimental CR implementation will also focus on the relationship between spectrum sensing technique and system performance. Our goal is to provide a simple implementation of a pair of secondary users who can use spectrum sensing to avoid interference with a primary user. Presented in this chapter are the different aspects of our experimental setup. These include equipment used, flowgraphs and blocks created, and physical setup of the USRP systems.

3.1 Equipment

The following subsections will provide additional information about the components of our experimental CR setup as a preface to our implementation method.

3.1.1 Computer Systems

For our experimental setup, three laptop PCs were used: a Dell XPS 9500, a Lenovo Yoga 720, and a Toshiba Satellite. Each PC ran a copy of Ubuntu 16.04 LTS and were equipped with at least one USB 3.0 port to which the USRPs were connected. As the CR systems would require more computational power then the PU system, the Lenovo and the Dell were chosen to run the CR systems since each are equipped with a quad-core Intel i7 processor.

3.1.2 Ettus B210 USRP

For the SDR component of our CR, the Ettus B210 Universal Software Radio Peripheral (USRP) was used. Each B210 device features a fully integrated, dual-channel RF front end, capable of transmitting and receiving signals with carrier frequencies ranging from 70 MHz to 6 GHz, along with a fully reprogrammable Spartan6 XC6SLX150 FPGA that can send and receive samples from the ADCs and DACs at up to 61.44 megasamples per second. The RF front end features the AD9361 chip, which is capable of streaming data with 200kHz - 56 MHz of analog bandwidth. The Spartan6 FPGA is responsible for controlling the AD9361, and the USRP connects to a host PC via USB 3.0. The USB 3.0 power bus is sufficient to power the B210, which makes for a convenient setup. The USRP Hardware Driver (UHD) provides the communication interface between the USRP and host PC, as shown in Figure 3.1 [34]. The FPGA code is written in Verilog, whereas the UHD is written in C and C++. The UHD allows the user to transmit and analyze RF data, as well as control various features of the USRP, including sampling rate, power, and frequency, through software on the host PC. The user can either build their own signal processing applications on top of the UHD

API, or use software platforms like GNU Radio or LabVIEW that interface with UHD and provide existing tools for signal processing.

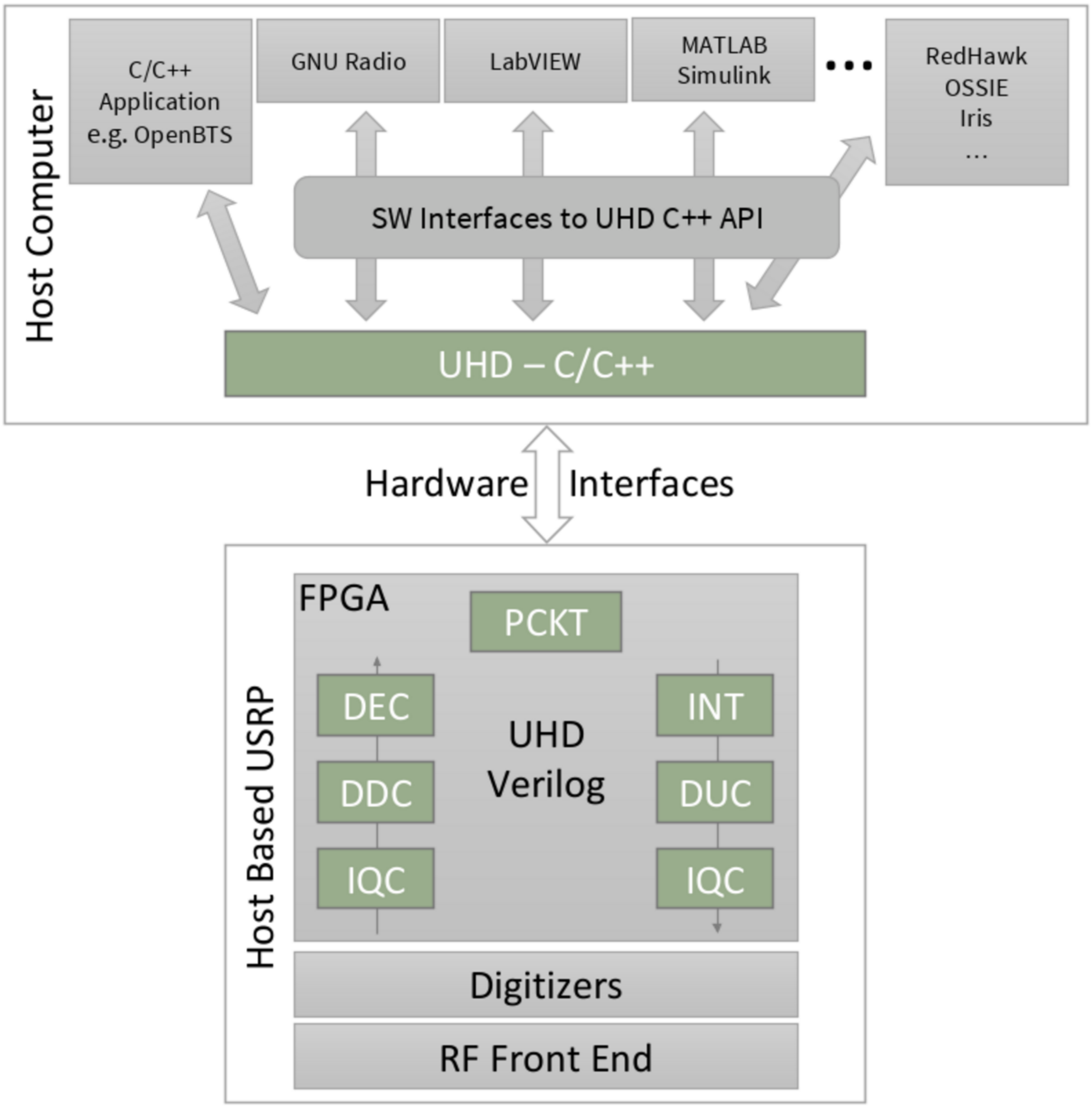
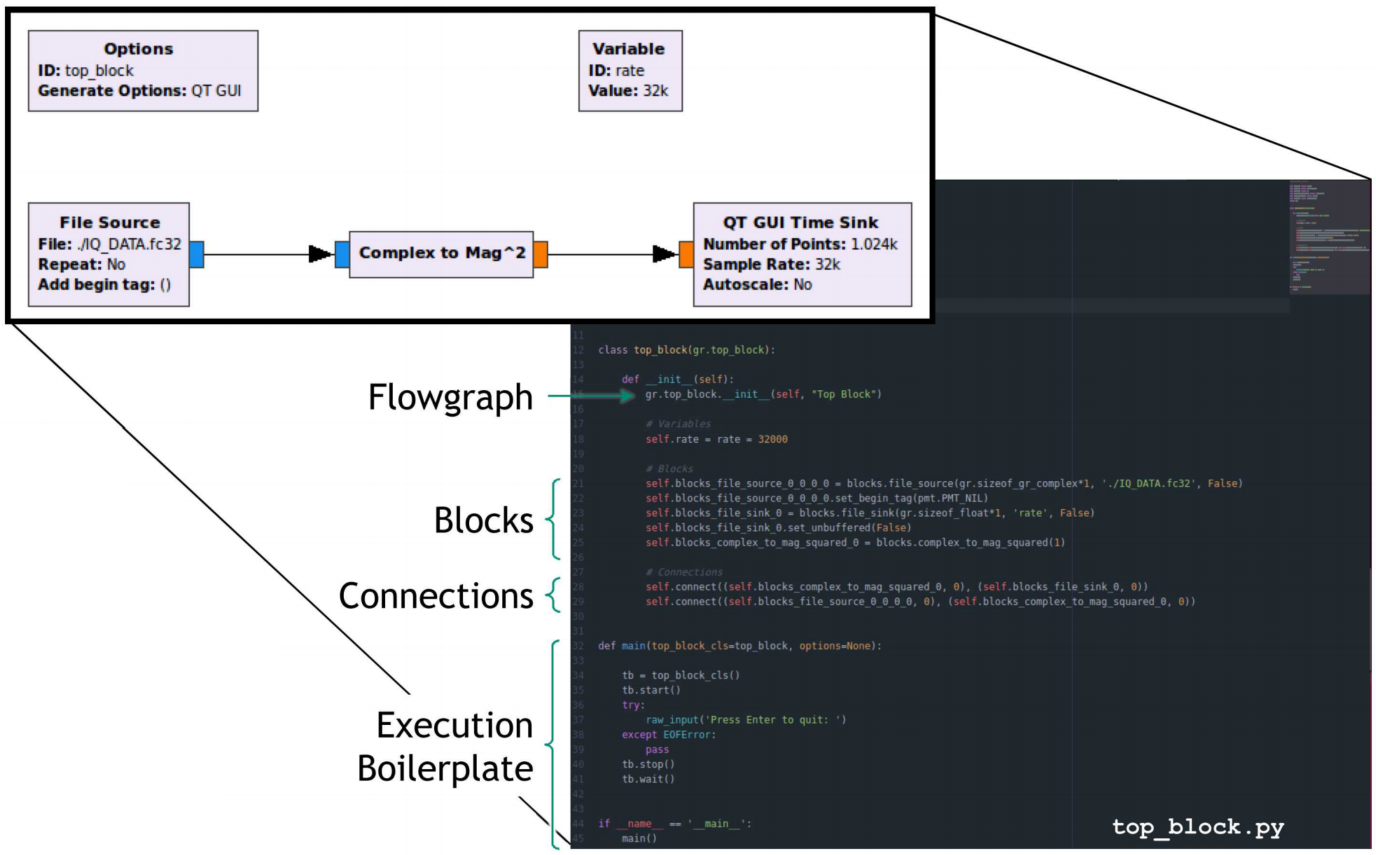


Figure 3.1: Interface between USRP and a host computer

3.1.3 GNU Radio

For the software backend of our SDR hardware platform, GNU Radio was used. GNU Radio is a free and open source software framework that provides a library of signal processing tools to facilitate the development and analysis of radio systems. GNU Radio's graphical tool, GNU Radio Companion (GRC), uses a flowgraph-oriented user interface in which signal processing functionality takes the form of interconnected software blocks, as shown in Figure 3.2 [35]. The blocks are interconnected with arrows that represent the flow of data, with different colors signifying different data types.



```python

class top_block(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Top Block")

        # Variables
        self.rate = rate = 32000

        # Blocks
        self.blocks_file_source_0_0_0_0 = blocks.file_source(gr.sizeof_gr_complex*1, './IQ_DATA.fc32', False)
        self.blocks_file_source_0_0_0_0.set_begin_tag(pmt.PMT_NIL)
        self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_float*1, 'rate', False)
        self.blocks_file_sink_0.set_unbuffered(False)
        self.blocks_complex_to_mag_squared_0 = blocks.complex_to_mag_squared(1)

        # Connections
        self.connect((self.blocks_complex_to_mag_squared_0, 0), (self.blocks_file_sink_0, 0))
        self.connect((self.blocks_file_source_0_0_0_0, 0), (self.blocks_complex_to_mag_squared_0, 0))


def main(top_block_cls=top_block, options=None):

    tb = top_block_cls()
    tb.start()
    try:
        raw_input('Press Enter to quit: ')
    except EOFError:
        pass
    tb.stop()
    tb.wait()


if __name__ == '__main__':
    main()
```

Figure 3.2: Example of a GNU Radio flowgraph and its generated Python code

The execution of a GRC flowgraph is accompanied by the creation of a Python script that contains the various flowgraph elements, integrated together in code. While the unification of flowgraph components is implemented using Python, the block routines themselves are typically written in C or C++. For this reason, SWIG is needed to give the Python scripts access to the C++ routines [36]. Overall, the GRC greatly simplifies the use of GNU Radio

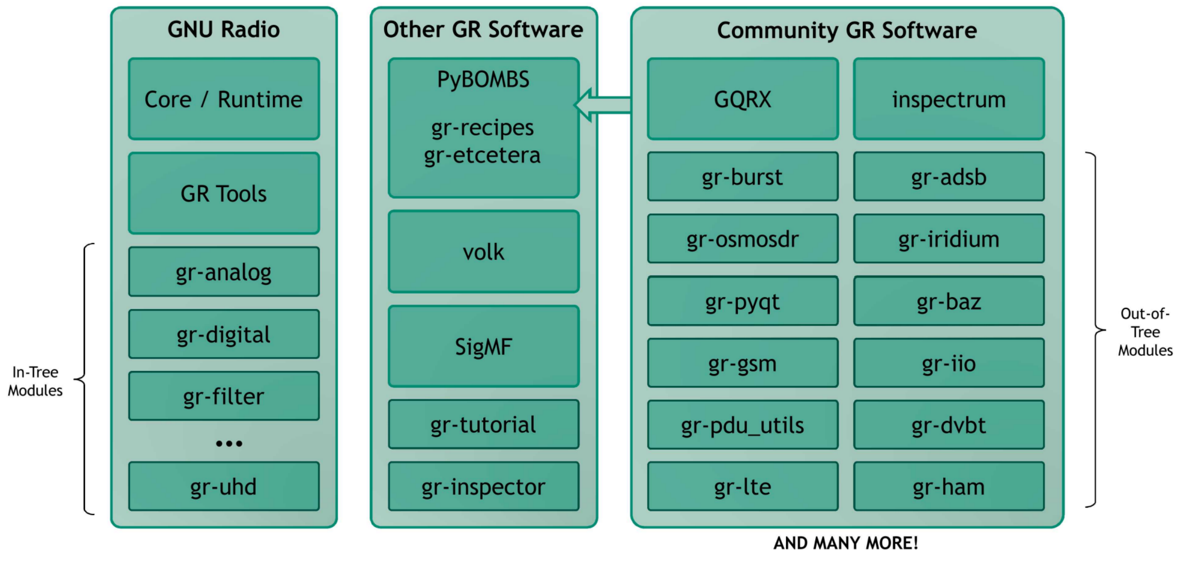


Figure 3.3: Organization of GNU Radio software

by providing a graphical interface through which to generate this integrated Python script as opposed to the user writing it from scratch.

In addition to an extensive collection of signal processing tools, GNU Radio comes with an array of example codes and tutorials to help beginners get started. For more advanced users, GNU Radio also provides mechanisms for personalization outside of the realm of built-in tools. For example, users that want to add certain features to existing blocks or design entirely new functionality can create a custom embedded Python block. These flexible features are due in large part to the collaborative nature of GNU Radio as an open source project. Each new user has the potential to make a contribution to the GNU Radio project as a whole.

Figure 3.3 [35] further illustrates the extent to which user contributions extend GNU Radio's capabilities. In addition to the signal processing building blocks that GNU Radio provides, there are also built-in tools, like PyBOMBS and gr modtool, that assist in the development and organization of custom modules. It is readily apparent, then, that community involvement is a crucial component of the GNU Radio project. Because of this, GNU Radio is constantly improving, which is one of the major advantages to working with open source software. These benefits, combined with the fact that GNU Radio is the leading software platform supporting USRP drivers, made it the obvious choice for our experiment.

3.2 Flowgraphs, Blocks, and External Python Code

The following section discusses the GNU Radio flowgraphs and blocks that were used, but do not provide spectrum sensing capabilities. The spectrum sensing flowgraphs are discussed in section 3.3. The components explained this section were responsible for providing network capabilities, controlling channel switching, and simulating primary users during the experiments. In addition, some external Python code was used to accumulate results from large quantities of tests and form UDP connections between the secondary users.

3.2.1 GR-MAC

As a basis to develop our CR system, the GNU Radio MAC (GR-MAC) program was used. GR-MAC is a simple, open-source program developed in GNU Radio that acts as a MAC layer, the lower layer 2 sub-layer of the internet protocol stack, which is shown in Figure 3.4, for use with USRPs [37].

The GR-MAC program creates a TUN/TAP interface within the Linux operating system that acts as a virtual network interface device which can be assigned an IP address. The GR-MAC program, which is synonymous with *simple_trx.py* in the further sections, encapsulates incoming IP datagrams from the network layer into Ethernet frames, modulates the bits of

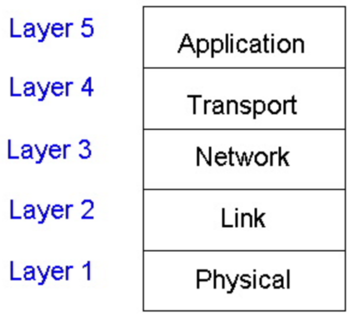


Figure 3.4: Internet Protocol stack

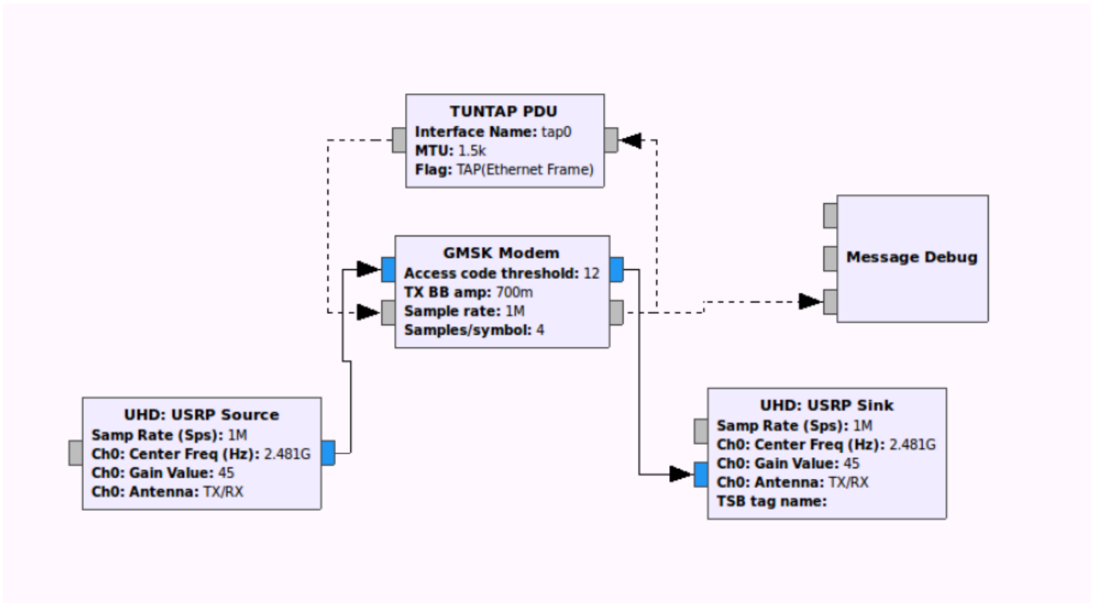


Figure 3.5: GR-MAC Flowgraph in GNU Radio

the frames via a GMSK modem, and sends them to the physical layer which is the USRP. Incoming frames are demodulated, sent to the TUN/TAP interface (layer 2), decapsulated, and continue to be sent up the protocol stack, as shown in Figure 3.5.

