MQP AW1 - CELL

Jamming Security for LTE Networks

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science.

Submitted by:

Julien Ataya,

Electrical and Computer Engineering

Matthew Farah,

Electrical and Computer Engineering

Project Advisor:

Professor Alexander Wyglinski,

Electrical and Computer Engineering

Worcester Polytechnic Institute

Submitted on: March 30, 2020

Abstract

The goal of this project is to build a 4G LTE jamming detector testbed to demonstrate its effectiveness during a jamming attack. The setup was implemented through the use of OpenAirInterface, GNU Radio, Python, and C via multiple software-defined radios and an LTE capable smartphone. The approach was successful and the base station was able to bypass a random frequency hopper interfering with the LTE downlink signal.

Acknowledgements

The project would not have been successful without all the help and resources received from many members of Worcester Polytechnic Institute, National Instruments, and the Air Force Research Lab. We want to thank Professor Wyglinski, our faculty advisor, for the guidance and enthusiasm he provided throughout the entirety of the project. We want to thank Kuldeep Gill, a PhD student working under Professor Wyglinski, for all the time he dedicated in helping us get acquainted with LTE technology and help throughout the many issues that arose along the way. We would also like to thank YaYa Brown and Cynthia Teng for helping us develop the testbed during A Term 2019 and for providing an intelligent jammer for us to use and develop security measures into the LTE system. We would also like to give our appreciation to Stephen Hary from the AFRL for helping us in the participation of the Beyond 5G competition and National Instruments for providing the team with multiple software-defined radios. Finally, we would like to thank Bob Lee from the Wright Brothers Institute for his role in the competition.

Authorship Page

Both authors contributed an equal amount of time towards project and report completion.

Executive Summary

Communication systems, whether it be 4G LTE or 5G, have become an integral part of society today and have caused people to be reliant on consistent connection between mobile phones and the LTE signal [1]. However, this connection can be disrupted through the use of a jamming attack, which can cause denial of service. These denial of service attacks have been increasing in recent years, making the jamming attacks becoming more complex [2]. Hence, there is a need to build more advanced security protocols that mitigate the interference of different jamming attacks. This project has developed an LTE testbed using software-defined radios and has built security protocols into the testbed to protect the system from wireless denial of service attacks. The jamming scheme, a single-tone jammer, was provided by another student team.

There were many different ways of actually building anti-jamming security into the testbed. The use of the field-programmable gate array (FPGA) found on the software-defined radio hardware was a potential solution, but it was determined that this low-latency processor was not necessary to achieve real-time results. The team decided to utilize a radio to act as a receiver and perform computations using GNU Radio/Python to ultimately bypass jamming. The testbed included a USRP B210 acting as the base station, a USRP N210 as a jammer, a Samsung Galaxy S4 as the user equipment, and a USRP X310 acting as the jamming detector. All of these components were analyzed inside of a Faraday cage to shield RF signals, and a 4-core Linux computer running Ubuntu 18.04 was connected to all of the software-defined radios in the system and was used to run the core network. In order to get everything running, the two main tutorials that were used were one provided by the Open Cells project and the other from Chance Tarver, who has developed his own step-by-step guide to implementing an LTE network using OpenAirInterface. Another student team consisting of YaYa Brown and Cynthia Teng developed

the narrow-band jammer using GNU Radio. Figure ES.1 shows this team's setup inside the Faraday cage and Figure ES.2 shows the testbed on the Linux computer. The screens are top-left: jammer and jamming detector terminals, top-right: LTE eNodeB and EPC terminals, bottom-left: jamming detector running, bottom-right: jammer running.

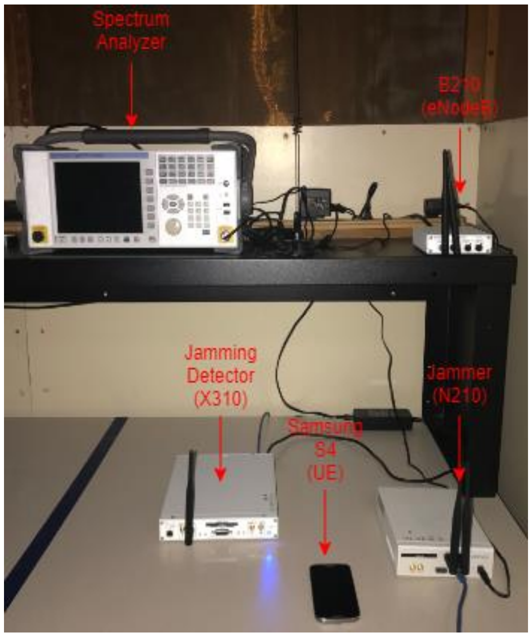


Figure ES.1: Hardware Setup of Testbed in Faraday Cage

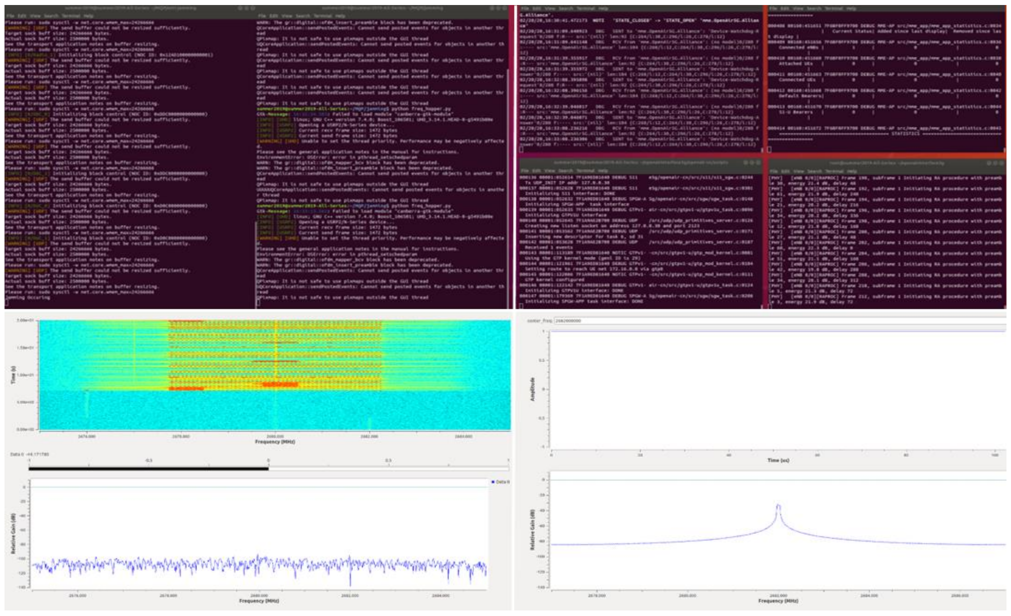


Figure ES.2: Testbed Setup with Jamming Detected

Figure ES.3 shows a concept diagram of how the jamming detector works in an LTE cellular network. The uplink signal is not disrupted, but the downlink signal is being targeted by the jammer and also monitored by the detector.

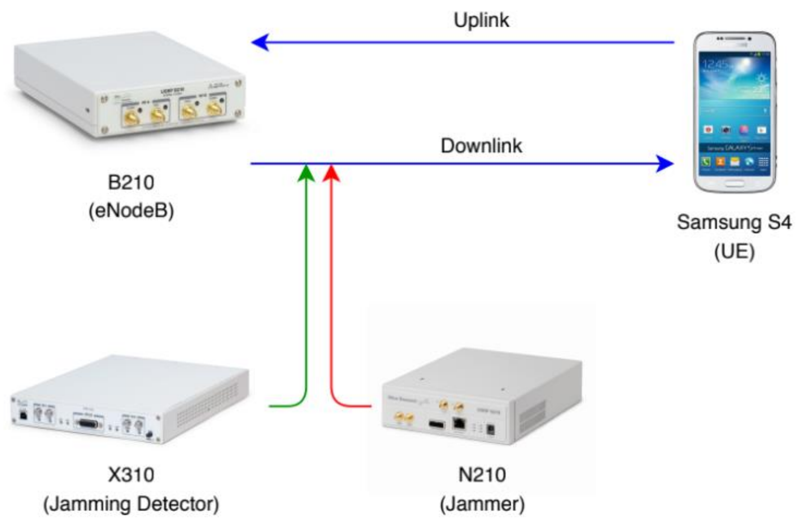


Figure ES.3: Concept Diagram of LTE Jamming Detection Testbed. USRP Images Taken from [25]

The development of the jamming detector, a USRP X310, was done by modifying a GNU Radio generated Python script, and this was then interfaced with the OpenAirInterface code to change the transmission frequency of the downlink signal. This detector would first self-determine a threshold power level for an ideal LTE signal and decide whether or not the power level is typical of an LTE signal. If any power spikes were found, then jamming was present and the X310 would send a signal to notify the eNodeB. To bypass the interference, the eNodeB would change the downlink center frequency in LTE Band 7 to enable the Samsung Galaxy S4 to reconnect to it. Figure ES.4 shows a conceptual flow chart of how the jamming detector works.

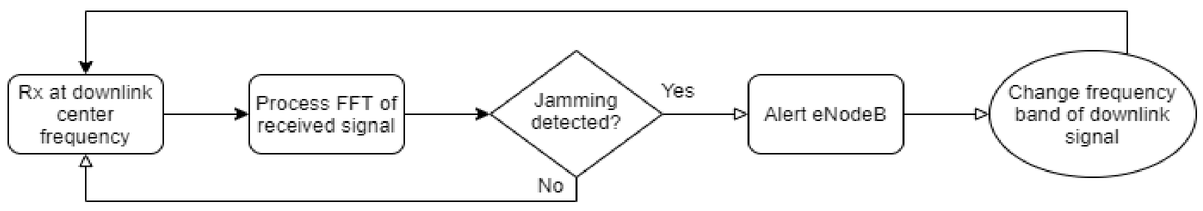

Rx at downlink center frequency
Process FFT of received signal
Jamming detected?
Yes
No
Alert eNodeB
Change frequency band of downlink signal

Figure ES.4: Conceptual Flow Diagram of Jamming Detector

In order to test the effectiveness of the anti-jamming approach, Wireshark, an open-source packet analyzer, was used to evaluate metrics such as throughput and packet receiving rate. Figure ES.5 shows the graph of the packet receiving rate as a function of time for the LTE system without the jamming attack and Figure ES.6 shows the plot after the eNodeB changed the downlink signal frequency to mitigate jamming. The packet loss analysis when the jammer was performing DoS showed that the packet receiving rate decreased to 0. Once the eNodeB was alerted of the interfering signal and it changed the downlink transmission frequency, the packet receiving rate increased. This shows how the jamming was bypassed and the UE was still able to connect to the Internet.

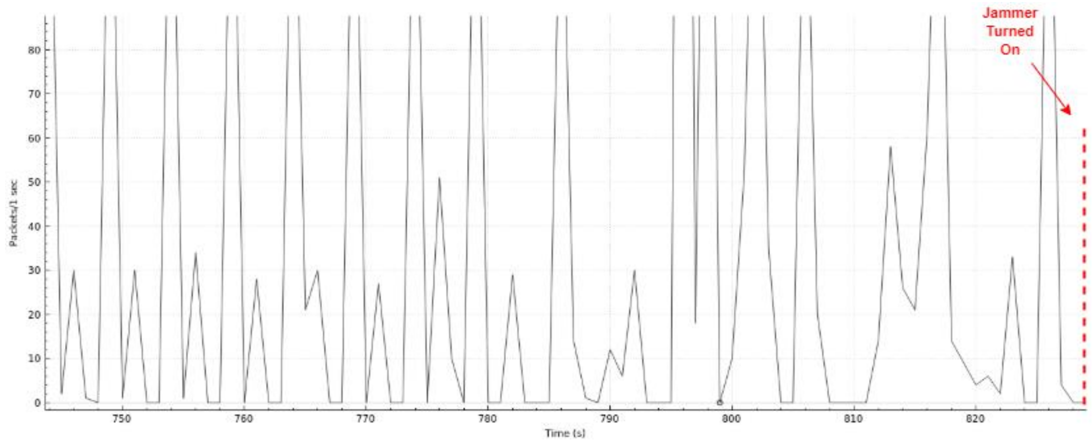


Figure ES.5: Packing Receiving Rate vs. Time at 2.68 GHz Downlink Frequency

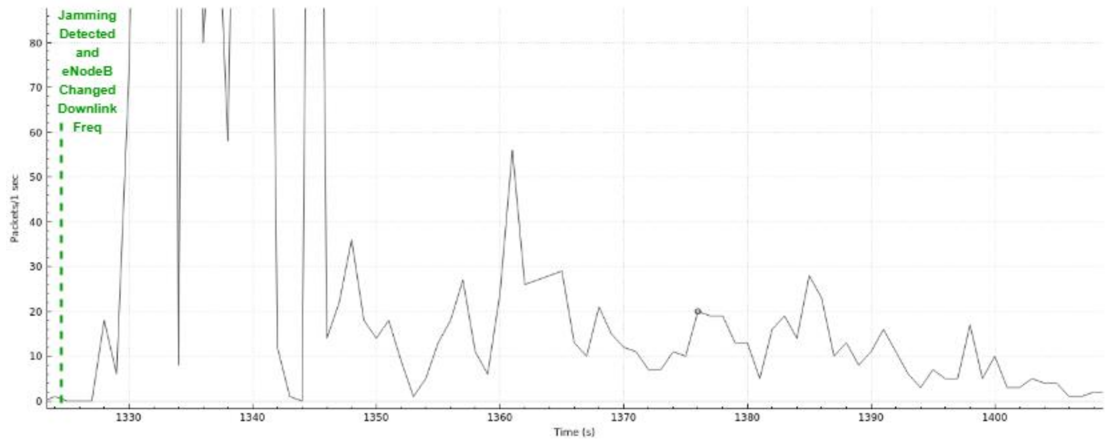


Figure ES.6: Packet Receiving Rate vs. Time After eNodeB Changed Downlink Frequency