IP datagrams and link-layer Ethernet frames are passed in GNU Radio via messages, which are represented by the dashed lines. The bits of these messages are modulated and demodulated via the GMSK signal processing block shown in the middle of the flowgraph. In the case of an IP datagram being transmitted by the program, the datagram is first received by the TUN/TAP interface block as an Ethernet frame. The bits of the Ethernet frame are then converted to GNU Radio messages and sent to the GMSK signal processing block and are modulated into a complex valued stream. This stream is then passed to the UHD sink block. The UHD then sends the data stream to the USRP, which acts as the physical layer (layer 1). The USRP transmits the signal at a center frequency specified within the UHD sink block. This process is performed in reverse order in the case of the program receiving IP datagrams. GMSK modulated signals are received by the USRP at a center frequency specified by the UHD source block and are passed to the UHD. The UHD then passes this signal through the UHD source block in the form of a complex valued stream, which is then demodulated by the GMSK signal processing block. The GMSK signal processing block then outputs a GNU Radio message carrying the bits of the transmitted Ethernet frame to the TUN/TAP interface. At this point, the TUN/TAP interface then passes the Ethernet frame up to the network layer (layer 3) as an IP datagram.

3.2.2 Channel Switching Block

Before a description of the channel switching block is given, its parameters and terminology are defined. In summary, the purpose of the channel switching block is to control the channel switching features in a secondary user. The channel switching block is designed to be easily implemented in any secondary user, without requiring that user to be aware of what channel sensing method is being used. The channel switching block makes a decision once every cycle, and tells the USRP source and sink blocks to change frequencies using message passing.

A cycle is defined as the completion of the set of the three steps in our cognitive radio. The first step is the channel sensing, during which TX/RX is impossible. The second step is channel switching. The third step is TX/RX. In general, the third step is the longest of the three.

The time interval during which a cycle is completed is called the cycle interval, I_c. Since there are three steps in a cycle, I_c is the sum of the time intervals of each of the steps.

The time interval during which channel sensing occurs is called the channel sensing interval, I_s. I_s is the product of the time interval it takes to sense one channel I_{s_0}, and the number of channels being checked, N. This relationship is shown in (3.1);

$$I_s = I_{s_0} N. \tag{3.1}$$

The time interval during which channel switching occurs is extremely small. The decision is made within one single execution of the Python code described later in this section and, for the purposes of the experiment, may be approximated to be 0.

The time interval that is allowed for TX/RX, called the TX/RX interval, I_{rt}, is the remaining amount of time in the cycle. Since the cycle interval I_c is the sum of each of the intervals of the steps, the duration of the TX/RX interval can be found using (3.2);

$$I_{rt} = I_c - I_s = I_c - I_{s_0} N. \tag{3.2}$$

It should be noted that the value of I_s is usually very small compared to the value of I_{rt}, because the TX/RX time interval is typically much larger than the channel sensing interval.

By simple algebraic manipulation, the entire cycling interval is given by (3.3);

$$I_c = I_s + I_{rt} = I_{s_0} N + I_{rt}. \tag{3.3}$$

The time at which the first step in the cycle begins must be determined and understood by both secondary users. This is because, if one of the users is in the TX/RX step while the other user is in the channel sensing step, no data can be traded. Therefore, the secondary users must be synchronized to minimize data loss. Synchronization is achieved by using Python's time library and a float modulus operator. Since the time library uses the system time, the system times of the secondary users must be synchronized using the Internet before the session begins. The current time is represented as T_t. Its units are seconds since Unix epoch. When (3.4) is true, the first step in the cycle begins;

$$T_t \bmod I_c \approx 0. \tag{3.4}$$

Therefore, the time at which both primary users will enter their first step is synchronized.

An example for a possible session is given in Table 3.1. In this example, $I_c = 5$ seconds and $I_{s_0} = 0.2$ seconds. The secondary users begin their transmission in a vacant channel, and at 12:00:07, a primary user starts transmitting, making the channel occupied. Table 3.1 explains how the secondary users would react under the given conditions.

The channel switching block was implemented using an embedded Python block in the GNU Radio flowgraphs, as shown in Figure 3.6.

The Python block is used in both the matched filter detector and the energy detector. The inputs to the block are the moving average and two message strobe blocks. The moving average provides the block with a number between 0.0 and 1.0, which it uses to determine whether or not a primary user is present. The message strobe blocks send messages regularly every 500 ms. In the top message strobe, the value sent is a custom frequency value, which

Table 3.1: Example scheduling timeline for secondary users with a cycle interval of 5 seconds and a sensing interval of 0.2 seconds

Time	Action taken by CR	Resulting Frequency (MHz)	Resulting Step
11:59:59.00	Channel switching for the SU is turned on by the user. A new iteration of the cycle will occur at 12:00:00, because 12:00:00 (in seconds since Unix epoch) mod 5 = 0. The CR starts in the TX/RX step.	2481	TX/RX
12:00:00.00	The channel sensing step begins.	2481	Sensing
12:00:00.20	The current channel has been checked, and a decision is made based upon the average maintained throughout this step. The channel is determined to be unoccupied, and, since the channel switching decision step takes approximately no time, the TX/RX step begins almost immediately.	2481	TX/RX
12:00:05.00	The TX/RX step concludes, and the channel sensing step begins.	2481	TX/RX
12:00:07.00	A primary user enters the channel. Once this occurs, the secondary users will experience high packet loss.	2481	Sensing
12:00:10.00	TX/RX stops, channel sensing begins in the occupied channel.	2481	Sensing
12:00:10.20	The previous channel (2.481 GHz) is found to be occupied, so the secondary users check the next channel (2.482 GHz). This new channel must be checked for primary users before it can be used.	2482	Sensing
12:00:10.40	The new channel (2.482 GHz) is found to be suitable, so TX/RX begins. Notice that the time that it took to do the channel analysis is $I_s = I_{s_0}N = 0.2$ seconds * $2 = 0.4$, because two channels were checked.	2482	TX/RX

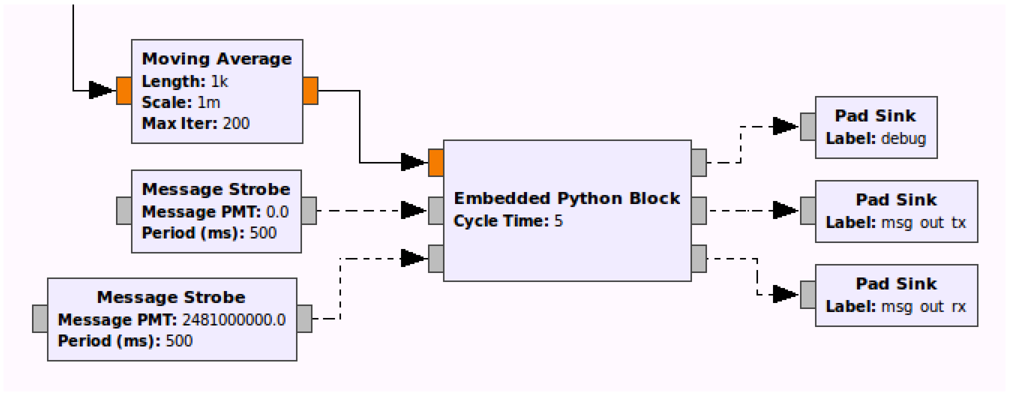


Figure 3.6: Embedded Python block and neighboring blocks from the energy detector flowgraph

is used by the USRPs when the channel switching function is turned off. In the bottom message strobe, the value is a boolean, which GNU Radio interprets as a float. If the float is set to one, then channel switching is turned on. If it is set to zero, then channel switching is turned off. Both of the values provided by the message strobes are configurable by the user at runtime, using the QT GUI.

The outputs from the Python block are three pad sinks, which are fed to the parent block. The first pad sink is a debug, which allows relevant information to be output to the terminal. The second pad sink is a message output system for the transmitter, which is linked to the USRP sink block in the parent graph. This allows the Python block to tell the USRP sink to change its center frequency and gain. The last pad sink is a message output system for the receiver, which is linked to the USRP source block in the parent graph. This allows the Python block to tell the USRP source to change frequencies.

While the full content of the Python code is given in the appendices, a discussion of the code is given below, and Figure 3.7 is included to describe the main work function.

This first part of the code is simply the declaration of the variables. Because of the nature of GNU Radio, global variables are used. Otherwise, the values for the variables

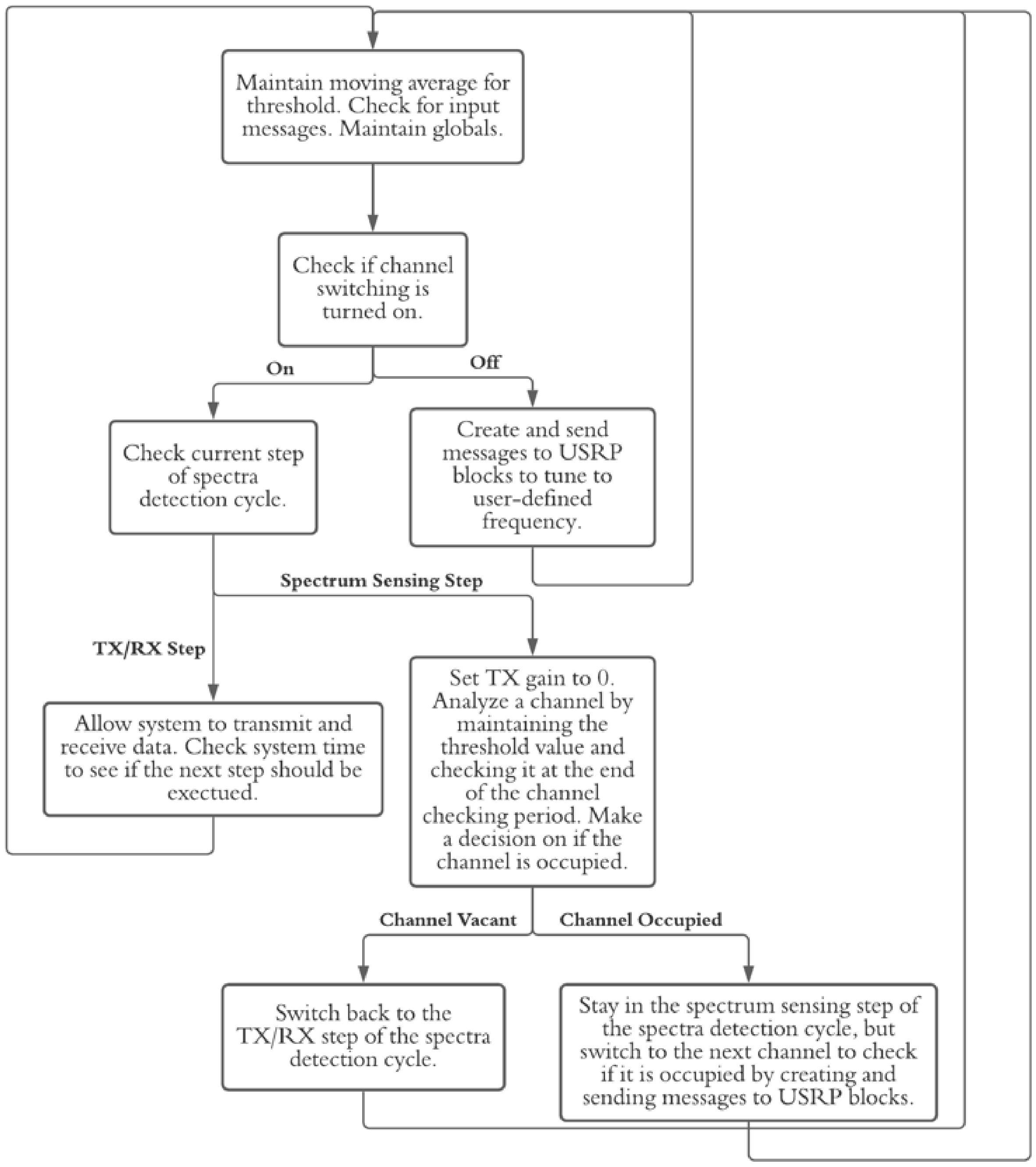

Maintain moving average for threshold. Check for input messages. Maintain globals.
Check if channel switching is turned on.
On
Off
Check current step of spectra detection cycle.
Create and send messages to USRP blocks to tune to user-defined frequency.
Spectrum Sensing Step
TX/RX Step
Allow system to transmit and receive data. Check system time to see if the next step should be exectued.
Set TX gain to 0. Analyze a channel by maintaining the threshold value and checking it at the end of the channel checking period. Make a decision on if the channel is occupied.
Channel Vacant
Channel Occupied
Switch back to the TX/RX step of the spectra detection cycle.
Stay in the spectrum sensing step of the spectra detection cycle, but switch to the next channel to check if it is occupied by creating and sending messages to USRP blocks.

Figure 3.7: A logical flowchart of the embedded Python block

would be lost every time the Python code is executed on a new set of data in the data flow. The CHANNELS array is the list of possible frequencies that may be used by the secondary users. These are separated by 1 MHz each, and all exist in the unused 14th Wifi channel, to avoid possible contamination of the experiment by unpredictable users. The purpose of the *increment_channel* method is to make sure that the variable tracking the current channel would always stay within in the range of the CHANNELS array. In the initialization function for the block, the parameters, input signals, and message ports are all defined. For this block, there is only one parameter: cycle time, which is used to determine how long the CR will transmit for before it performs channel sensing. The input signal is the output from the moving average block, from which the determination on channel occupancy is made. The result of the moving average block is discussed in subsection 3.3.1. Message handler methods are used to update the switching and custom frequency variables. These messages are discussed at the beginning of this subsection.

The logic for the main work function of the Python block is shown in Figure 3.7. The work function is split into two parts, depending on the channel switching mode. If channel switching is turned off, the code uses messaging to tell the USRP source and sink blocks to set their frequency to the frequency given by the input message. If channel switching is turned on, the functionality is slightly more complicated. A continuous, rolling average for the input threshold is maintained. The operation of the rest of the block is determined by the checking variable. The checking variable represents the two steps of channel switching operation. The first step is channel sensing. The second step is data transmission and reception. These steps are time-sensitive and dependent upon the scheduling of the cognitive radio.

In the first step, the spectrum sensing commences. While a channel is being checked, the code simply maintains the threshold average. At the end of this time, the average threshold, which is the average of the input signals, is checked against the value 0.5. A value above 0.5 means that the channel is occupied. If the channel is deemed suitable, then the spectrum detection stops, and data trading begins. Otherwise, the process is repeated on the next channel, and so on, until a suitable channel is found. In principle, the cognitive radio remains in the sensing step forever if an acceptable channel is never found.

In the second step, packet transmitting and receiving occurs. At the beginning of this stage, the transmitter gain is turned to an appropriate value via message passing, and at the end of this stage, the gain is turned to zero, so that it doesn't cause interference to the other secondary user. During this stage, the CR is allowed to send and receive data, while the Python code waits until it is time for spectrum sensing to occur.

These two modes occur in a cycle, so that the CR could run indefinitely, checking the spectrum at regular intervals. For example, if the interval were set to 5 seconds, the spectrum detection would occur at the times 12:00:00, 12:00:05, 12:00:10, 12:00:15, ... Determining the best interval is the focus of one of our experiments, discussed in section 4.2.

3.2.3 Rapid Testing Block and Drivers

Some of our experiments required thousands of channel detections and decisions to occur rapidly. While the channel switching block works well for the functional cognitive radio implementations, it does not suit the purpose for high-volume testing. Therefore, it needed to be modified and augmented by another block and some external Python code. The code for these blocks is provided in the appendix of this thesis, but they are described in this section.

In order to execute the rapid tests, the secondary users need to check one channel many times in a row. Even if the channel is occupied, the secondary users need stay in the channel so that checking can continue. Therefore, some of the functionality of the channel switching block needed to be removed. The averaging and comparison to a threshold value is still the same, but, in the modified block, the channel switching does not occur. No message is sent to the USRP sink and source blocks, so that they stay in the same channel. Instead, a new message is sent to another, new block, called the rapid testing block, telling it that the channel is either occupied or vacant. After the message is sent, execution continues normally, and, since the channel was not switched, the secondary user can detect the same channel again.

The function of the rapid testing block is extremely simple. It takes the message received from the simplified channel block and writes it to a text file, numbering each of the messages. The name of the text file is passed to the flowgraph as a parameter, and is supplied by the rapid test driver. The rapid testing block and the simplified channel block are shown in Figure 3.8.

The final feature that needs to be added in order to execute rapid testing is provided by the rapid test driver Python code. This code sends the desired filename and number of desired tests to the rapid testing flowgraphs as parameters. Once the flowgraph has completed the desired number of tests, the number of times a channel was decidedly occupied is counted by reading the log file and a summary is added at the beginning of the log.