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	**Description**
1G	First Generation Cellular Networks
2G	Second Generation Cellular Networks
3G	Third Generation Cellular Networks
3GPP	3rd Generation Partnership Project
4G LTE	Fourth Generation Long Term Evolution
CDMA	Code-division Multiple Access
COTS	Commercial Off-the-shelf
CRC	Cyclic Redundancy Check
DoS	Denial of Service
DSP	Digital Signal Processors
DSSS	Direct Sequence Spread Spectrum
E-UTRAN	Evolved UMTS Terrestrial Radio Access Network
EPC	Evolved Packet Core
FDD	Frequency Division Duplexing
FDMA	Frequency-division Multiple Access
FHSS	Frequency-hopping Spread Spectrum
FPGA	Field-programmable Gate Arrays
GPP	General Purpose Processors
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HSS	Home Subscriber Server
IQ	In-phase and Quadrature
MIMO	Multiple In Multiple Out
MME	Mobility Management Entity
MQP	Major Qualifying Project
OAI	OpenAirInterface
OFDMA	Orthogonal Frequency-division Multiple Access
PGW	Packet Data Network Gateway

PHY	Physical Layer
RF	Radio Frequency
SC-FDMA	Single Carrier Frequency-division Multiple Access
SDR	Software-defined Radio
SGW	Serving Gateway
SoC	System on Chip
SON	Self-organizing Networks
TDD	Time Division Duplexing
UE	User Equipment
USRP	Universal Software Radio Peripheral

1. Introduction

This chapter explains the significance of Long-Term Evolution (LTE) networks and their presence in society, along with the current work that has been done exploring the effects of jamming on them. Moreover, this section describes the team’s work and how it is relevant towards further advancements regarding jamming mitigation for LTE networks.

1.1 Motivation

Fourth Generation (4G) LTE networks help businesses, both small and large, throughout the world. Industries such as health, education, and defense have benefited from the use of LTE due to the significant presence of smartphones in society. Moreover, people use their cell phones every day and one dropped call or a few seconds of slow data speeds are noticeable. Worldwide, there are currently over 5 billion smartphone users, which equates to over 60% of the global population [1]. Figure 1.1 shows the amount of 4G LTE connections around the world since 2012 and the projected amount for 2020. The amount has been increasing significantly and will continue to do so.

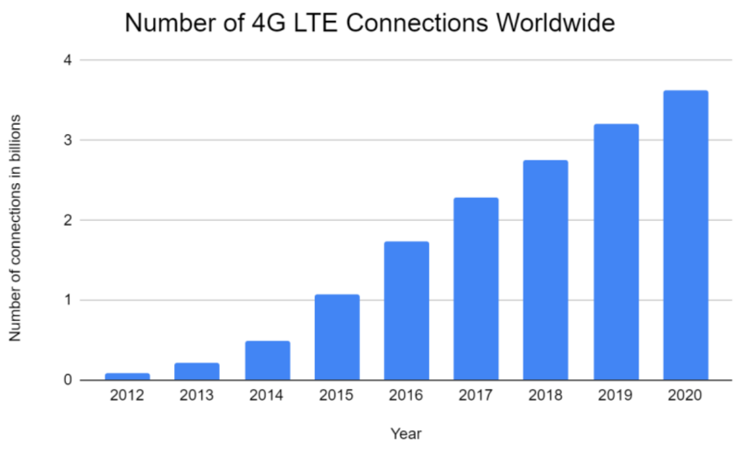


Figure 1.1: 4G LTE Connections Worldwide. Adapted from [3]

In addition to commercial use, the U.S. military uses LTE networks as a means of keeping soldiers on land, sea, and air all connected. Not only are commercial off-the-shelf (COTS) products such as smartphones easier to maintain and cheaper to acquire, but they are compatible with high-speed LTE, making them viable choices for the battlefield [4]. With LTE's already prominent presence and continuing growth, its technology is widely known, which also makes it increasingly vulnerable to jamming attacks. A study done at Virginia Tech discovered that a 4G LTE base station can be easily disrupted using fairly simple and inexpensive technology. In Particular, Jeff Reed and Marc Lichtman describe the control instructions that make the network so vulnerable when they are targeted with high-power noise transmission. According to the study, using a laptop and a software-defined radio (SDR), which can cost as little as $650, is sufficient to create this disturbance [5]. The motivation behind this project is to detect these security threats and implement a method to provide alerts to the cellular base station.

1.2 Current State of the Art

Investigating low-cost LTE networks has been made possible through the use of SDRs. Moreover, open-source software such as srsLTE [6], GNU Radio [7], and OpenAirInterface (OAI) [8] [20] can be used to interface with the Universal Software Radio Peripheral (USRP) hardware and create the LTE systems.

There is very little research available on LTE jamming mitigation [10], however, there are a few potential solutions that have been proposed. One group of researchers developed proof of concepts for methods of enhancing LTE security through the physical (PHY) layer communication [10]. By using spread spectrum modulation of downlink broadcast channels, this method aims to counteract jammers that attack key signals in the center 1 MHz frequency of the downlink. Direct Sequence Spread Spectrum (DSSS) distributes these signals across the entire 10 MHz of bandwidth to make targeted attacks more difficult by increasing the number of key subcarriers [10]. An alternative method that has been investigated involves the encryption of significant signals such as the master information block and the system information block. These contain vital network configuration parameters that allow the user equipment (UE) to connect to the LTE network, however, the signals are transmitted in the clear and are vulnerable to attacks. Adding encryption would strengthen this security and mitigate signal corruption [11]. Another study looking into military applications mentions an LTE system, where the eNodeB can detect jamming by using self-organizing network (SON) and solves the issue with radio frequency (RF) frequency hopping [12]. In situations where the frequency of the LTE signal is not known to the jammer, minimizing the transmitted power can make it difficult for the attacker to discover them [13]. However, if the signal transmissions are already known, maximizing the power would increase resistance to the jammer by requiring a jamming signal strong enough to disrupt the LTE signal [13]. The goal for this project is to expand on current research and determine the best solution for networks experiencing high-power wireless jamming.

1.3 Technical Challenges

In LTE systems, the scale of importance of reliable service depends on the application, but oftentimes interference can cause drastic outcomes. Jamming attacks can lead to denial of service (DoS), which essentially blocks the proper transmission, and can have negative consequences. According to the latest security incident statistics, DoS attacks have been rising substantially, and this becomes ever more important with the further advancement of technology that relies on LTE [14].

DoS attacks have not also become more sophisticated over the years, but can be easily performed through the use of software-defined radios such as USRPs. These jamming attacks have four major categories: flood attacks, amplification attacks, protocol vulnerability exploitation, and malformed packets [14]. Due to the complexity associated with each DoS, it is vital for LTE systems to contain active security measures to prevent DoS from occurring. The main themes associated with preventing DoS today include: the attack rate, attack attribution, effective traffic analysis, sophisticated attacks, effective mitigation, performance evaluation, and omnipresent connections [14]. A successful LTE system contains sufficient security measures to prevent DoS. Due to the continued increase in cyber-warfare attacks, this issue should not be taken lightly and further solutions that exist today must be investigated.

Due to the disturbance that jamming attacks create, it becomes even more complex to deal with all possible scenarios of creating anti-jamming protocols and in the end create a system that countermeasures most, if not all, jamming methods. Some of the current anti-jamming protocols include: Hermes Nodes, Control Channel Attack Prevention, MULEPRO, Cross-Layer Jamming Detection and Mitigation [15].

Hermes Nodes is a hybrid of DSSS and Frequency-hopping Spread Spectrum (FHSS). Here the DSSS makes use of a wide bandwidth to transmit the signal and makes the jammer believe that the real data is white noise instead. The FHSS allows for the avoidance of interference [9].

The Control Channel Attack Prevention approach is a way of scanning the control channel to detect if a node has been jammed or not. When a node has been found to be compromised, then the control channel is reset through the use of an updated hopping sequence [15].

MULEPRO extends to each node and each specific node will determine whether or not it is being jammed. This is done by having two modes on the eNodeB: normal mode where the common channel is only used and exfiltration mode that takes use of a milt-channel approach to send and receive data in a DoS attack [15].

The Cross-Layer Jamming Detection and Mitigation method is based on the PHY layer and takes use of a tree-based approach, where messages can be transmitted using a single hopping pattern. Once jamming is detected, the transmitter uses other test patterns in transmission [15].

Most of the current anti-jamming protocols are based upon spread spectrum techniques that include frequency hopping and Code-division Multiple Access (CDMA) [16]. These were initially developed for use in the military, but they do not provide sufficient security measures in the world today that is run on “high-speed multimedia wireless services” [16].

1.4 Contributions

Throughout the stages of the project, this Major Qualifying Project (MQP) has developed:

1. An LTE System that includes a base station and LTE compatible Samsung Galaxy S4. The system is located in a Faraday cage to shield RF signal, as this system is operating at a licensed band (2.68GHz initially). The Samsung Galaxy S4 successfully connects to the base station and has direct access to the internet, and is quite fast.
2. An anti-jamming approach that uses I/Q characteristics across the LTE signal spectrum to determine if jamming is being present. If jamming is found, a message will be sent to the base station to change to a different frequency and avoid the jamming.

1.5 Report Organization

The rest of this report is organized as the following: it provides a detailed explanation of LTE architecture and the components of the testbed. Chapter 3 outlines the team’s milestones with a Gantt chart for the academic year. The methodology and implementation of the mitigation techniques are then described in chapter 4 and analyzed in chapter 5. Recommendations for future work are given in Chapter 6.

2. Overview of LTE Technology

This chapter outlines LTE and provides insight to the multiple layers of the technology. It explains terms such as the UE and eNodeB, along with how they communicate with each other. This section also describes the operation of the core network within LTE. The security of this technology is also explored and the specific areas vulnerable to attack are addressed, along with potential methods of mitigation.

2.1 LTE Architecture

4G LTE has emerged from prior mobile telecommunication technologies, beginning with the first-generation mobile system (1G). This was a fairly basic system that only provided mobile voice services through Frequency Division Multiple Access (FDMA) for analog radio transmissions [17]. The second-generation cellular networks (2G) introduced digitization and compression of speech to increase the capacity of mobile users. 2G includes time and code multiplexing by using the Global System for Mobile Communications (GSM) and CDMA standards [17]. As the use of mobile phones began to grow rapidly, the technology continued to improve with third generation cellular networks (3G). A significant difference between 3G and 2G is the use of packet switching rather than circuit switching for data transmissions. Along with faster data speeds, 3G was also an improvement on prior technologies because it is the first technology to provide Internet access to mobile users. The current 4G LTE offers more bandwidth than 3G, resulting in faster data rates with peak download speeds of 100 Mbit/s and peak upload speeds of 50 Mbit/s [17]. A timeline visualizing this transition from 1G technology to 4G technology is shown in Figure 2.1.

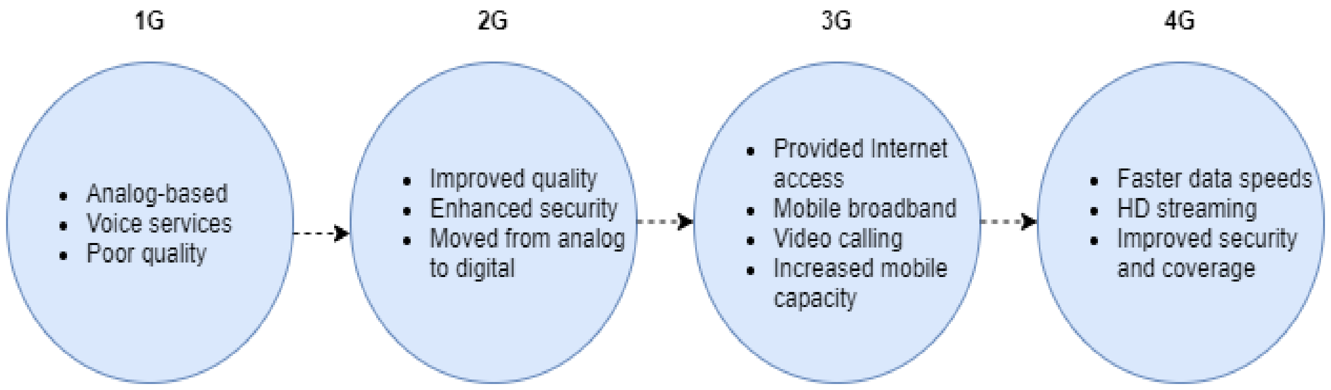


Figure 2.1: Timeline of Cellular Technology. Adapted from [18]

At a high level, a 4G LTE network is comprised of three main components: the UE, the Evolved Packet Core (EPC), and the eNodeB. Figure 2.2 illustrates the infrastructure, along with components of the EPC.