Together, these features allowed channel detections to be executed in a short amount of time so that we could obtain large amounts of data in a reasonable time.

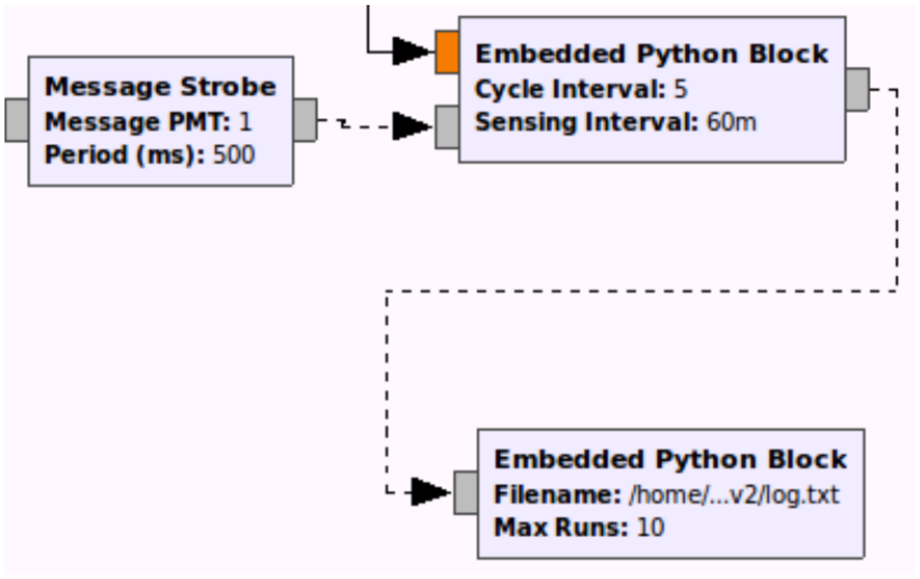


Figure 3.8: The rapid testing block and simplified channel block

3.2.4 Primary User

To simulate the scenario of a channel being occupied by a primary user, the flowgraph in Figure 3.9 was used to transmit a prescribed signal in a given channel.

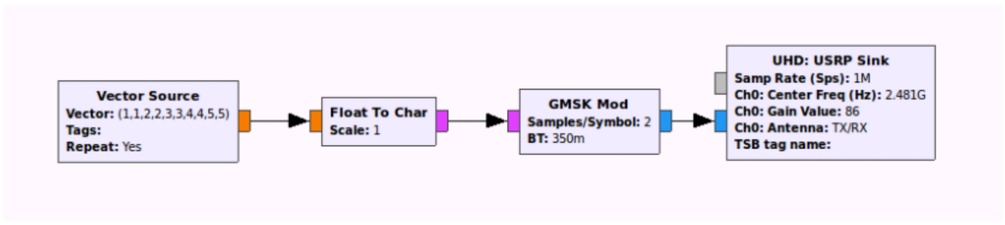


Figure 3.9: Flowgraph of primary user

The signal being sent by the PU was the arbitrarily chosen vector (1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10) modulated by GMSK.

3.2.5 Python UDP Sockets

In order to analyze the quality of the network connection between two secondary users, one-way communication was established using the UDP transport protocol. For the experiments in which this communication was used, discussed in section 4.2, one secondary user would transmit a number of packets of fixed length at random intervals, and the other secondary user would receive the data and count how many packets were received. This allowed the bit error rate of a one-way UDP connection to be calculated from the observed packet loss.

To do this, two Python programs were created, called *udp_transmit.py* and *udp_receive.py*. The entirety of these programs are in the appendix, but their main functions are described here. In both programs, a UDP socket is created with the IP address of the network interface made by the GR-MAC flowgraph and an arbitrarily chosen port number. In *udp_receive.py*, after the socket is opened, the program waits for data on the socket and prints out the total number of packets as they are received.

In *udp_transmit.py*, UDP packets whose data field is 1,000 bytes long are sent to the socket. The interval between sent packets is generated by the random function in Python's random library, whose bounds are 0.05 and 0.5 seconds. The same packet length was used throughout all of the experiments to maintain consistency, but the total number of packets sent was varied based on the needs of the experiment.

Together, these programs allowed bit error rate to be calculated as a metric for the quality of the network connection. This measurement was important to our experiments, because we wanted to see if bit error rate would increase when more time was spent sensing the channels, compared to the amount of time spent transmitting and receiving data.

3.3 Spectrum Sensing Flowgraphs and Theory

In this section, the spectrum sensing methods of energy detection and matched filter detection are discussed. The theory behind each method is explained and the GNU Radio flowgraph implementations of each are shown.

3.3.1 Energy Detector

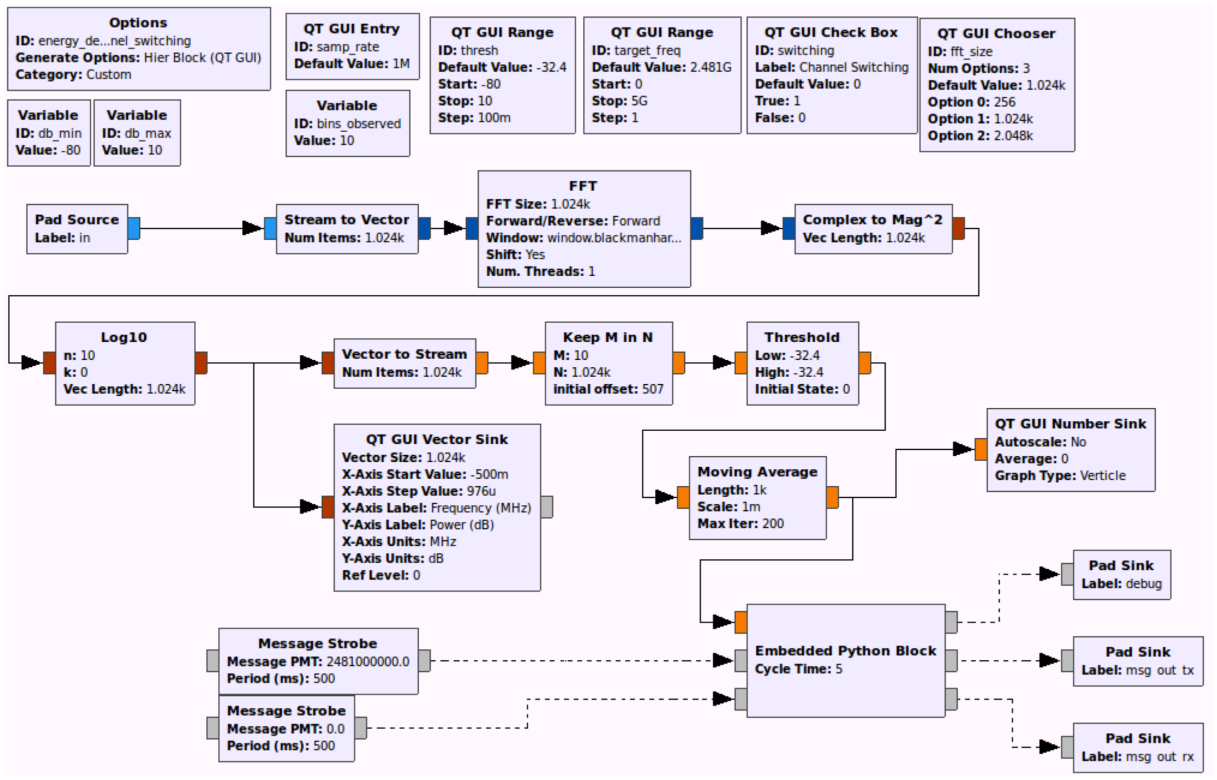
The energy detection method of spectrum sensing computes the energy of the received signal and compares that value with some prescribed threshold value. If the detected energy exceeds the threshold, a primary user is considered present. If not, the primary user is considered absent, and that given frequency band is determined to be a white space. The formula for calculating the energy is given by (3.5);

$$E_x = \sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2 . \tag{3.5}$$

Here, $x(n)$ is the nth sample of the received time signal, N is the number of samples, and $X(k)$ is the N point DFT defined as $\sum_{n=0}^{N-1} x(n)e^{-jkn2\pi/N}$.

The energy detection method is by far the simplest to implement, and the fact that it does not need any prior knowledge of the primary signal characteristic makes it an attractive choice for many CR applications. However, one major disadvantage is its inability to detect primary signals with low SNRs.

In order to construct a working implementation of an energy detector, a flowgraph was designed to use energy detection to make a channel switching decision. This flowgraph,


Options
ID: energy_de...nel_switching
Generate Options: Hier Block (QT GUI)
Category: Custom

Variable
ID: db_min
Value: -80

Variable
ID: db_max
Value: 10

QT GUI Entry
ID: samp_rate
Default Value: 1M

Variable
ID: bins_observed
Value: 10

QT GUI Range
ID: thresh
Default Value: -32.4
Start: -80
Stop: 10
Step: 100m

QT GUI Range
ID: target_freq
Default Value: 2.481G
Start: 0
Stop: 5G
Step: 1

QT GUI Check Box
ID: switching
Label: Channel Switching
Default Value: 0
True: 1
False: 0

QT GUI Chooser
ID: fft_size
Num Options: 3
Default Value: 1.024k
Option 0: 256
Option 1: 1.024k
Option 2: 2.048k

Pad Source
Label: in

Stream to Vector
Num Items: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Complex to Mag^2
Vec Length: 1.024k

Log10
n: 10
k: 0
Vec Length: 1.024k

Vector to Stream
Num Items: 1.024k

Keep M in N
M: 10
N: 1.024k
initial offset: 507

Threshold
Low: -32.4
High: -32.4
Initial State: 0

QT GUI Vector Sink
Vector Size: 1.024k
X-Axis Start Value: -500m
X-Axis Step Value: 976u
X-Axis Label: Frequency (MHz)
Y-Axis Label: Power (dB)
X-Axis Units: MHz
Y-Axis Units: dB
Ref Level: 0

Moving Average
Length: 1k
Scale: 1m
Max Iter: 200

QT GUI Number Sink
Autoscale: No
Average: 0
Graph Type: Verticle

Message Strobe
Message PMT: 2481000000.0
Period (ms): 500

Message Strobe
Message PMT: 0.0
Period (ms): 500

Embedded Python Block
Cycle Time: 5

Pad Sink
Label: debug

Pad Sink
Label: msg_out_tx

Pad Sink
Label: msg_out_rx

Figure 3.10: Energy detector child block

shown in Figure 3.10, was built as a hierarchical block so that it could act as a child block to the GR-MAC flowgraph from Figure 3.5. This relationship is shown in Figure 3.11.

Figure 3.11 shows a flowgraph which is very similar to the GR-MAC flowgraph. The notable difference is the presence of a new block entitled "QT GUI Energy Detector Channel Switching." The purpose of this block is to provide channel switching functionality, based on an energy detection. The signal from the USRP source block is sent straight into the energy detector hierarchical block, in which all of the necessary signal processing and message creation occurs. Messages are passed from outside of this block to the USRP source, USRP sink, and message debug blocks.

The messages which are passed to the source and sink blocks are of two types. The first type is a frequency command, which tells the blocks to switch to a target frequency. The

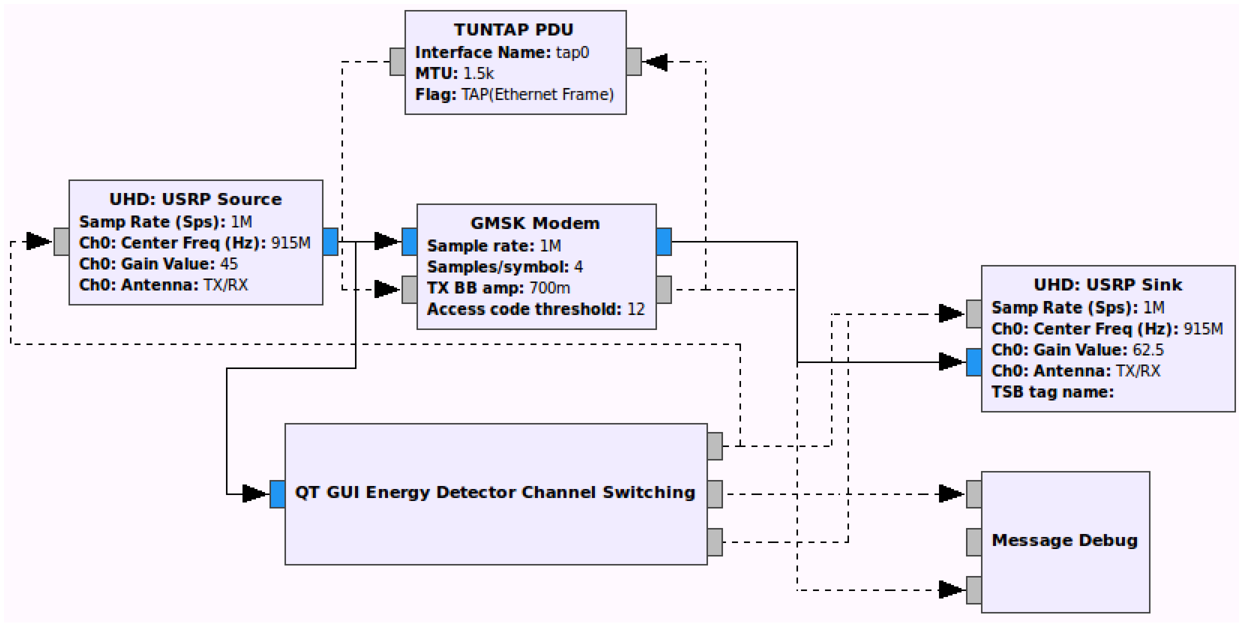


Figure 3.11: Energy detector hierarchical block in the parent GR-MAC flowgraph

second type is a gain command, which turns the transmitter off when spectrum detection is occurring, and back on after spectrum detection is complete. Additionally, some messages are sent to the message debug block in order to view information about the status of the system in the terminal.

While Figure 3.11 shows how the energy detector hierarchical block interacts with the components of the GR-MAC flowgraph, Figure 3.10 shows the flowgraph that is “inside” of the energy detector hierarchical block. The pad source with the blue square on the upper-left corner of the graph is the block which passes the USRP signal from the parent block to the child block. Similarly, there are three pad sinks on the bottom-right corner of the graph which correspond to the output messages seen in Figure 3.11. The data flow of the energy detector block has three main phases, which are described in the following list. Each of the members of this list describe the function of a singular block in the flowgraph shown in Figure 3.10.

1. Conversion from time signal to energy

   - The Pad Source block passes raw, complex data from the USRP Source Block, converted to baseband from the frequency known to the source and sink blocks.
   - A vector is filled with 1024 consecutive samples.
   - The vector is sent to an FFT block. The width of the FFT is 1 MHz, shown by the samp_rate variable in the flowgraph. The frequency resolution can be determined by dividing the sampling rate by the size of the FFT. The size of the FFT is shown in the QT GUI chooser block in the graph, and is 1024. Therefore, the frequency resolution for each block is $(1 \times 10^6)/(1.024 \times 10^3) = 967.5625\,\mathrm{Hz}$.
   - After the FFT block, the complex values in the resulting vector are squared. Therefore, the vector of complex values becomes a vector of floats.
   - The log of each of these values is taken. At this point, the data stream is a vector of floats whose values represent the energy of each frequency bin in decibels. As previously stated, in our graph, the signal from the USRP was converted from the frequency of interest to baseband. Therefore, knowing the width of the FFT was 1 MHz, the vector at this point in our graph contains the energy in the frequencies from $F - 1/2$ to $F + 1/2$ MHz, where F is the frequency of interest. Therefore, the frequency of interest is in the middle of the vector. This will be important information in the next phase of the energy detector.

2. Energy detection

   - The vector is converted back into a stream. The data is the same, but now they are in the form of a stream instead of a vector in order to allow the operation of the threshold detection with the subsequent blocks.

- The “Keep M in N” block counts 1024 (the size of the FFT and the vector before it is converted into a stream) data points (floats), and keeps 10 of them, with an offset. As stated earlier, the values in the middle of the vector are the ones of interest, and the offset guarantees that those values are the ones which are observed. So, the stream of data that is sent out of the Keep M in N block is simply the energy of the 10 middle bins of the FFT.
- These values are compared to a threshold value, which can be changed at execution via a slider in the QT GUI.
- In order to prevent anomalies from triggering false positives or negatives, a moving average is maintained over 1000 stream values.

3. Channel Switching Decision

    - The embedded Python block makes the final decision on if the channel is occupied or not, and what frequency the SU should switch to if it is. The decision of the Python block is powered by messages, and they were sent to the sink and source Blocks in the parent graph. This block is discussed in more detail in subsection 3.2.2.