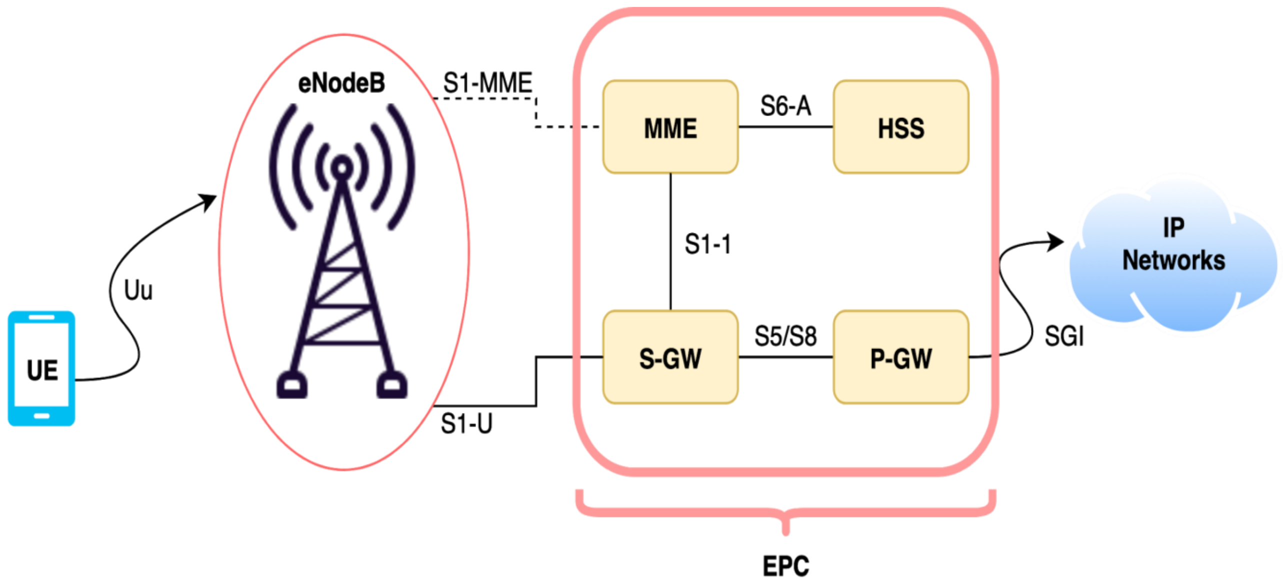


Figure 2.2: LTE Architecture Overview. Adapted from [19]

As shown in the diagram, the UE connects to the eNodeB, which performs dynamic resource allocation. The S-connections act as an interface between the two entities to which they are connected [19]. The EPC is responsible for the connection to the internet and is comprised of the Mobility Management Entity (MME), the Home Subscriber Server (HSS), the Serving

Gateway (SGW), and the Packet Data Network Gateway (PGW) [14]. The MME is the control node that handles the signal processing between the UE and the EPC. The HSS holds the identity of the MME that the user is attached to [21]. The PGW is responsible for IP address allocation for the UE and quality of service (QoS). SGW handles IP packet transfers when UE moves between eNodeBs and keeps the information regarding to the UE in its idle state. Figure 2.2 demonstrates a single eNodeB, but the overall connection between multiples eNodeBs is known as the E-UTRAN [19]. The E-UTRAN handles the radio resource management, header compression, security, and connectivity to the EPC. The roles of each component of the EPC is outlined in Table 2.1.

Table 2.1: Description of EPC Components

Entity	**Description**
Mobile Management Entity (MME)	The main control entity for the E-UTRAN. It communicates with HSS for user authentication and provides UEs with mobility management.
Home Subscriber Server (HSS)	A central database that stores user profiles. It provides user authentication information to the MME.
Serving Gateway (SGW)	It performs the routing and forwarding of user data packets.
Packet Data Network Gateway (PGW)	It gives the UE access to packet data networks, which provides Internet access.

2.2 LTE Physical Layer

This MQP will primarily deal with the LTE PHY layer when analyzing mitigation techniques for jamming. This layer is responsible for the communication between the UE and the eNodeB, which involves both the uplink and downlink. The uplink refers to when the UE sends data to the eNodeB and it uses a Single Carrier Frequency-division Multiple Access (SC-FDMA) signal, which is beneficial in that it reduces the amount of power that is required to be transmitted. This is necessary for the uplink due to the low power capabilities of UEs such as smartphones [22]. For the downlink, which is the connection from the eNodeB to the UE, LTE uses an Orthogonal Frequency-division Multiple Access (OFDMA) scheme. This allows for Multiple In Multiple Out (MIMO) connections in the system, increasing the overall throughput [22]. This MQP analyzes jamming mitigation techniques on the downlink, and there are many physical channels and signals that can be disrupted from jamming attacks. A channel refers to resource elements that carry information originating from higher layers, while a signal does not carry any information from higher layers.

First there are the 4G LTE Downlink Physical Channels. These include: The Physical Downlink Shared Channel (PDSCH), Physical Broadcast Channel (PBCH), Physical Multicast Channel (PMCH), Physical Control Format Indicator Channel (PCFICH), Physical Downlink Control Channel (PDCCH), Physical Hybrid ARQ Indicator Channel (PHICH), and Master Information Block (MIB). These channels are responsible for carrying data from the higher layers and are the interface between 3GPP standards. A description of each channel is described in Table 2.2.

Table 2.2: 4G LTE Downlink Physical Channels [23]

Channel	**Description**
Physical Downlink Shared Channel (PDSCH)	Carries data and signaling messages from Downlink Shared Channel (DL-SCH) and paging messages from Paging Channel (PCH).
Physical Broadcast Channel (PBCH)	Carries the Master Information Block from the Broadcast Channel.
Physical Multicast Channel (PMCH)	Carries multimedia broadcast/multicast service data of the Multicast Channel.
Physical Control Format Indicator Channel (PCFICH)	Carries organization of data and control information in the downlink.
Physical Downlink Control Channel (PDCCH)	Carries scheduling commands and grants.
Physical Hybrid ARQ Indicator Channel (PHICH)	Carries information that indicates an acknowledgement or negative acknowledgement for the UE to determine if a retransmission is necessary.
Master Information Block (MIB)	Carriers system information including the bandwidth and frame number.

Next there are the 4G LTE Downlink Physical Signals. These include: Primary Synchronization Signal (PSS) and Second Synchronization Signal (SSS), Cell-specific Reference Signal (C-RS), MBSFN Reference Signal (MBSFN-RS), UE-Specific Reference Signal (UE-RS), and Positioning Reference Signal (P-RS). These signals do not contain information coming from the higher layers. A description of each signal is described in Table 2.3.