3.3.2 Matched Filter Detector

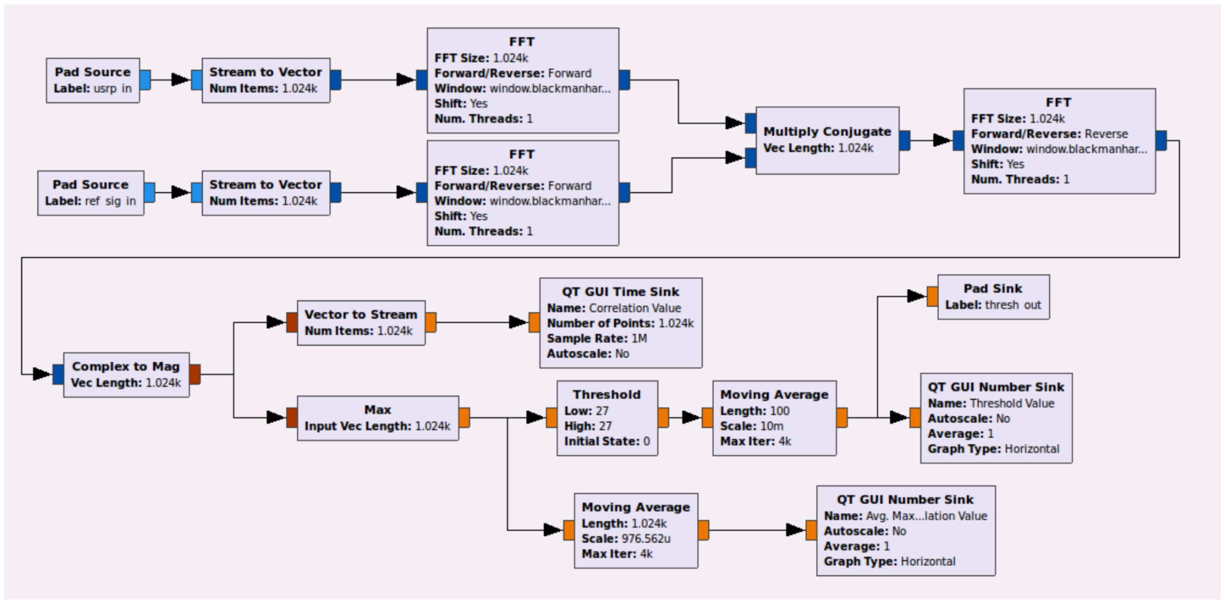
Matched filter detection (MFD) is a method of signal detection in which a received signal of interest is compared to a template of a known signal (e.g., a signal from a PU) and “matched” based on a calculated correlation value representing the “likeness” of the two signals [38]. The higher the correlation value, the more alike the signals are. The MFD technique is a common method for sensing PU signals in a frequency channel of interest, since one of its appealing features is that is works quite well in low SNR environments. However, due to the necessity of having prior knowledge of the primary user’s signal, it is not always the most practical method of detection. In our GNU Radio flowgraph implementation of MFD, the sampled signal of the spectrum channel of interest and the known PU signal are cross-correlated with each other to produce a correlation value, ρ. The value of ρ will be compared to a prescribed threshold value, λ to determine whether or not a PU’s signal is present at a particular time. If $\rho < \lambda$, it is concluded that the PU signal is not present, otherwise it is concluded that the PU signal is present. The equation used to compute the correlation value is given in (3.6);

$$\rho = \max \left\{ \left| \frac{1}{N} \sum_{k=0}^{N-1} \left[X_1(k) X_2(k)^* \right] e^{jkn2\pi/N} \right| \right\}. \tag{3.6}$$

Here, $X_1(k)$ and $X_2(k)$ are the N point DFT vectors of the incoming USRP signal x_1 and the reference PU signal x_2, respectively. $X_1(k)$ is multiplied by the conjugate of $X_2(k)$, and an inverse DFT (IDFT), defined as $\frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{jkn2\pi/N}$, is performed. The magnitude of each complex element is computed, and the maximum value is used as the correlation value.

The flowgraph implementation of the matched filter detector is organized as two hierarchical blocks: one to compute the correlation value, and the other to provide channel switching functionality. The second of these is then incorporated into the GR-MAC program and given a reference PU signal.

Figure 3.12 shows the flowgraph responsible for computing the correlation value of the two signals of interest. The following list describes the flow of data in the MFD correlation flowgraph.


Pad Source
Label: usrp in

Stream to Vector
Num Items: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Pad Source
Label: ref sig in

Stream to Vector
Num Items: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Multiply Conjugate
Vec Length: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Reverse
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Complex to Mag
Vec Length: 1.024k

Vector to Stream
Num Items: 1.024k

QT GUI Time Sink
Name: Correlation Value
Number of Points: 1.024k
Sample Rate: 1M
Autoscale: No

Max
Input Vec Length: 1.024k

Threshold
Low: 27
High: 27
Initial State: 0

Moving Average
Length: 100
Scale: 10m
Max Iter: 4k

Pad Sink
Label: thresh out

QT GUI Number Sink
Name: Threshold Value
Autoscale: No
Average: 1
Graph Type: Horizontal

Moving Average
Length: 1.024k
Scale: 976.562u
Max Iter: 4k

QT GUI Number Sink
Name: Avg. Max...lation Value
Autoscale: No
Average: 1
Graph Type: Horizontal

Figure 3.12: Matched filter detector correlation flowgraph

1. Correlation Calculation

   - Pad sources produce streams of complex valued baseband signals. The topmost source acquires the signal stream from the USRP and, and the lower source acquires a reference PU signal stream.
   - These streams are then sent to a stream-to-vector block where they populate vectors of length 1024.
   - An FFT is then performed on each vector. The length of the FFT is equal to the length of the source vectors, 1024.
   - The vector results of the FFT blocks are then sent to a multiply conjugate block.

Here, the vector from the topmost FFT block is multiplied by the conjugate of the lower FFT block.

- The resulting vector is then sent to a reverse FFT block, which performs an IDFT and brings the values back into the time domain.
- The complex valued vector is sent to a complex-to-mag block where the magnitude of each element is computed. Now, the data is organized as a vector of float values of length 1024.
- Here, the flowgraph branches. In the topmost branch, the vector is converted into a stream and sent to a time sink, which displays the correlation value in the time domain. In the lower branch, the vector of correlation values is sent into a max block which outputs the maximum value of the vector.
- Continuing in the lower branch, the maximum correlation value of each vector is sent to a threshold block. The threshold block compares the incoming maximum correlation value to a prescribed threshold value and outputs a one or a zero depending on whether the maximum correlation value is above or below the threshold. The output of the threshold block is then sent to a moving average block in order to eliminate anomalies from triggering false positives. This average threshold value is then sent to a QT number sink and a pad sink.
- The maximum correlation value is also sent to a moving average block that takes the average over 1024 samples and outputs that to a QT number sink.

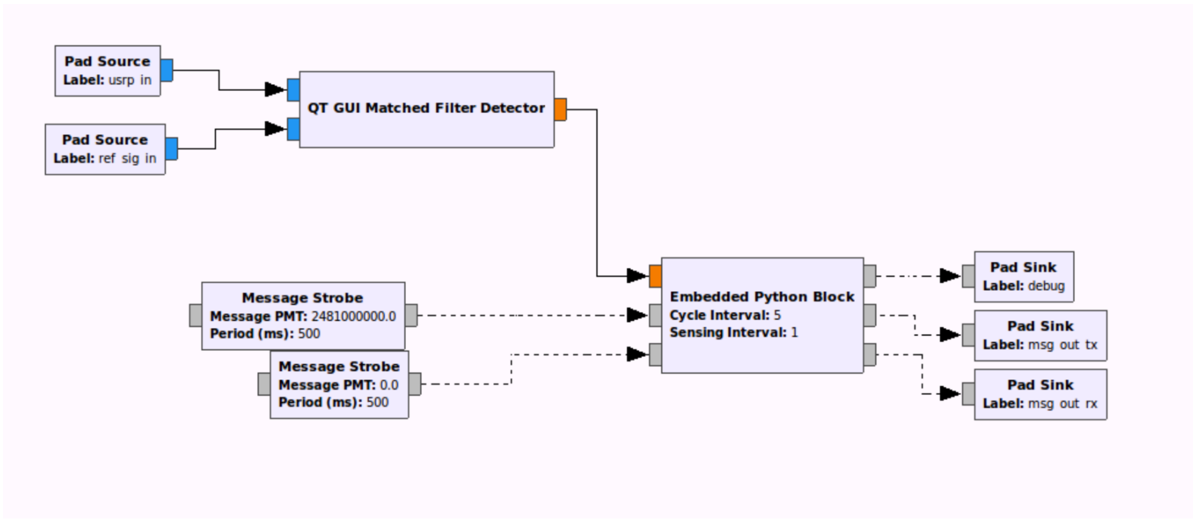

Pad Source
Label: usrp in
Pad Source
Label: ref sig in
QT GUI Matched Filter Detector
Message Strobe
Message PMT: 2481000000.0
Period (ms): 500
Message Strobe
Message PMT: 0.0
Period (ms): 500
Embedded Python Block
Cycle Interval: 5
Sensing Interval: 1
Pad Sink
Label: debug
Pad Sink
Label: msg out tx
Pad Sink
Label: msg out rx

Figure 3.13: Matched filter detector flowgraph with channel switching block

The MFD correlation flowgraph is encapsulated into a hierarchical block, which is then used in a second hierarchical block that adds channel switching functionality. The flowgraph of this second hierarchical block is shown in Figure 3.13. As shown, the MFD correlation block is labeled as "QT GUI Matched Filter Detector," and the resulting threshold value is sent to an embedded Python block that is responsible for channel decision and switching.

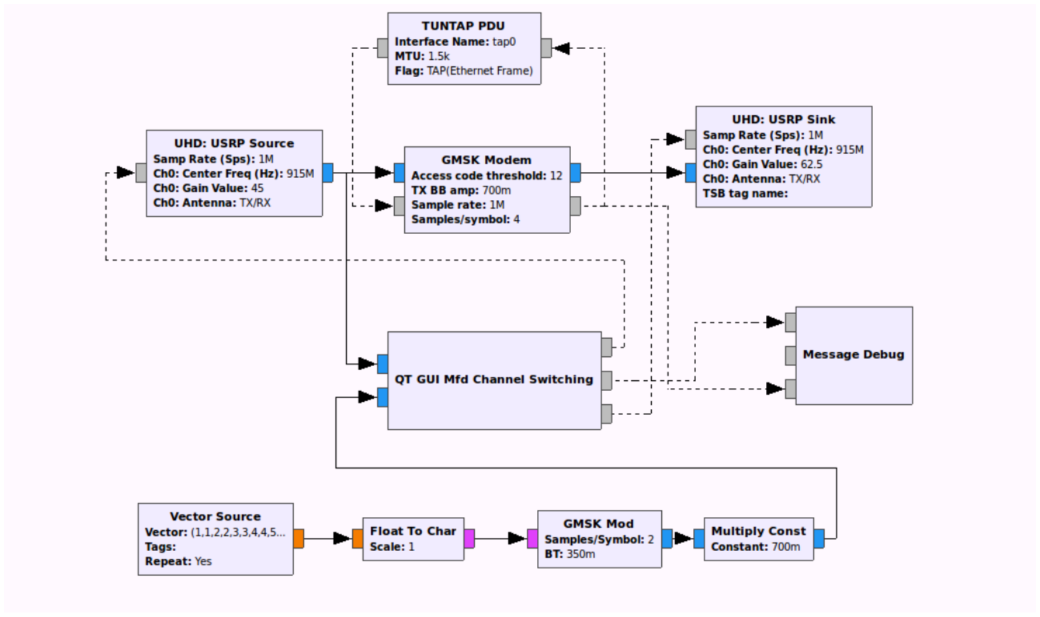

TUNTAP PDU
Interface Name: tap0
MTU: 1.5k
Flag: TAP(Ethernet Frame)
UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 915M
Ch0: Gain Value: 45
Ch0: Antenna: TX/RX
GMSK Modem
Access code threshold: 12
TX BB amp: 700m
Sample rate: 1M
Samples/symbol: 4
UHD: USRP Sink
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 915M
Ch0: Gain Value: 62.5
Ch0: Antenna: TX/RX
TSB tag name:
QT GUI Mfd Channel Switching
Message Debug
Vector Source
Vector: (1,1,2,2,3,3,4,4,5...
Tags:
Repeat: Yes
Float To Char
Scale: 1
GMSK Mod
Samples/Symbol: 2
BT: 350m
Multiply Const
Constant: 700m

Figure 3.14: Matched filter detector hierarchical block in the parent GR-MAC flowgraph

Figure 3.14 shows the GR-MAC flowgraph with the added matched filter detector and channel switching block, labeled “QT GUI Mfd Channel Switching”. As shown, the first of the two signals comes directly from the UHD: USRP source block, and the second is a copy of the PU signal discussed in subsection 3.2.4.

3.4 Physical Setup

The physical setup of the experiment can be seen in Figure 3.15. Each USRP was attached to a laptop PC via USB 3.0, with the PC running experiment-specific programs. In order to cause desired interference to the secondary users when that was the goal of the experiment, USRPs 1 and 3 were used as secondary users, and USRP 2 was used as the primary user.

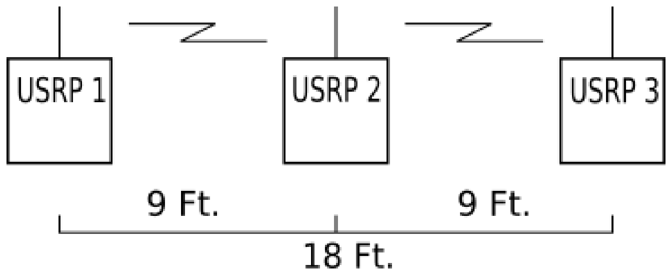


Figure 3.15: Physical setup of USRP systems

3.5 Summary of Experimental Background

In this chapter, an overview of the various aspects of our cognitive radio implementations was given. This overview included the equipment used, both hardware and software, as well as theoretical background information on the two methods of spectrum sensing used: energy detection and matched filter detection. A brief description of the physical setup of our CR systems was also provided. In the following chapter, we will explain the various experiments that were conducted in order to test performance of the CR systems, as well as discuss the results of the experiments.

Chapter 4

Experimental Execution and Results

Using the flowgraphs and the USRPs as previously described, four experiments were performed to test different individual aspects of our cognitive radio, with the final experiment being a series of tests to display the total functionality of the cognitive radios as a whole. The experiments were designed to determine the performance of our CR implementations using metrics of probability of successful detections, p_d, and bit error rate (BER). The experiments were designed to to complete the following objectives.

1. Find the method of spectrum sensing with the higher value of p_d,

   i.e., $\arg\max_{p_d}\{p_{d,\mathrm{MFD}}, p_{d,\mathrm{ED}}\}$.

2. Determine the minimum sensing interval such that p_d is maximum,

   i.e., $I_{s0} = \min_{I_{s0}}\{\arg\max\{p_d(I_{s0})\}\}$.

3. Determine the minimum cycle interval such that BER is minimum,

   i.e., $I_c = \arg\min_{I_c}\{\mathrm{BER}(I_C)\}$.

4. Determine the performance of our CR implementations in terms of BER in the presence of a PU signal as varying SNRs.

4.1 Discussion of Units, Measurements, and Thresholds

This section discusses some important considerations about the units and measurements that were used during the tests. These details are considered before the experiments are discussed because understanding the results of the experiments relies upon them. In the first part of this section, the process of acquiring units of decibels is discussed. In the second part, a short description of the threshold values and how they were determined is given.

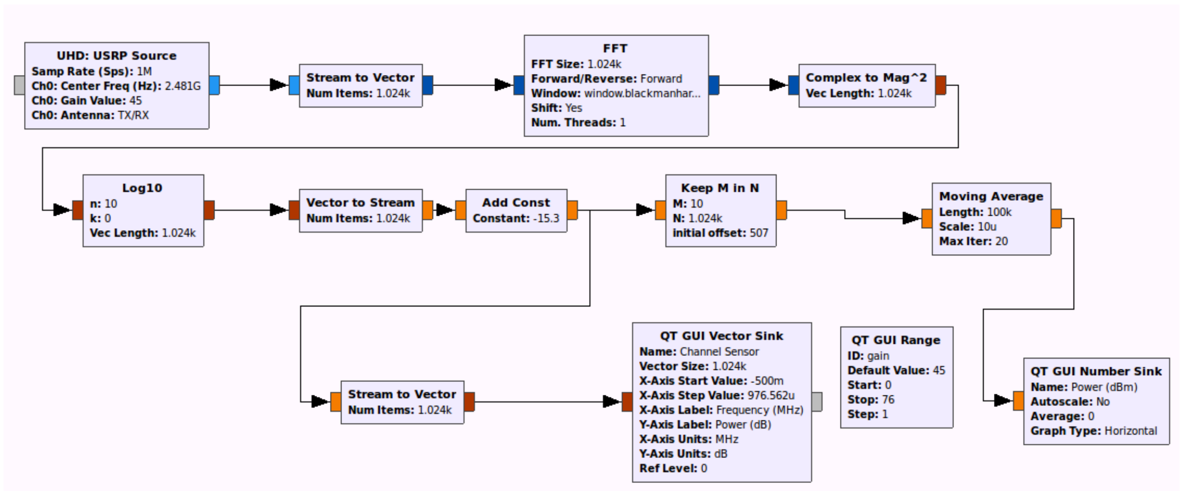
4.1.1 Units and Measurements

This section discusses the units that were used throughout the test. In an ideal cognitive radio experimental setup, the measurements would be recorded and displayed in units of power, namely decibel-milliwatts. By default however, GNU Radio provides relative power in units of decibels. Though decibels are not a measurement of absolute power, they allow relevant calculations for signal-to-noise ratios to be made. During our testing, we attempted to acquire absolute units of decibel-milliwatts through manual calibration of the USRPs, however, we eventually decided to use relative units of decibels. This section discusses the process that led up to that decision.

Within the UHD USRP block provided in GNU Radio, the transmitter and receiver gains are settable by the user. While the values for transmitter gain are in units of decibels, the absolute power emitted from a USRP can be determined by using a power meter. In our testing attempt to aquire absolute units of power, the transmitter port on the USRP was connected to an Agilent P-Seriers power meter via an SMA cable. The GNU Radio flowgraph which simulated a primary user, as shown in subsection 3.2.4, was run with the transmitter gain first set to the minimum value of 0dB. The reading on the power meter was recorded. Then, the transmitter gain was increased by 1 dB increments, up to its maximum of 86 dB, and the power meter reading was recorded at every increment.

Our calibration method required using a loopback cable to create a direct connection between a USRP's transmitter port and another USRP's receiver port. However, since the maximum transmitter power is much higher than the maximum recommended input power, a 30 dB attenuator must be used on the loopback cable. Because of the need for an attenuator in our USRP calibration tests, the previous process of connecting the USRP transmitter to the Agilent power meter and recording the transmitter power at increasing gain values was replicated with the 30dB attenuator attached. This was done to verify that the attenuator would indeed decrease our recorded power values by 30 dB.

This process gave us the expected value for absolute output power of the transmitter for every value of its gain. The next step in the calibration required using two USRPs, one to transmit and the other to measure the power. On one of the USRPs, the primary user flowgraph was run. A very simple flowgraph was created to act as a GNU Radio power meter, and it is shown in Figure 4.1. The purpose of this flowgraph is to act similarly to the Agilent power meter, but using the USRP. The operation of this flowgraph is similar to the energy detector flowgraph, which is described block-by-block in section 3.3.1. This flowgraph



UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 2.481G
Ch0: Gain Value: 45
Ch0: Antenna: TX/RX

Stream to Vector
Num Items: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Complex to Mag^2
Vec Length: 1.024k

Log10
n: 10
k: 0
Vec Length: 1.024k

Vector to Stream
Num Items: 1.024k

Add Const
Constant: -15.3

Keep M in N
M: 10
N: 1.024k
initial offset: 507

Moving Average
Length: 100k
Scale: 10u
Max Iter: 20

Stream to Vector
Num Items: 1.024k

QT GUI Vector Sink
Name: Channel Sensor
Vector Size: 1.024k
X-Axis Start Value: -500m
X-Axis Step Value: 976.562u
X-Axis Label: Frequency (MHz)
Y-Axis Label: Power (dB)
X-Axis Units: MHz
Y-Axis Units: dB
Ref Level: 0

QT GUI Range
ID: gain
Default Value: 45
Start: 0
Stop: 76
Step: 1

QT GUI Number Sink
Name: Power (dBm)
Autoscale: No
Average: 0
Graph Type: Horizontal

Figure 4.1: Power meter flowgraph

contains a number sink which outputs the power observed in the middle frequency bin. This number shown by the number sink is analogous to the reading shown on the display of the Agilent power meter. An offset block is added before the number sink, which is configurable by the user during runtime. The offset block is used to match the value of the number sink with the expected value observed by the power meter at a certain transmitter gain. This flowgraph was run on the second USRP, and the two USRPs were attached with a loopback cable, buffered by a 30dB attenuator.

This setup allowed us to obtain an offset value based on the differences between the readings of the Agilent power meter and the power meter flowgraph, at any given transmitter gain. For example, the power which was recorded from the Agilent power meter when the transmitter gain was set to 70 dB was -4.0 dBm. Attaching the transmitter to the USRP receiver and running the power meter flowgraph would result in a power reading of 15.9 dBm. This is 19.9 dBm higher than the expected -4.0 dBm value, so the offset in the flowgraph was set to -19.9 dBm. This offset was confirmed by checking several different values above the noise floor.

Eventually, several issues were discovered regarding this method of power measurement. The default receiver gain for the GR-MAC flowgraph that was described in section 3.2.1 is 45 dB. However, when the power meter was run with its gain set to 45 dB and our offset of -19.9 dBm was used, the power readings would gradually hit a ceiling at high values of the transmitter gain. We would expect that, in power values that were above the noise floor, the readings from the power meter flowgraph would follow a 1:1 ratio with the transmitter gain. That is, for every 1 dB increase in gain, we would expect a 1 dBm increase in absolute power, as was found by the readings taken from the Agilent power meter. However, when the receiver gain on the power meter flowgraph was set to 45 dB, the 1:1 ratio would not hold true at high values for the transmitter gain. As the transmitter gain was increased by 1 dB, the increase in power from the flowgraph readings would diminish to below 1 dB.

Therefore, the receiver gain was set to 0 dB, and we attempted to find the correct offset value again. The result was the elimination of the power ceiling, and the expected results from the power meter received. However, it was found that the operation of the GR-MAC flowgraph required a gain of 45 dB in order to provide data communication. This made it impossible to obtain correct power readings on the same flowgraphs that would allow for GR-MAC functionality.

Additionally, when the power meter flowgraph was run with 0 dB receiver gain with an antenna instead of with a loopback cable, the power readings were only slightly above the noise floor, even at the high transmitter gain values. This fact, in addition to the nonlinear power readings at high gains, led us to abandon the effort of finding absolute power in units of dBm. Instead, as is seen in the results section of this paper, the units used are dB rather than dBm.

4.1.2 Thresholds

In both the matched filter detector and energy detector, the final decision of whether a channel is occupied or not is made by the channel switcher embedded Python block. As discussed in subsection 3.2.2, the input to the Python block is a one if a threshold from the previous block is met, or a zero if it is not. The decision made by the Python block is based on an average of these boolean values, and the meaning behind the threshold is now discussed.

In the matched filter detector, the maximum correlation value is compared to a threshold, and in the energy detector, the energy at a specified frequency is compared to a threshold. After having used our cognitive radio system in different physical spaces, and at different times, we noticed that the radio's sensing decisions were highly sensitive to the threshold settings. Therefore the thresholds were chosen to be static, meaning they would not change during the course of a trial. Before each trial, the threshold values were adjusted such that the average threshold value would not exceed the arbitrarily chosen value of 0.1 when the primary user was not present, but would exceed that value when it was.

4.2 Experiments

Using the flowgraphs and USRPs as described in the previous chapter, four experiments were performed. These experiments serve to display the performance and functionality of our simple cognitive radio systems. The goal of these experiments was to determine the probability of successful detection of both sensing methods, and what the ideal values used for our adjustable parameters should be to create a cognitive radio system which provides the most robustness.

The probability of successful detection for both sensing methods is the subject of the first experiment. In the second and third experiments, the ideal sensing interval and cycle intervals are determined using the metric of bit error rate derived from packet error rate. The fourth and final experiment serves as a high-level view of our cognitive radio system performance using bit error rate at varying PU SNRs.

4.2.1 Experiment One: Sensing Method Comparison

The goal of the first experiment was to determine which of the spectrum sensing methods would provide a higher probability of successful PU detection at lower SNRs. In this experiment, one USRP acted as the primary user, and the other two USRPs acted as secondary users and used the two channel sensing methods to detect the presence of primary user. In order to subject both of the channel sensing methods to the same conditions, the two secondary user USRPs were placed close to each other at the same distance away from the primary user. In order to rapidly record the outcomes of large quantities of detections in a reasonable amount of time, we used the modified energy detector and matched filter detector flowgraphs and implemented a Python driver for the tests. These are described in subsection 3.2.3.

The SNR of the primary user was recorded by using the power meter flowgraph on the same USRP which would later run one of the two sensing methods. First, the average power of the noise was recorded for several seconds. Then, the primary user was turned on and the power of the signal was recorded. The power value of the noise was subtracted from the power value of the signal to obtain the SNR. As discussed in section 4.1, the units of these readings are dB.

Once the SNR of the primary user was recorded, the modified flowgraphs were run by the Python test driver. In the modified flowgraphs, Python code is used to output the results of a channel detection to a file. After a trial is over, this file is automatically read by the test driver that started the trial, and the number of times that the channel was decidedly occupied is counted. For both detection methods, 10,000 channel detections were attempted in every trial. The number of trials was chosen arbitrarily.

It is important to emphasize that the independent variable in this experiment is the SNR of the primary user, not the transmitter gain. The dependent variable, then, is the number of times an occupied channel was successfully detected. Because the outcome can only be one of two possibilities, either the PU signal was detected or not, the event of successful detection can be defined as a Bernoulli random variable P_{detect} with probability p. Let n_s be the number of successful Bernoulli trials and n be the total number of Bernoulli trials. Then, p can be calculated as shown in (4.1). As n goes to infinity, the true value of p is obtained. Therefore, a sufficiently large value of 10,000 trials was conducted.

$$p = \frac{n_s}{n}. \tag{4.1}$$

The transmitter gain of the PU was first set to its maximum value for which the SNR at the power meter was observed to be 0 dB and no detections could occur. This was found to be 42 dB. Thus, at a transmitter gain of 41 dB or lower, the signal was found to be under the noise floor. Then, the transmitter gain was increased and the SNR was recorded by the power meter. After the SNR was acquired, the modified matched filter and energy detector flowgraphs were executed to collect the results of 10,000 detection attempts. The tests were repeated and the transmitter gain was increased until both the energy detector and the matched filter detector were deemed to be consistently successful. As shown by the results section, at suitably high SNRs, both channel sensing methods achieved 100% detection in some trials.

4.2.2 Experiment Two: Finding Ideal Sensing Interval

The goal of this experiment was to acquire the minimum sensing interval for which both spectrum sensing methods would work reliably. Ideally, this interval should be as short as possible so that more time can be allocated for the TX/RX phase of the cycle. However, it still needs to be long enough so each method of spectrum sensing will maintain the previously determined probability of successful detection.

During the sensing interval, an average is maintained within the flowgraphs. This average is the result of boolean inputs, whose value determines if a user was detected in the observed channel or not. A value of one means that a user was detected, and a value of zero means that the observed channel was absent, thus, the average value lies between zero and one. If the average is above 0.5, this means that a user has been detected in the target channel. This process and its significance is discussed in more depth in subsection 3.2.2. If the channel checking is executed with no averaging function in the flowgraph, the results are highly vulnerable to anomalies. So, the averaging function is necessary to acquire reliable data.

The sensing interval is the interval over which the average is maintained. Initially, one would assume the ideal sensing interval would be as large as possible, because this would allow a long average value to be maintained, allowing the CR to provide a very accurate decision on whether the channel is occupied or not. However, TX/RX is impossible during the sensing interval. To maximize the allotted time for TX/RX, the ideal sensing interval should actually be the smallest interval possible. Since very low sensing intervals are vulnerable to anomalies, this test was designed to find the minimum sensing interval which would maintain the probability of successful detection for each sensing method found in the previous experiment.

In this experiment, one USRP was used as the primary user and the other two were used as the secondary users. The primary user was set to its maximum gain value. The expected behavior, then, was for the channel sensors to detect an occupied channel 100% of the time. However, anomalies and insufficient values for successful averaging would cause the channel sensors to mistakenly determine a vacant channel. The first sensing interval was chosen to be an arbitrarily low number, so that the large amount of mistakes in averaging and anomalies would cause the number of successful detections to be very low for both of the channel sensing methods. Then, the sensing intervals were incrementally increased until both detectors could detect with high accuracy.

The same Python test driver used in the first experiment was used for this experiment, and, as in the first experiment, both detection methods were run over 10,000 trials to correctly identify whether the channel was occupied or not. Again, the probability of detection p was calculated using (4.1) and was used as a performance metric to compare the outcome to the sensing interval.

4.2.3 Experiment Three: Finding Ideal Cycle Interval

The objective of the third experiment was to determine the minimum cycle interval in which the bit error rate between two secondary users would be sufficiently low. In the previous experiment, the minimum sensing interval was found, and, as discussed in section 3.2.2, the sensing interval should be considered as a portion of the cycle interval. The mathematical difference between the cycle interval and the sensing interval is the TX/RX interval, during which data is allowed to be traded. Therefore, the ideal cycle interval can be determined by measuring the packet error rate at varying cycle times, and calculating bit error rate from packet error rate.

The cycle interval also determines the period between consecutive channel sensing steps. In a system whose sole purpose is to avoid an occupied channel and does not have to trade any data, the optimal cycle interval would be as small as possible. This is because, at a low cycle interval, the channel sensing step could occur as frequently as possible, thus decreasing time spent in a channel once it becomes occupied. Conversely, in a system whose sole purpose is to trade data and is indifferent to the occupancy of a channel, the ideal cycling interval would be as large as possible. This is because data would be traded during the entire interval. Therefore, a balance must be found where data loss from the channel sensing interval is not too high, but the CR system is still able to sense the spectrum frequently enough to avoid the presence of a primary user in the channel.

By design, the channel switching functionality in the flowgraphs is independent of the channel sensing method. The Python block that allows channel switching to occur in the energy detector also does so in the matched filter detector. Additionally, the detection of a primary user is irrelevant for this test because a primary user is detected during the sensing interval, which is not a focus of this experiment. Thus, no primary user was simulated during this experiment, and only the energy detector was used.

Two secondary users were configured with the energy detector flowgraphs. One of the secondary users acted as a transmitter while the other acted as a receiver. They both used the Python programs discussed in section 3.2.5 to generate and send a one-way data stream of packets. For each of the following trials, the Python programs allowed one secondary user, called SU-A to send 10,000 packets of total length 1,042 to the other secondary user, called SU-B. The packets were counted by the receiver, and the result of this count was recorded and used to find bit error rate for a one-way data stream between the two secondary users. To calculate bit error rate, each bit in a packet was assumed to be an error with probability p_e independent of all other bits in the packet. Thus, the process of packet error can be described as a Bernoulli process, since each of the 1042 bits in a packet has a probability of error p_e. The probability of packet error p_p can be calculated using (4.2), where N is the number of bits in the packet;

$$p_p = 1 - (1 - p_e)^N. \tag{4.2}$$

Each packet was sent using UDP as the transport layer protocol, and UDP will drop a packet if it is corrupt, i.e., a bit is in error. Therefore, we can assume any packet not correctly received had at least one bit in error. In this experiment, we observed the number of packets not received out of 10,000 to calculate p_p. Solving (4.2) for p_e results in (4.3), which was used to calculate bit error rate;

$$p_e = 1 - \sqrt[N]{1 - p_p}\ . \tag{4.3}$$

In order to execute this experiment, we first needed to acquire an ideal bit error rate. This would be the bit error rate during a trial in which channel switching never takes place so that TX/RX is possible all of the time. The two secondary users were initialized by first running the energy detector flowgraph which incorporated the MAC-layer interface, discussed in section 3.2.1. Then, they were assigned IP addresses using the *ifconfig* command.

```
# sudo ifconfig tap0 192.168.200.1
```

To ensure that the secondary users were connected correctly and were able to communicate, we used the ping command before continuing on to the test.

```
# ping 192.168.200.2
```

After the two secondary users were initialized, the Python transmitter program was run on PU-A, and the receiver program was run on PU-B. The number of packets received by PU-B was recorded and used to calculate the number of packets not received. The result of the first trial served as an ideal scenario for the remainder of the experiment. Because no channel switching occurred, the first trial contained the smallest packet loss.

Subsequent trials were executed similarly, but with channel switching enabled with varying cycle intervals. The sensing interval used for these experiments was the ideal sensing interval found during the second experiment. This is discussed in subsection 4.3.2 in more depth, but, in summary, 60 ms was found to be a sufficient detection interval for both channel sensing methods. The first cycle interval tested was set just above the sensing interval, and resulted in a sort of zero-point for this experiment, for which the TX/RX interval was so small that no data could be traded. The experiment was run for every cycle interval from 0.09 seconds to 0.39 seconds, in 0.03-second increments.

4.2.4 Experiment Four: Display of Cognitive Radio Functionality

Using the data we had acquired from the previous experiments and the experience we had gained from setting them up, our final experiment was designed to demonstrate each aspect of our CR system working together. These tests were not meant to be as rigorous or specific as the previous ones. Instead, they were meant to display a summary of how our cognitive radio implementations work. The most significant and important feature of our cognitive radio system is that the secondary users are able to detect when a primary user arrives in their current channel and simultaneously switch to an unoccupied channel while maintaining minimal bit error rate. This experiment aims to demonstrate this functionality in action.

For these tests, one primary user and two secondary users were set up in the much the same way as in previous experiments. The two secondary users were first initialized, using either the matched filter detector implementation or the energy detector implementation. Then, the *ifconfig* function was used to assign the secondary users network IP addresses, as had been done for the second experiment. After using the *ping* function to ensure that each secondary user could communicate with the other, the channel switching feature on each was enabled, and the Python program which counts the number of received packets was started. The primary user began transmitting to cause interference at the same time that the Python transmitter program was started. The result of this was that the secondary users began their one-way communication in an occupied channel. During their channel sensing intervals, the secondary users checked for the presence of the primary user, and if the PU's signal had a large enough SNR to where the it could be detected, then the secondary users switched channels and maintained the one-way data stream.

Tests in this experiment were run for the energy detector and the matched filter detector at varying PU SNRs.

4.3 Results

Our main interest in this thesis was to provide a comparison between the energy detection and matched filter detection methods, and use them to design a basic cognitive radio system which can detect whether a primary user is present in its desired channel for communication or not. We were able to achieve this goal by executing the experiments in the previous section. Additionally, we were able to begin to investigate the processes involved in channel switching and secondary user synchronization. This section discusses the results of our experiments and begins to address their shortcomings.