Table 2.3: 4G LTE Downlink Physical Signals [23]

Signal	**Description**
Primary Synchronization Signal (PSS) and Second Synchronization Signal (SSS)	PSS and SSS each carry information used by the UE to obtain cell identity.
Cell-specific Reference Signal (C-RS)	Sent by the eNodeB to support channel estimation at the UE.
MBSFN Reference Signal (MBSFN-RS)	Reference signal for Multimedia Broadcast Multicast Service and is used for channel estimation.
UE-Specific Reference Signal (UE-RS)	Sent to the UEs for channel estimation.
Positioning Reference Signal (P-RS)	Sent by the eNodeB to support location-based services.

The uplink connection between the eNodeB and UE has a set of physical channels and signals, similar to the downlink. The 4G LTE Uplink Physical Channels are: Physical Uplink Shared Channel (PUSCH), Physical Uplink Control Channel (PUCCH), and Physical Random-Access Channel (PRACH). These channels contain resource elements that come from the higher levels (similar to the downlink). An in-depth description of these channels is seen in Table 2.4.

Table 2.4: 4G LTE Uplink Physical Channels [23]

Channel	**Description**
Physical Uplink Shared Channel (PUSCH)	Carries data and signaling messages and Uplink Control Information (UCI) when the UE is transmitting data.
Physical Uplink Control Channel (PUCCH)	Carries UCI when the UE is only transmitting control information.
Physical Random-Access Channel (PRACH)	Carries random access transmissions from the Random-Access Channel.

Finally, the LTE Uplink Physical Signals are: Demodulation Reference Signal (D-RS) and

Sounding Reference Signal (S-RS). These two signals are used by the physical layer and do not actually carry information from higher layers, similarly to the downlink. Table 4.5 discusses these signals in detail.

Table 2.5: 4G LTE Uplink Physical Signals [23]

Signal	**Description**
Demodulation Reference Signal (D-RS)	Sent by the UE to the eNodeB for channel estimation.
Sounding Reference Signal (S-RS)	Sent by the UE and configured by the eNodeB for power reference to support frequency dependent scheduling.

The physical uplink and downlink channels/signals are depicted through the use of a frame structure. LTE uses two types of frame structure: Frequency Division Duplexing (FDD), which separates the uplink and downlink by frequency, and Time Division Duplexing (TDD), which separates uplink and downlink by time [24]. Figure 2.3 depicts an FDD frame structure.

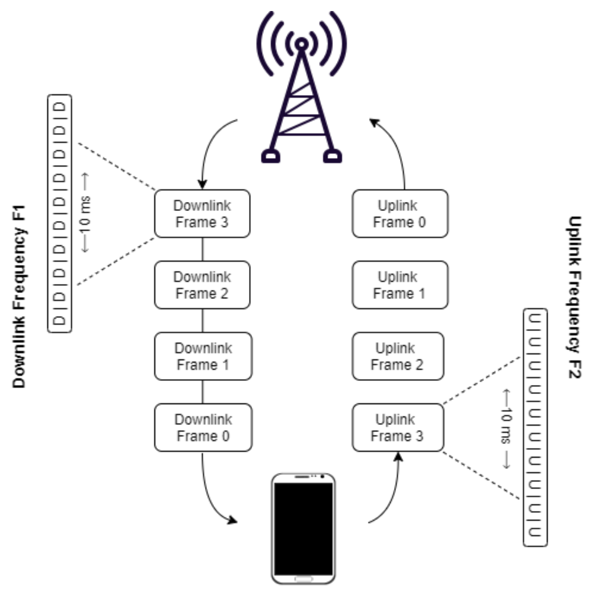


Figure 2.3: LTE FDD Frame Structure. Adapted from [26]

Each transmitted frame is 10 ms long and is broken up into ten 1 ms subframes containing two 0.5 ms slots each. In an OFDMA scheme, a resource block is a 1 slot long time slice and 180 kHz wide in frequency [24]. The subframes contain subcarriers that carry the uplink and downlink signals. Figure 2.4 shows an example of a single LTE subframe. The signals in the subframe occupy various slots when being transmitted to a receiver.

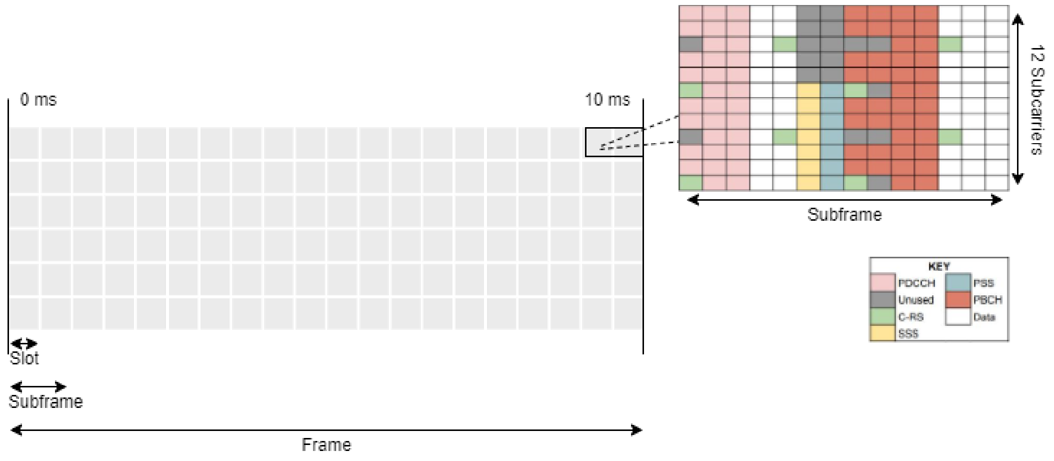


Figure 2.4: LTE Resource Block. Adapted from [27]

These symbols carry the uplink and downlink channels and signals previously described. Figure 2.4 shows how the channels and signals are transmitted within the resource block and how each symbol is either reserved or unused. Determining the exact frequency and time slices that key LTE signals are sent at enables jammers to target significant parts of the downlink that can drastically affect the whole network connection [25]. Frequency hopping randomly targets these signals and effectively causes DoS.

2.3 LTE Error Detection

LTE technology has countermeasures for dealing with error, which occurs when the transmitted data is different from the received data. To detect this, there is a cyclic redundancy check (CRC) that looks for any issues in bit sequences and the 3GPP-LTE standard uses a 24-bit

CRC [29]. These check bits are appended to the end of the transmitted data to make the entire sequence divisible by a predetermined binary number [29]. The receiver then divides the acquired bits by this number from which two scenarios can occur. If there is a remainder, the received data is rejected and must be retransmitted, but if the numbers are divisible, the data is assumed to be correct and is therefore accepted. Figure 2.5 shows a conceptual flow diagram of the sender and receiver using CRC to detect corrupt transmission [29].

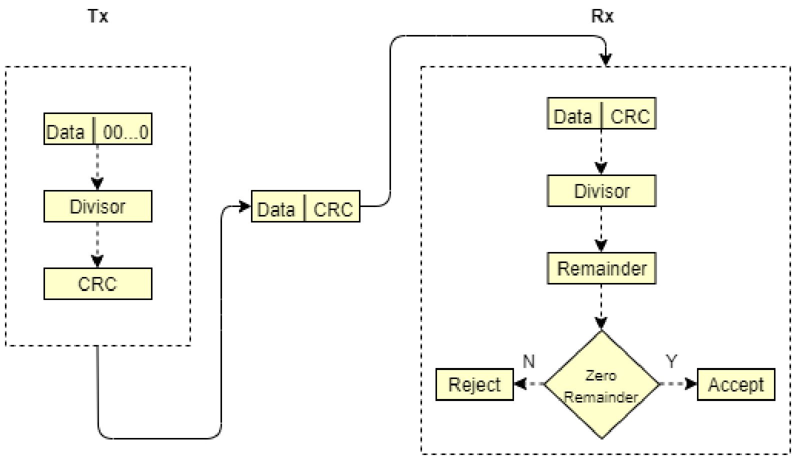


Figure 2.5: CRC Flow Diagram. Adapted from [29]

The method of error detection via the CRC method is very simple to implement and is suitable for determining data that was corrupted by noise. However, the CRC does not have the ability to detect intentional interference not caused by noise, and exposes a large vulnerability in the LTE system. The narrow-band jamming, which would not be detected by CRC, that this MQP analyzes is frequency hopping. Here, high-power signals will transmit at random times, disrupting key signals and the network as a whole. As a result, additional mitigation techniques are necessary to successfully alert the eNodeB and to avoid a jamming attack, and this MQP explores a potential solution.

2.4 OpenAirInterface

OAI is an open-source platform that provides software and hardware tools for use in 3GPP networks (*e.g.* 3G and LTE). It was developed by the Mobile Communications Department at EURCOM and offers real-time transceiver applications (*e.g.* eNodeB, UE, EPC) through the use of software defined radios [30]. As OAI is open-source, it includes a GitHub code repository to all the components required to completely set up an LTE testbed with a working eNodeB and UE [28]. The software packages include a fully functional 4G LTE EPC which contains the MME, HSS, and SPGW. It also provides the E-UTRAN software to be used by the SDR to get the eNodeB working. All of the software is designed to be used across a Linux-based system with USRP functionality. Within the software packages themselves, high-level emulation modes are also available which offer realistic and controllable experimentation. The purpose of providing all of these elements to the user is to ease the development of a real-time stack with software-defined radios [30].

In order to use the code repository correctly and use it to its full potential, it is important to understand the software packages and what is and is not offered in them. Table 2.6 shows the OAI software implementations of the LTE framework.

Table 2.6: Software Offerings for OAI [30]

LTE Framework	**OAI Full Software Implementations of:**
Downlink Channels	PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH
Uplink Channels	PRACH, PUSCH, PUCCH, SRS, DRS
Physical Layer	⇒ MAC, RLC, PDCP, RRC, MCH, MCCH, MTCH ⇒ Priority-based MAC scheduler with dynamic MCS selection ⇒ Fully reconfigurable protocol stack ⇒ Standard S1AP and GTP-U interfaces to the Core Network ⇒ IPv4 and IPv6
Core Network	SGW, PGW, MME & HSS
UE	⇒ UE handling procedures: attach, authentication, service access, radio bearer establishment ⇒ Transparent access to the IP network (no external SGW or PGW are necessary). Configurable access point name, IP range, DNS and E-RAB QoS
Hardware	USRPs (many different variants, but this project used B200, B210/N210, X310)

A flow diagram describing the software package layout of OAI is shown in Figure 2.6. Essentially, it is showing how there are three main components towards implementing OAI. First, the UE must be correctly configured and is adjustable to multiple UEs, as long as everything is programmed correctly on the UE. Second, there is the eNodeB component, where the diagram depicts all the software adjustable and available options for the eNodeB. The final component is the EPC that shows all of its programmable parts as well.