4.3.1 Experiment One: Sensing Method Comparison

The first experiment was the most significant focus of our project. This experiment investigated the sensitivity of the energy detector and matched filter models to a primary user at a low SNR, using the probability of successful detection as a performance metric. As the SNR of the primary user was increased from 0 dB, the number of successful detections for both of the detectors was recorded, out of 10,000, for each trial. Figure 4.2 shows a graph

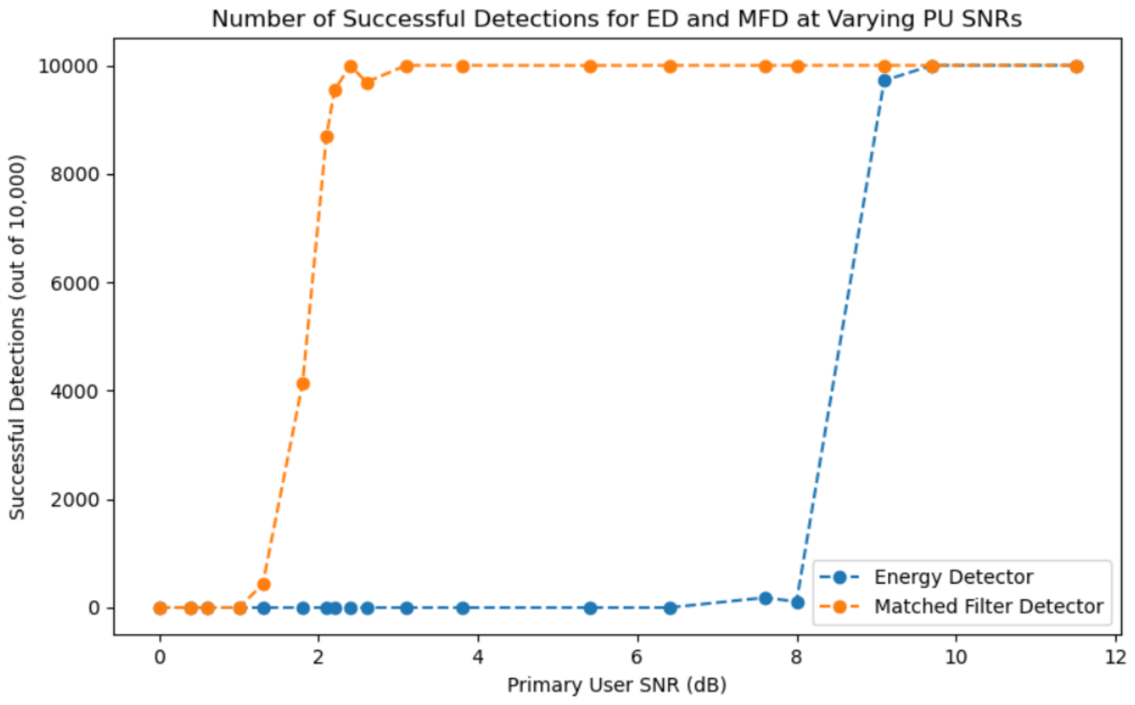


Figure 4.2: Comparison of matched filter detector and energy detector performance based on the number of successful detections out of 10,000 detection attempts for varying primary user signal-to-noise ratios

comparing the performances of these methods.

As shown, the matched filter detector begins detecting the presence of the primary user reliably at an SNR roughly 6.4 dB lower than the energy detector. Both detection methods jump from virtually 0 detections to nearly the perfect 10,000 detections relatively quickly as primary user SNR is increased. For example, at 1 dB SNR, the matched filter failed to detect the signal in all trials, but it had nearly 10,000 successful detections when the primary user's SNR was increased to 2.6 dB, a mere 1.6 dB increase. The window in which the detectors perform imperfectly, but above 0, may seem insignificant, but its implications are important. To remain synchronized, our cognitive radio system relies on two secondary users making the same decision at the same time, but both performing individual analyses. If one secondary user finds the channel to be occupied, but the other does not, one secondary user will switch channels while the other does not. Obviously, this is detrimental to our system performance. However, the window where this may occur is small, and, as SNR is increased, far more consistent performance is observed.

The matched filter detector performed with an obvious and vast superiority to the energy detector. The lowest SNR for which the matched filter's probability of successful detection was 1 was 3.1 dB, while the lowest SNR for consistently successful detections for the energy detector was 9.7 dB. This means that the matched filter detector was more reliable than the energy detector by a margin of 8.6 dB SNR for the primary user signal.

Therefore, the matched filter detector provided a much higher sensitivity to a low-SNR primary user in our experiments. This is consistent with our findings in the literature, where we found the main advantage of matched filter detection to be its higher sensitivity to primary user signals in low SNR environements.

4.3.2 Experiment Two: Finding Ideal Sensing Interval

In the second experiment, we attempted to find the minimum sensing interval which would maintain the performance of the channel sensing implementation. For this test, a primary user at maximum gain was sensed by a secondary user 10,000 times, at varying sensing intervals, beginning at 0.01 seconds. Both detection methods performed poorly, and the interval was incremented for the subsequent tests until both methods were consistently successful. Figure 4.3 displays the results of this experiment by comparing the energy detector and matched filter detector at different sensing intervals.

The result of this experiment reveals a strange characteristic of the detection methods.

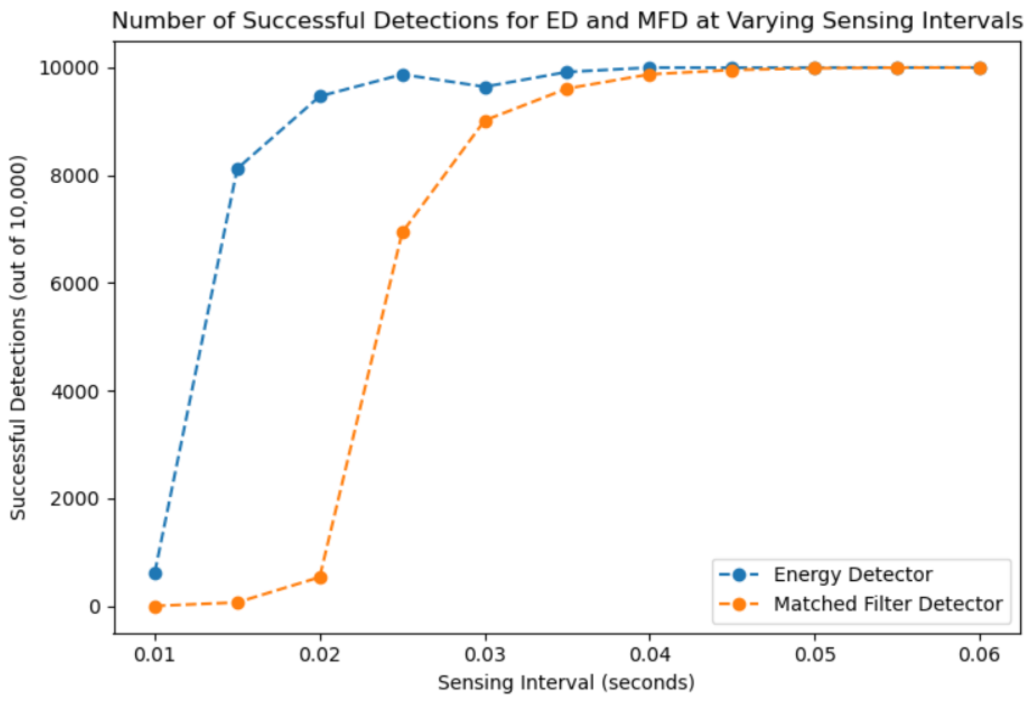


Figure 4.3: Successful detections out of 10,000 detection attempts for matched filter detector and energy detector for varying spectrum sensing intervals

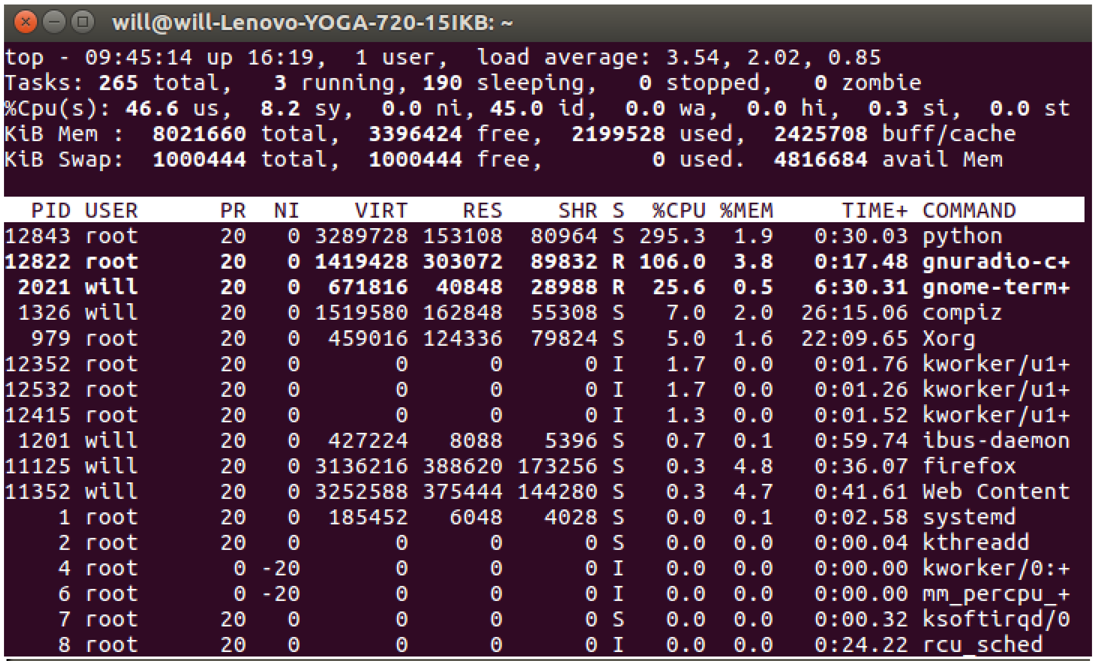
Before running this experiment, we expected that the energy detector and the matched filter detector to behave similarly despite the sensing interval. However, the energy detector was able to accurately make a detection 10,000 times as a sensing interval of 0.04 seconds, whereas the matched filter did not achieve a probability of successful detections of one until the sensing interval was set to 60 ms.

Though we did not provide a further experimental investigation in this thesis, which would have been beyond the scope of this study, the result of this experiment allows us to form a conjecture about the cause of the disparity. It is known from the literature that the matched filter detector is a far more computationally complex method of signal detection than the energy detector. While the energy detector only requires one FFT and a few more relatively simple mathematical functions to be performed on each set of data, the matched filter detector requires several FFTs, as well as the generation of a sample signal. The difference in computational complexity and its effect on the flowgraphs should be investigated in great depth in order to draw any real conclusions. However, a basic understanding can be achieved by running the *top* command on the computer while it is executing each flowgraph. The results of *top* while the energy detector and matched filter are executing can be seen in Figure 4.4.

In the first line under the white bar in the output of *top*, details about the resource usage of the currently running flowgraph, (denoted as “python”), can be seen. One obvious difference between performance breakdown of the two flowgraphs is that, for the energy detector, 295.3% of the available processing power is being used, while for the matched filter detector, 533.6% is being used. Since the Lenovo Yoga laptop that this *top* had been run on has 8 virtual cores, these values should be read as parts of 800%.

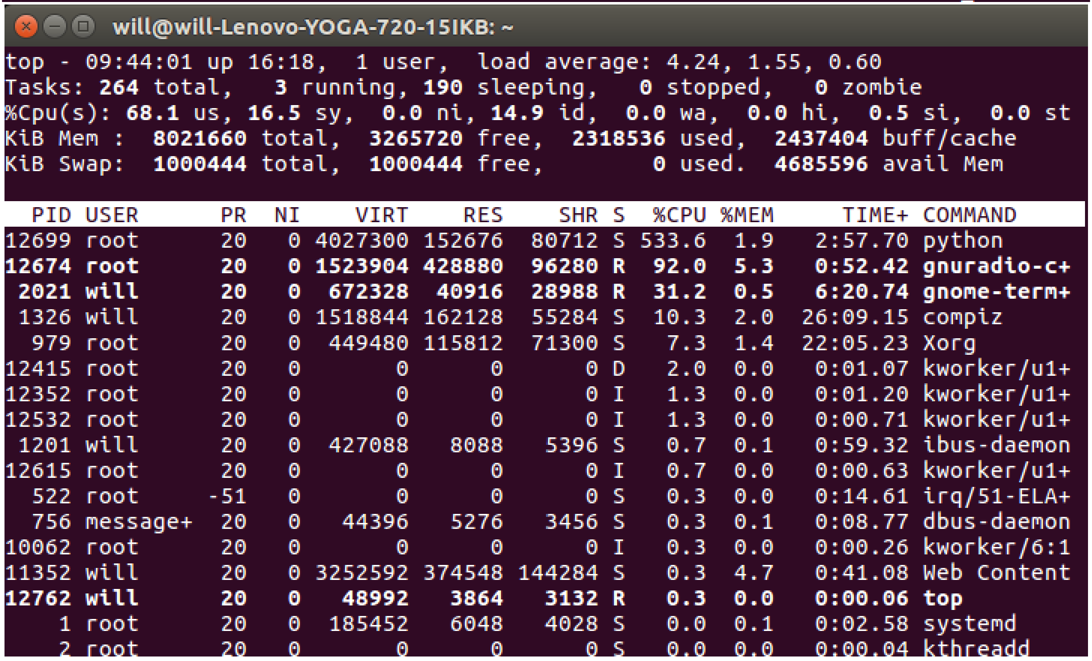
Though the higher resource usage of the matched filter detector is not a definitive reason for the higher required sensing interval, the GNU Radio flowgraphs do require a certain level of performance. It became obvious to us early on in our experience using GNU Radio that seemingly insignificant things, such as a difference in computational intensity, can drastically change the results of the simulations.

Despite the difference between the two flowgraphs, the 60 ms sensing interval that both of the methods were able to use was far smaller than we had originally expected it would be based on previous experimentation. For the third experiment, 60 ms was used as the sensing interval because both of the methods could operate with this interval, and we wanted to maintain consistency.



```
top - 09:45:14 up 16:19,  1 user,  load average: 3.54, 2.02, 0.85
Tasks: 265 total,   3 running, 190 sleeping,   0 stopped,   0 zombie
%Cpu(s): 46.6 us,  8.2 sy,  0.0 ni, 45.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem :  8021660 total,  3396424 free,  2199528 used,  2425708 buff/cache
KiB Swap:  1000444 total,  1000444 free,        0 used.  4816684 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
12843 root      20   0 3289728 153108  80964 S 295.3  1.9   0:30.03 python
12822 root      20   0 1419428 303072  89832 R 106.0  3.8   0:17.48 gnuradio-c+
 2021 will      20   0  671816  40848  28988 R  25.6  0.5   6:30.31 gnome-term+
 1326 will      20   0 1519580 162848  55308 S   7.0  2.0  26:15.06 compiz
  979 root      20   0  459016 124336  79824 S   5.0  1.6  22:09.65 Xorg
12352 root      20   0       0      0      0 I   1.7  0.0   0:01.76 kworker/u1+
12532 root      20   0       0      0      0 I   1.7  0.0   0:01.26 kworker/u1+
12415 root      20   0       0      0      0 I   1.3  0.0   0:01.52 kworker/u1+
 1201 will      20   0  427224   8088   5396 S   0.7  0.1   0:59.74 ibus-daemon
11125 will      20   0 3136216 388620 173256 S   0.3  4.8   0:36.07 firefox
11352 will      20   0 3252588 375444 144280 S   0.3  4.7   0:41.61 Web Content
    1 root      20   0  185452   6048   4028 S   0.0  0.1   0:02.58 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.04 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:+
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_+
    7 root      20   0       0      0      0 S   0.0  0.0   0:00.32 ksoftirqd/0
    8 root      20   0       0      0      0 I   0.0  0.0   0:24.22 rcu_sched
```



```
top - 09:44:01 up 16:18,  1 user,  load average: 4.24, 1.55, 0.60
Tasks: 264 total,   3 running, 190 sleeping,   0 stopped,   0 zombie
%Cpu(s): 68.1 us, 16.5 sy,  0.0 ni, 14.9 id,  0.0 wa,  0.0 hi,  0.5 si,  0.0 st
KiB Mem :  8021660 total,  3265720 free,  2318536 used,  2437404 buff/cache
KiB Swap:  1000444 total,  1000444 free,        0 used.  4685596 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
12699 root      20   0 4027300 152676  80712 S 533.6  1.9   2:57.70 python
12674 root      20   0 1523904 428880  96280 R  92.0  5.3   0:52.42 gnuradio-c+
 2021 will      20   0  672328  40916  28988 R  31.2  0.5   6:20.74 gnome-term+
 1326 will      20   0 1518844 162128  55284 S  10.3  2.0  26:09.15 compiz
  979 root      20   0  449480 115812  71300 S   7.3  1.4  22:05.23 Xorg
12415 root      20   0       0      0      0 D   2.0  0.0   0:01.07 kworker/u1+
12352 root      20   0       0      0      0 I   1.3  0.0   0:01.20 kworker/u1+
12532 root      20   0       0      0      0 I   1.3  0.0   0:00.71 kworker/u1+
 1201 will      20   0  427088   8088   5396 S   0.7  0.1   0:59.32 ibus-daemon
12615 root      20   0       0      0      0 I   0.7  0.0   0:00.63 kworker/u1+
  522 root     -51   0       0      0      0 S   0.3  0.0   0:14.61 irq/51-ELA+
  756 message+  20   0   44396   5276   3456 S   0.3  0.1   0:08.77 dbus-daemon
10062 root      20   0       0      0      0 I   0.3  0.0   0:00.26 kworker/6:1
11352 will      20   0 3252592 374548 144284 S   0.3  4.7   0:41.08 Web Content
12762 will      20   0   48992   3864   3132 R   0.3  0.0   0:00.06 top
    1 root      20   0  185452   6048   4028 S   0.0  0.1   0:02.58 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.04 kthreadd
```

Figure 4.4: The results of *top* for the energy detector (top) and matched filter detector (bottom)

4.3.3 Experiment Three: Finding Ideal Cycle Interval

While we found the minimum sensing interval in the previous experiment, the operation of the secondary users requires a balance between channel sensing and TX/RX. The goal of the third experiment was to find the cycle interval which would provide an acceptable balance, such that channel sensing could occur, while maintaining an acceptable bit error rate. For these tests, bit error rate was calculated by first estimating the packet error rate by subtracting the number of packets received from the number of packets sent in a one-way communication and then using (4.3). Because the intervals in the cycle are independent of the channel sensing method, only the energy detector was used for this test. However, this is later discussed as a shortcoming in chapter 5. Table 4.1 quantifies the results of this experiment, and they are graphed in Figure 4.5.

Table 4.1: Bit error rate for energy detector with network functionality for increasing cycle intervals

Cycle Interval *(s)*	**Calculated BER**
0.09	1
0.12	3.22e-4
0.15	1.68e-4
0.18	2.41e-4
0.21	1.65e-4
0.24	1.12e-4
0.27	6.01e-5
0.30	8.15e-5
0.33	8.38e-5
0.36	8.57e-5
0.39	5.75e-5
∞ (ideal)	5.01e-5

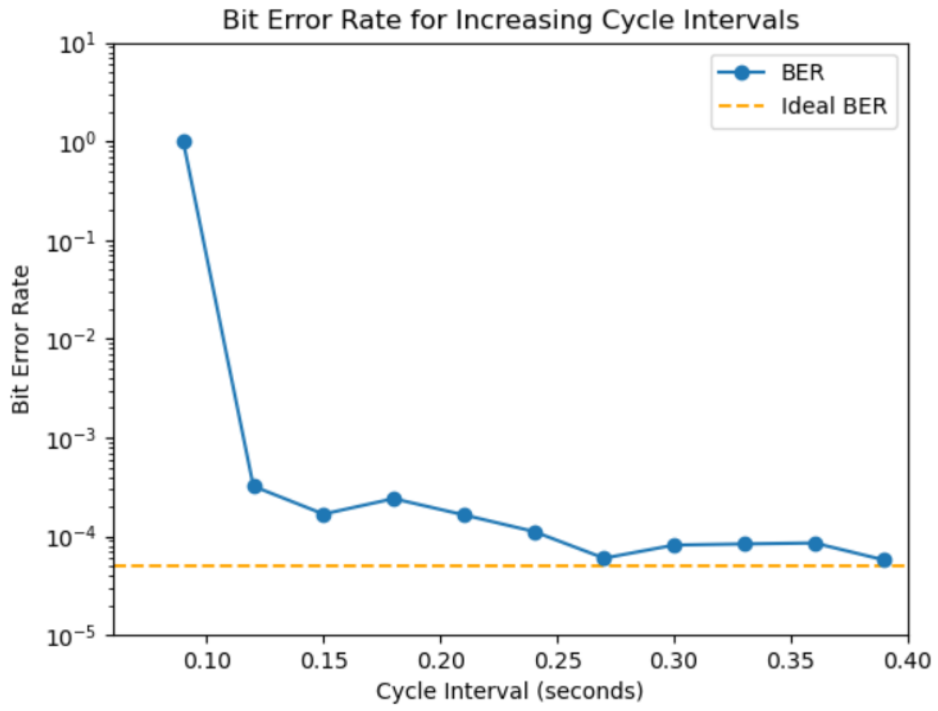


Figure 4.5: Bit error rate for energy detector with network functionality for increasing cycle intervals

At a cycle interval of 90 ms, the bit error rate was 1. This cycle interval, then, was the greatest cycle interval where a ”worst-case scenario” was observed. A drastic improvement in bit error rate was made by increasing the cycle interval only slightly, and it continued to improve gradually as the cycle interval continued to increase. In Figure 4.5, the bit error rate can be seen approaching the ideal bit error rate as the cycle interval increases, where the ideal bit error rate was calculated by counting the number of packets received when no channel sensing step occurred.

The increase of the cycle interval allows the TX/RX interval to be increased, so less time is spent losing data. At a cycle interval of 390 ms, the bit error rate was found to be suitably close to the ideal bit error rate.

4.3.4 Experiment Four: Display of Cognitive Radio Functionality

The goal of the final suite of tests was to display the total functionality of our cognitive radio system. For these tests, the primary user was activated so that the secondary users would switch channels during their data communication. The execution of the fourth experiment contained the most shortcomings of the four because of mistakes in programming and misunderstanding the data. This subsection introduces those shortcomings, which are elaborated in the conclusion.

Initially, this test was attempted using the ideal sensing and cycle intervals determined by the second and third experiments. Though these values were decided experimentally and individually, they were found to be insufficiently small when applied in the context of the rest of the cognitive radio system. To the best of our abilities, we had synchronized the clocks of our PC systems so that the secondary users would not encounter interference from one user transmitting while the other was in their channel sensing step. For example, if the sensing interval for the secondary users was 60 ms, it is imperative that the system clocks for the secondary users must be synchronized to a level of certainty such that the two clocks would not contain an offset close to 60 ms.

The reason for this can be realized by studying Table 3.1, which gives an example for a possible channel switching scenario. In this scenario, the sensing interval is 0.2 seconds. It can be seen that, during this interval, TX/RX stops so that the secondary users won't transmit any data. However, if there is an offset in the clocks of the two systems, one secondary user will continue to transmit data during this period. This could cause the other secondary user to falsely conclude that the channel is occupied by a primary user, and switch channels. In our system, this would put the secondary users helplessly on different channels with no way of achieving synchronicity.

In practice, it was found that this problem could be eliminated by simply increasing the sensing and cycle intervals because a large sensing interval can act as a buffer in which no interference from the opposite secondary user would be received. Therefore, in the fourth experiment, we set the sensing interval for the secondary users to 200 ms instead of the ideal 60 ms, and we set the cycle interval to 1 second instead of the ideal 390 ms. Though the values that were used were not ideal, they allowed us to display a functioning implementation of a cognitive radio system who was sensitive to the presence of a primary user.

In this experiment, the bit error rate was calculated in the same way as in the previous experiments. At the beginning of each trial, a primary user was activated so that the secondary users would have to switch channels during the experiment. The SNR of the primary user was recorded and the bit error rate on the one-way data stream between the secondary users was calculated from the number of packets received out of 1,000. The results of the experiment are provided by Table 4.2.

Because so few tests were run in this experiment, the results are slightly skewed and inconsistent. However, there are still important findings that can be derived from the results. The first thing to notice is that the interference from the primary user did not increase the bit error rate of the secondary users, except for in the final trial. Both channel sensing methods were able to detect the primary user at a low enough SNR that would not cause interference.

We had expected there to be a large bit error rate for the trials in which the primary user SNR was barely above the threshold for detection, such as in the trials for which the

Table 4.2: Bit error rate for different primary user signal-to-noise ratios for the energy detector and matched filter detector

PU SNR (dB)	**Control BER**	**ED BER**	**ED Switched**	**MFD BER**	**MFD Switched**
1.0	1.04e-4	1.07e-4	N	9.52e-5	N
1.8	8.85e-5	1.00e-4	N	8.68e-5	N
2.2	8.85e-5	9.26e-5	N	1.05e-4	N
2.6	8.01e-5	9.79e-5	N	1.23e-4	N
3.8	8.63e-5	9.34e-5	N	1.16e-4	N
5.4	7.50e-5	9.69e-5	N	1.24e-4	Y
6.4	8.98e-5	9.58e-4	N	8.48e-5	Y
8.0	8.90e-5	1.01e-4	N	8.31e-5	Y
9.7	9.16e-5	8.36e-5	Y	8.80e-5	Y
11.5	1.08e-4	1.20e-4	Y	9.29e-5	Y
29.0	1	1.33e-4	Y	1.28e-4	Y

SNR was 8.0, 9.7, or 11.5 dB for the energy detector or 3.8, 5.4, or 6.4 dB for the matched filter detector. We expected this because it seemed likely that one of the secondary users would detect the primary user and switch channels, while the other would not, completely disrupting communications. However, the secondary users made the same switching decisions in the same trials, allowing for communication to continue in an unoccupied channel.

Finally, the most significant part of the results lies in the last trial. At an SNR of 29 dB, the BER for the control test was 1, meaning that no packets were received and communication between secondary users was impossible. However, when channel sensing and switching functionality was enabled, both the energy detector and the matched filter detector detected the presence of the primary user and switched channels, allowing them to continue communication in an unoccupied channel. This trial demonstrated the core functionality of the cognitive radio system designed and implemented in this thesis, and thus, its basic functionality had proven to be a success.

Chapter 5

Conclusions

In this thesis, cognitive radio was introduced and discussed in the context of existing literature. Two of our own implementations of cognitive radio systems were then presented along with the tools and equipment used to create them. Central to our experimental implementations were the methods of primary user detection. One system featured energy detection, while the other featured matched filter detection. The cycle interval was defined as the interval of time spent in two sub-intervals of the cognitive radio's operation: the sensing interval and the TX/RX interval. Then, four experiments were discussed and performed in order to determine the probability of successful detection of each method of spectrum sensing at varying primary user SNRs, to determine the minimum sensing interval value that maintains the probability of successful detection, to determine the minimum cycle interval that results in minimum BER, and to observe the total functionality of our CR systems in the presence of a PU.

In Experiment 1, the probability of the successful detection of each method of spectrum sensing was determined by running both the ED and the MFD in the presence of a PU. The amount of trials that each method successfully detected the PU's presence was recorded out of 10,000. Assuming each trial was a Bernoulli trial, the probability of successful detection was calculated. The results of this experiment show that the MFD provided a much higher probability of detection than the ED at lower PU SNRs. The ED was able to achieve a high probability once the SNR exceeded 9.1 dB.

In Experiment 2, the minimum sensing interval which would maintain the probability of detection found in Experiment 1 was determined. This was done using each spectrum sensing method and varying the length of the sensing interval from 10 ms to 60 ms. Again, this experiment was run over 10,000 Bernoulli trials at each sensing interval to achieve a confident value of probability. It was found that each method performed poorly at intervals lower than 30 ms, and both were able to achieve a probability consistent with the previous experiment at an interval of 60 ms.

In Experiment 3, the objective was to determine the value of the cycle interval that would minimize the BER between the two secondary users. Using an infinite cycle interval as our control, i.e., channel switching functionality was disabled, each secondary user's cycle interval was varied from 90 ms to 390 ms, increasing in increments of 30 ms. It was found that, as expected, the BER decreased with increased cycle intervals. At a cycle interval of 390 ms, the BER became close to the ideal, and the decision was made that the trade-off between cycle time and BER was acceptable.

Finally, in Experiment 4, we displayed the functionality of our CR systems as a whole with spectrum sensing, channel switching, and data transmission enabled. The execution and results of this experiment revealed some problems with our previous conclusions and the implementations from which they were derived. For example, the sensing interval and cycle interval parameters were found to be insufficiently small in practice, and led to unsatisfactory performance. Increasing these parameters to larger values led to greatly improved performance. We measured the performance of our CR systems in terms of BER at varying PU SNRs. In each trial, we started the CRs in an unoccupied channel, began data transmission, and then turned on the PU in the same channel. The results of this experiment are shown in Table 4.2. The most significant result from this experiment comes from our final trial, when the PU was transmitting at such a high energy level that the BER when the radios stayed in the occupied channel was 1. However, when channel switching was enabled, both the ED based CR and the MFD based CR successfully switched into an unoccupied channel and were able to communicate with a much better BER.

From the results of Experiment 1, it was concluded that the CR implementation that utilizes MFD is the more robust of the two presented in this thesis in terms of probability of PU signal detection. This is primarily due to the MFD method's ability to more reliably detect the presence of a PU signal at lower SNRs than the ED method. From Experiment 3, it is also important to note that we were able to obtain a BER between the CR systems that came close to that of a regular radio system in an unoccupied channel by finding an ideal cycle interval. This shows that CR systems are able to transmit data with minimal penalty. Finally, the results of the final trial of Experiment 4 show that both of our CR implementations were able to perform well by switching out of an occupied channel where they would not have been able to communicate at all with each other, and into an unoccupied channel where they could communicate at a much lower BER.

Our experimental design produced several shortcomings. Now that they have been conducted, we are able to reflect on these shortcomings and what we could have done to provide better tests for our system. All of the experiments could have benefited from more trials and being executed multiple times. Many of the results show inconsistencies and anomalies in the data. This is because either not enough experiments were run, or not enough values were tested. For example, the curves in Figure 4.2 would likely give us a much better understanding of the behavior of the detection methods had more trials been executed. Additionally, our experiments did not investigate more traditional metrics of cognitive radio performance, such as probability of false detection. Ultimately, this was due to time constraints, and the probability of successful detection was deemed a more valuable metric to spend time investigating and measuring.

In the third experiment, only the energy detector was used to find the ideal cycle interval. One of our greatest takeaways from this project is that systems very rarely work the way you intend them to. Perhaps repeating the cycle interval tests with the matched filter detector would have revealed some previously unknown characteristic about the detection method or the experiment itself.

The largest and most disappointing shortcoming was that, after spending weeks forming and executing experiments to find the ideal values for sensing and cycle intervals, we were not able to use them for the final demonstration in the fourth experiment. This was most likely due to a number of details being overlooked in the setup and the programming for the channel switcher blocks. From the results of the first three experiments, we believe that a functional implementation can be achieved with low sensing and cycle intervals by decreasing the offset in the system clocks and providing more robust checks of the system time in the channel switcher block code. Further work would be needed to test this belief.

Our experiments contained at most only one primary user and two secondary users. In a real-world application, this arrangement would be rare. Most wireless networks consist of far more users, and the consideration of many users in a cognitive radio network was a common theme in our literature review. Because of our focused effort on the investigation of spectrum sensing techniques, our networks lacked variety, and we are unable to make a conjecture as to the scalability of our system. Further work may also include increasing the number of primary and secondary users to determine whether or not the results of our experimentation could be extended to a network of N secondary users and M primary users. Thus, practical values of N and M may be obtained.

Cognitive radio has undoubtedly come a long way since Mitola first introduced the idea to the communications community. However, despite seeing several significant milestones such as TVWS and incorporation of SAS into the CBRS Band, cognitive radio technology still has a ways to go before its implementation will be commonplace [5], [25]. The reason behind CR's relatively slow commercial deployment can be largely traced back to uncertainty surrounding CR's performance in practical settings, as well as its inability to provide a pressing use case that current cellular networks do not already offer. Developing a better understanding of CR performance will be a crucial step forward in expanding CR use cases. Much of the uncertainty surrounding performance analysis of practical CR implementations relates to unreliability of primary user detection techniques. As the most important metric for CRN performance, the CR nodes' ability to effectively detect primary users' signals and mitigate harmful interference to them is of central importance to CR's applicability in commercial settings. However, as we have seen with our experiments, meeting these performance requirements is often a challenging task, and there are often significant limitations imposed by the sensing method itself.

In general, there exists a trade off between detection sensitivity and computational complexity, which complicates the CR system design process. As a result, future work in the implementation of CR systems will necessarily involve complexity reduction of the more sophisticated sensing techniques that tend to be more reliable, such as cyclostationary feature detection and other wideband sensing techniques. Nevertheless, the diverse, flexible features that CR paradigm has to offer along with its existing success in several applications bode well for the future of this technology.

BIBLIOGRAPHY

Bibliography

[1] C. Tellambura and S. Kusaladharma, *An Overview of Cognitive Radio Networks.* Wiley Encyclopedia of Electronics and Electrical Engineering, Mar. 2017, ch. Networking.

[2] D. Das and S. Das, "A Survey on Spectrum Occupancy Measurement for Cognitive Radio," *Wireless Personal Communications*, vol. 85, p. 2581–2598, Jul. 2015.

[3] "Cognitive Radio for Public Safety," https://www.fcc.gov/general/cognitive-radio-public-safety, accessed: 2020-10-28.

[4] D. Scaperoth, "Configurable SDR Operation for Cognitive Radio Applications using GNU Radio and the Universal Software Radio Peripheral," Master's thesis, Virginia Tech, 2007.

[5] A. Kaushik, S. K. Sharma, S. Chatzinotas, B. Ottersten, and F. K. Jondral, *Modeling and Performance Analysis of Cognitive Radio Systems from a Deployment Perspective.* Singapore: Springer Singapore, 2019, pp. 87–128. [Online]. Available: https://doi.org/10.1007/978-981-10-1394-2_4

[6] A. Rehan and G. Y. Arfat, "Detection of Vacant Frequency Bands in Cognitive Radio," Master's thesis, Blekinge Institute of Technology, School of Engineering, 2010.

[7] K. Ghosh and A. Nath, “Cognitive Radio Networks: a Comprehensive Study on Scope and Applications,” *International Journal of Innovative Research in Advanced Engineering*, vol. 2, pp. 44–56, Jan. 2016.

[8] S. Mehta, S. Sharma, and R. Khanna, “SMT-8036 Based Implementation of Secured Software Defined Radio System for Adaptive Modulation Technique,” in *Advances in Computing and Communications*, A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 205–212.

[9] V. T. Nguyen, F. Villain, and Y. L. Guillou, “Cognitive Radio RF: Overview and Challenges,” *VLSI Design*, vol. 2012, 2012. [Online]. Available: https://doi.org/10.1155/2012/716476

[10] T. Mukherjee and A. Nath, “Issues and Challenges in Spectrum management in Cognitive Radio Networks,” *Current Trends in Technology and Science*, vol. 4, pp. 545–554, Jul. 2015.

[11] S. Li, T. H. Luan, and X. Shen, “Channel Allocation for Smooth Video Delivery over Cognitive Radio Networks,” in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, 2010, pp. 1–5.

[12] M. Zareei, I. A. K. M. Muzahidul, S. Baharun, C. Rosales, L. Azpilicueta, and N. Mansoor, “Medium Access Control Protocols for Cognitive Radio Ad Hoc Networks: A Survey,” *Sensors (Basel, Switzerland)*, vol. 17, 2017.

[13] Y. Arjounne and N. Kaabouch, "A Comprehensive Survey on Spectrum Sensing in Cognitive Radio Networks: Recent Advances, New Challenges, and Future Research Directions," *Sensors (Basel)*, vol. 19(1), no. 126, 2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6339174/

[14] B. Ahsant and R. Viswanathan, *A Review of Cooperative Spectrum Sensing in Cognitive Radios*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 69–80. [Online]. Available: https://doi.org/10.1007/978-3-642-32180-1_4

[15] A. Ranjan, Anurag, and B. Singh, "Design and analysis of spectrum sensing in cognitive radio based on energy detection," in *2016 International Conference on Signal and Information Processing (IConSIP)*, 2016, pp. 1–5.

[16] C. Tellambura, *Spectrum Sensing Methods and Their Performance.* Singapore: Springer Singapore, 2018, pp. 1–22. [Online]. Available: https://doi.org/10.1007/978-981-10-1389-8_6-1

[17] K. B. Letaief and W. Zhang, "Cooperative Communications for Cognitive Radio Networks," *Proceedings of the IEEE*, vol. 97, no. 5, pp. 878–893, 2009.

[18] A. Sharmila and P. Dananjayan, "Spectrum Sharing Techniques in Cognitive Radio Networks–A Survey," in *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*, 2019, pp. 1–4.

[19] J. Mitola, "Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio," 2000.

[20] J. Mitola and G. Q. Maguire, "Cognitive radio: Making software radios more personal," *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, 1999.

[21] M. López-Benítez, *Overview of Recent Applications of Cognitive Radio in Wireless Communication Systems*. Singapore: Springer Singapore, 2018, pp. 1–32. [Online]. Available: https://doi.org/10.1007/978-981-10-1389-8_59-1

[22] F. C. Commission, "In the Matter of Unlicensed Operation in the TV Broadcast Bands, Additional Spectrum for Unlicensed Devices Below 900 MHz and in the 3 GHz Band," https://www.fcc.gov/document/matter-unlicensed-operation-tv-broadcast-bands-additional, 11 2008.

[23] O. Holland, *Spectrum Policy and Cognitive Radio Standards*. Singapore: Springer Singapore, 2018, pp. 1–24. [Online]. Available: https://doi.org/10.1007/978-981-10-1389-8_46-1

[24] J. H. Martin, L. S. Dooley, and K. C. P. Wong, *Cognitive Radio and TV White Space (TVWS) Applications*. Singapore: Springer Singapore, 2019, pp. 1935–1970. [Online]. Available: https://doi.org/10.1007/978-981-10-1394-2_62

[25] P. Anker, *Cognitive Radio: The Need to Align Regulations with Technology*. Singapore: Springer Singapore, 2019, pp. 1537–1557. [Online]. Available: https://doi.org/10.1007/978-981-10-1394-2_45

[26] K. Ishizu, K. Mizutani, T. Matsumura, Z. Lan, and H. Harada, *IEEE 802.11af Wi-Fi in TV White Space*. Singapore: Springer Singapore, 2019, pp. 1509–1535. [Online]. Available: https://doi.org/10.1007/978-981-10-1394-2_53

[27] M. D. Mueck, S. Srikanteswara, and B. Badic, "Spectrum Sharing: Licensed Shared Access (LSA) and Spectrum Access System (SAS)," Intel, Whitepaper, 2015.

[28] "3.5 GHz Band Overview," https://www.fcc.gov/wireless/bureau-divisions/mobility-division/35-ghz-band/35-ghz-band-overview, accessed: 2021-03-03.

[29] "CBRS, SAS and Spectrum Sharing: The Complete Guide," https://blinqnetworks.com/cbrs-sas-spectrum-sharing-guide/, accessed: 2021-03-03.

[30] A. H. Sakr, H. Tabassum, E. Hossain, and D. I. Kim, "Cognitive spectrum access in device-to-device-enabled cellular networks," *IEEE Communications Magazine*, vol. 53, no. 7, pp. 126–133, 2015.

[31] A. Abu Alkheir and H. T. Mouftah, "Cognitive Radio for Public Safety Communications," in *Wireless Public Safety Networks 2*, D. Câmara and N. Nikaein, Eds. Elsevier, 2016, pp. 295 – 316. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9781785480522500104

[32] N. Uchida, G. Sato, K. Takahata, and Y. Shibata, "Optimal Route Selection Method with Satellite System for Cognitive Wireless Network in Disaster Information Network," in *2011 IEEE International Conference on Advanced Information Networking and Applications*, 2011, pp. 23–29.

[33] E. C. Y. Peh, Y. Liang, Y. L. Guan, and Y. Pei, "Energy-Efficient Cooperative Spectrum Sensing in Cognitive Radio Networks," in *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, 2011, pp. 1–5.

[34] "UHD," https://kb.ettus.com/UHD, accessed: 2021-03-15.

[35] J. A. Gilbert, “Accessing the RF Spectrum with GNU Radio.” Mar. 2019. [Online]. Available: https://www.osti.gov/biblio/1639511

[36] M. A. Sarijari, A. Marwanto, N. Fisal, S. K. S. Yusof, R. A. Rashid, and M. H. Satria, “Energy detection sensing based on GNU radio and USRP: An analysis study,” in *2009 IEEE 9th Malaysia International Conference on Communications (MICC)*, 2009, pp. 338–342.

[37] J. Malsbury, B. Seeber, and M. Dickens, “GR-MAC,” https://github.com/jmalsbury/gr-mac, Sept. 2014.

[38] R. Baraniuk, *Signals and Systems*, 15th ed. OpenStax CNX, Oct. 2015. [Online]. Available: http://cnx.org/contents/77608400-65b9-4f03-8a5f-536c611866bb@15.4.

Appendix A

Channel Switching Block

All of the Python code used for the channel switching block, the rapid testing block, the UDP transmitter and receiver, and the testing drivers are included in this appendix. Descriptions for these codes are given in section 3.2.

```
"""
Embedded Python Blocks:

Each this file is saved, GRC will instantiate the first class it finds to get
ports and parameters of your block. The arguments to __init__  will be the
parameters. All of them are required to have default values!
"""
import numpy as np
from gnuradio import gr
import pmt
import time
import math

###### Definition of Global Variables ######
# CHANNELS[x] will be the wifi channel x+1.
#CHANNELS = [2412e6, 2417e6, 2422e6, 2427e6, 2432e6, 2437e6, 2442e6, 2447e6,
    2452e6, 2457e6, 2462e6, 2467e6, 2472e6, 2484e6]
#CHANNELS = [2484e6, 2486e6, 2488e6, 2490e6, 2492e6, 2494e6]
CHANNELS = [2481e6, 2482e6, 2483e6, 2484e6, 2485e6, 2486e6, 2487e6, 2488e6,
    2489e6, 2490e6, 2491e6, 2492e6, 2493e6, 2494e6, 2495e6]

DETECT_CHANNEL_TIME = 0.2 # time interval in seconds for spectrum analysis,
    per channel
checking = 1 # Three modes for checking. 1 = TXRX, 2 = Finding a suitable
    channel, 3 = found a suitable channel, waiting for checking period to
    expire
tscs = 0.0  # Time since check started
channels_checked = 0 # Number of channels checked in a cycle
threshold_avg = 0.0
previous_channel_determined = False

current_channel = 0
# Using last_channel to ensure that messages are only passed when the channel
    changes (last_channel != current channel)
```

```
# This is to avoid performance issues when trying to pass messages at a high
    rate
last_channel = current_channel
test_freq = 1000
switching = 0
custom_freq = CHANNELS[current_channel]

tx_amp = 0.0


####### Helper Functions ######

def increment_channel():
    global current_channel
    current_channel += 1
    if current_channel > len(CHANNELS)-1:
        current_channel = 0

###### Main Block Class ######

class blk(gr.sync_block):
    # Block Initiation
    def __init__(self, cycle_time=5):  # only default arguments here
        gr.sync_block.__init__(
            self,
            name='Embedded␣Python␣Block',
            in_sig = [np.float32],
            out_sig=None,

        )

        # Create the input message ports for switching and custom frequency,
            when
        # switching is turned off.
        self.message_port_register_in(pmt.intern('switching'))
        self.message_port_register_in(pmt.intern('custom_freq'))
        self.set_msg_handler(pmt.intern('switching'), self.set_switching)
        self.set_msg_handler(pmt.intern('custom_freq'), self.set_custom_freq)

        # Create the output message ports for channel switching and debugging
        self.message_port_register_out(pmt.intern('msg_out_rx'))
        self.message_port_register_out(pmt.intern('msg_out_tx'))
        self.message_port_register_out(pmt.intern('debug'))
        self.cycle_time = cycle_time

    ####### Helper Functions for Message Reception ######
    def set_switching(self, msg):
        '''
        Incoming message handler to determine if channel switching is on or
            off
        '''
        global switching
        # For some reason, python requires casting to float, then to int
        switching = int(float(pmt.to_python(msg)))
```

```
def set_custom_freq(self, msg):
    '''
    Incoming message handler to determine the target frequency to use
    if channel switching is off.
    '''
    global custom_freq
    custom_freq = float(pmt.to_python(msg))

###### Main Work Function for Block ######
def work(self, input_items, output_items):
    # Set global variables for continuous processing
    global current_channel
    global last_channel
    global switching
    global custom_freq
    global checking
    global tscs
    global channels_checked
    global threshold_avg
    global previous_channel_determined
    global tx_amp



    # Set threshold to the most recent threshold value
    threshold = input_items[0][-1]
    threshold_avg = threshold_avg * 0.9 + threshold * 0.1 #Do Rolling
        average for threshold
    if switching == 1.0: # If channel switching is turned on

        if checking == 2: # If checking is set to check-channel mode
            current_time = time.time()
            if current_time >= tscs+DETECT_CHANNEL_TIME*(channels_checked
                +1):

                if not previous_channel_determined:
                    # If the previous channel has not yet been deemed
                        unsuitable, make a determination before
                    # moving on
                    if threshold_avg > 0.5:
                        print(CHANNELS[current_channel], "unsuitable,␣
                            checking␣next␣channel")
                        increment_channel()
                        self.message_port_pub(pmt.intern('msg_out_rx'),
                            pmt.cons(pmt.to_pmt('freq'), pmt.to_pmt(
                            CHANNELS[current_channel])))
                        self.message_port_pub(pmt.intern('msg_out_tx'),
                            pmt.cons(pmt.to_pmt('freq'), pmt.to_pmt(
                            CHANNELS[current_channel])))
                        channels_checked += 1
                        threshold_avg = 0.0
                    else:
```

```
                    print(CHANNELS[current_channel], "no␣primary␣user␣
                        detected...␣staying␣here,␣avg=", threshold_avg
                        )
                    checking = 1

                else:
                    # If the previous channel has been deemed unsuitable,
                        accumulate average for threshold_avg
                    # Suitability will be determined after the channel is
                        switched!
                    pass

        else:
            # If checking == 1, the CR is in the TXRX step, and is not
                checking.
            # If not checking, repeatedly check if it is time for checking
            if math.fmod(time.time(), self.cycle_time) <= 0.02:
                print("Starting␣checking")
                # Change checking mode to checking
                checking = 2
                # Switch to a channel in CHANNELS list
                self.message_port_pub(pmt.intern('msg_out_rx'), pmt.cons(
                    pmt.to_pmt('freq'), pmt.to_pmt(CHANNELS[
                    current_channel])))
                self.message_port_pub(pmt.intern('msg_out_tx'), pmt.cons(
                    pmt.to_pmt('freq'), pmt.to_pmt(CHANNELS[
                    current_channel])))
                # Set the transmitter gain to 0, to ensure no interference
                     with channel switching
                tx_amp = 0.0
                self.message_port_pub(pmt.intern('msg_out_tx'), pmt.cons(
                    pmt.to_pmt('gain'), pmt.to_pmt(tx_amp)))
                # Start the timer, used in the channel switching
                    functionality
                tscs = time.time()

            elif (tx_amp == 0.0):
                # Set the transmitter gain to 62.5, to allow for
                    transmission/reception of data
                # Check the tx_amp value so that messages are not
                    continously sent, which would greatly hinder
                    performance.
                tx_amp = 62.5
                self.message_port_pub(pmt.intern('msg_out_tx'), pmt.cons(
                    pmt.to_pmt('gain'), pmt.to_pmt(tx_amp)))

    else:
        # If switching is off, transmit at the desired custom frequency
        self.message_port_pub(pmt.intern('msg_out_rx'), pmt.cons(pmt.
            to_pmt('freq'), pmt.to_pmt(custom_freq)))
        current_channel = 0

    return len(input_items[0])
```

Appendix B

Simplified Channel Switching Block

```
"""
Embedded Python Blocks:

Each this file is saved, GRC will instantiate the first class it finds to get
ports and parameters of your block. The arguments to __init__ will be the
parameters. All of them are required to have default values!
"""
import numpy as np
from gnuradio import gr
import pmt
import time
import math

###### Definition of Global Variables ######
tscs = 0.0  # Time since check started
threshold_avg = 0.0
switching = 0

###### Main Block Class ######

class blk(gr.sync_block):
    def __init__(self, cycle_interval=5, sensing_interval=0.2):  # only
        default arguments here
        gr.sync_block.__init__(
            self,
            name='Embedded_Python_Block',
            in_sig = [np.float32],
            out_sig = None
        )

        # Create the input message ports for switching and custom frequency,
            when
        # switching is turned off.
        self.message_port_register_in(pmt.intern('switching'))
        self.set_msg_handler(pmt.intern('switching'), self.set_switching)

        # Create the output message ports for channel switching and debugging
        self.message_port_register_out(pmt.intern('out'))
        self.cycle_interval = cycle_interval
        self.sensing_interval = sensing_interval

    ####### Helper Functions for Message Reception ######
    def set_switching(self, msg):
        '''
        Incoming message handler to determine if channel switching is on or
            off
        '''
        global switching
        # For some reason, python requires casting to float, then to int
        switching = int(float(pmt.to_python(msg)))

    ###### Main Work Function for Block ######
    def work(self, input_items, output_items):
        # Set global variables for continuous processing
        global switching
        global tscs
        global threshold_avg


        # Set threshold to the most recent threshold value
        threshold = input_items[0][-1]
        threshold_avg = threshold_avg * 0.9 + threshold * 0.1 #Do Rolling
            average for threshold

        if switching == 1.0: # If channel switching is turned on

            current_time = time.time()
            if current_time >= tscs+self.sensing_interval:

                if threshold_avg > 0.5:
                    self.message_port_pub(pmt.intern('out'), pmt.intern(str("
                        occupied")))

                else:
                    self.message_port_pub(pmt.intern('out'), pmt.intern(str("
                        vacant")))

                threshold_avg = 0.0
                tscs = time.time()

            else:
                # If the previous channel has been deemed unsuitable,
                    accumulate average for threshold_avg
                # Suitability will be determined after the channel is switched
                    !
                pass

        return len(input_items[0])
```

Appendix C

Rapid Testing Block

```
"""
Embedded Python Blocks:

Each this file is saved, GRC will instantiate the first class it finds to get
ports and parameters of your block. The arguments to __init__  will be the
parameters. All of them are required to have default values!
"""
import numpy as np
from gnuradio import gr
from datetime import datetime
import pmt
import os
import signal

global i

class blk(gr.sync_block):
    #def __init__(self):
    def __init__(self, filename="/home/will/Desktop/rapid_test_v2/trial_00.txt
       ", max_runs=5):  # only default arguments here
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',
            in_sig = None,
            out_sig = None
        )
        global i
        self.message_port_register_in(pmt.intern('monitor'))
        self.set_msg_handler(pmt.intern('monitor'), self.handle_input)
        self.max_runs = max_runs
        self.fname = filename
        i = 0

    def handle_input(self, msg):
        global i
        i += 1
        f = open(self.fname, mode='a')
        f.write(str(i) + " " + pmt.to_python(msg) + "\n")
        f.close()

        if i == self.max_runs:
            print(self.fname)
            print(str(i) + " tests completed. Stopping.")
            os._exit(0)


    def work(self, input_items, output_items):
        return len(output_items[0])
```

Appendix D

UDP Transmitter

```
import socket
import string
import random
import time
import datetime

UDP_IP = "192.168.200.2"
UDP_PORT = 5005

MAX_WAIT = 0.1       # Max time to wait between sending messages (actual time
    is random)
MIN_WAIT = 0.090

PACKET_LENGTH = 1000
TOTAL_PACKETS = 1000

# Initialize Socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

p = 0
while p < TOTAL_PACKETS:

    msg = ''.join(random.choice(string.ascii_lowercase + string.
        ascii_uppercase) for _ in range(PACKET_LENGTH))
    # Send the message to the socket
    sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))

    # Sleep for a random amount of time
    time.sleep(random.uniform(MIN_WAIT, MAX_WAIT))
    print(p)
    p += 1

print("Experiment Concluded. Number of packets transmitted:", p)
```

Appendix E

UDP Receiver

```
import socket
import string
import random
import time
import datetime

UDP_IP = "192.168.200.2"
UDP_PORT = 5005
DURATION = 5
MAX_LENGTH = 1000   # Maximum length of message
MAX_WAIT = 1        # Max time to wait between sending messages (actual time is
    random)
MIN_WAIT = 0.05
# Initialize Socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind((UDP_IP, UDP_PORT))
sock.settimeout(0.5)
start_time = time.time()
channel_interval = input("What is the channel interval for this test?")
filename = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S_client_") + str
    (channel_interval) + ".txt"
p=0
try:
    while True:
        try:
            data = ""
            data = sock.recv(1024)
            if data:
                p+=1
                print(p)
        except socket.timeout:
            pass

except KeyboardInterrupt:
    print(p)
print(p)
```

Appendix F

Rapid Test Driver

```python
import os
LOGPATH = "/home/will/Desktop/logs/"
tests = int(input("How many tests would you like to execute?\n"))
gain = int(input("What gain value is the PU at?\n"))
snr = float(input("What SNR is the power meter observing?\n"))

filename = str(LOGPATH + "trial_" + str(gain) + "dB.txt")
os.system(str("python ./ed/ed_rapid.py --fname '" + filename + "' --runs " +
    str(tests)))
#os.system(str("python ./mfd/mfd.py --fname '" + filename + "' --runs " + str(
    tests)))


occupied = 0
l = open(filename, "r")
lines = l.readlines()
l.close()
for l in lines:
    if l.find("occupied") != -1:
        occupied += 1

s = "TOTAL:" + str(occupied) + "/" + str(tests) + "\nSNR:" + str(snr) + "\n"
dummy = filename+".temp"
with open(filename, 'r') as read_obj, open(dummy, 'w') as write_obj:
    write_obj.write(s + "\n")
    for line in read_obj:
        write_obj.write(line)

    os.remove(filename)
    os.rename(dummy, filename)
```