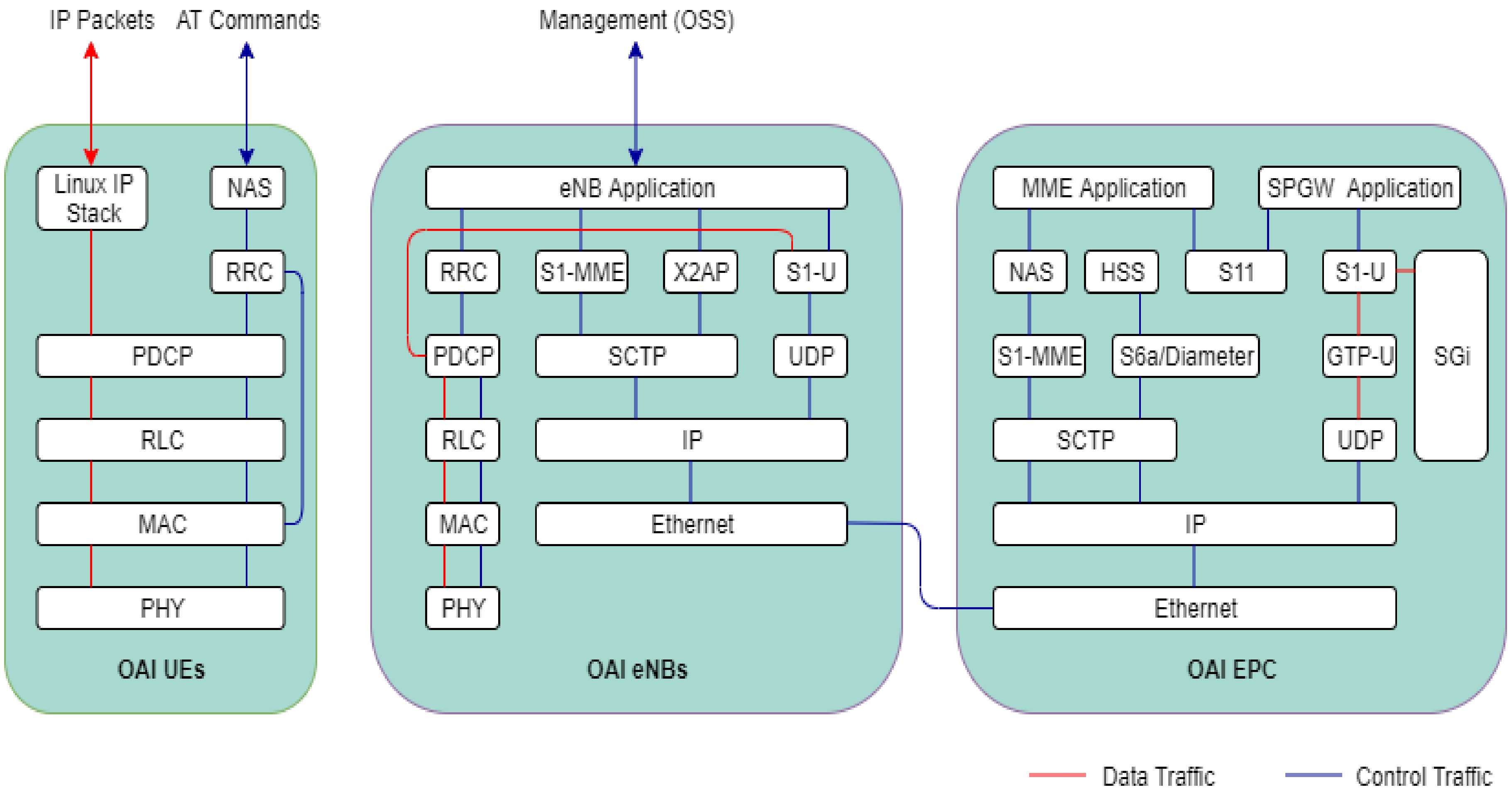


Figure 2.6: OAI Software Stack Flow Diagram. Adapted from [30]

2.5 Software-Defined Radios and GNU Radio

In order to build an LTE testbed that is fully operational, it is necessary not only to have the software required but to also obtain optimal hardware. The hardware that is used to build an eNodeB and other components for an LTE testbed consist of an SDR, which is a radio where all the physical layer functions are defined through software [31]. The ability to fully adjust the hardware properties is significant and due to this property, SDR has become the leading technology for researching and developing advanced communication systems.

Figures 2.7 and 2.8 show block diagrams of SDR functionality for receiving and transmitting applications, respectively. On the receive side, there is first an antenna to pick up the transmitted signal, and an RF tuner that converts that analog signal to IF. The IF signal is then fed into the analog to digital converter and the outputted samples are sent into the digital down converter (DDC). At the DDC, the digital mixer and local oscillator shift the IF samples to baseband and the low pass filter (LPF) confines the bandwidth of the final signal. The output of the DDC is then sent into the processing block that can be done with one or a combination of DSPs, FPGAs, and others. The transmitter of an SDR acts in the similar way of the receiver, but does the reverse process, as shown in Figure 2.8.

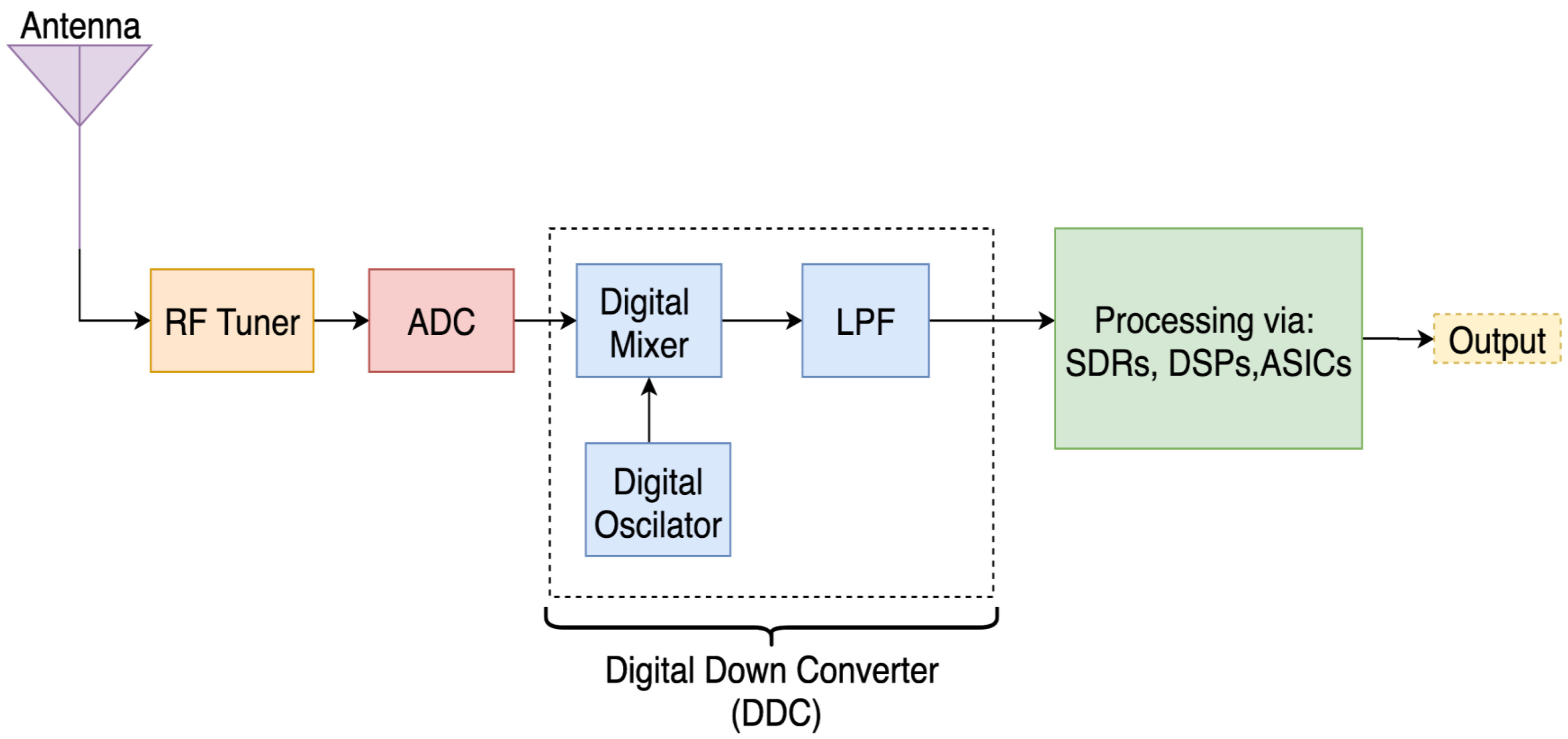

Antenna
RF Tuner
ADC
Digital Mixer
LPF
Digital Oscilator
Processing via: SDRs, DSPs,ASICs
Output
Digital Down Converter (DDC)

Figure 2.7: Block Diagram of the Receiving Functionality of SDR. Adapted from [32]

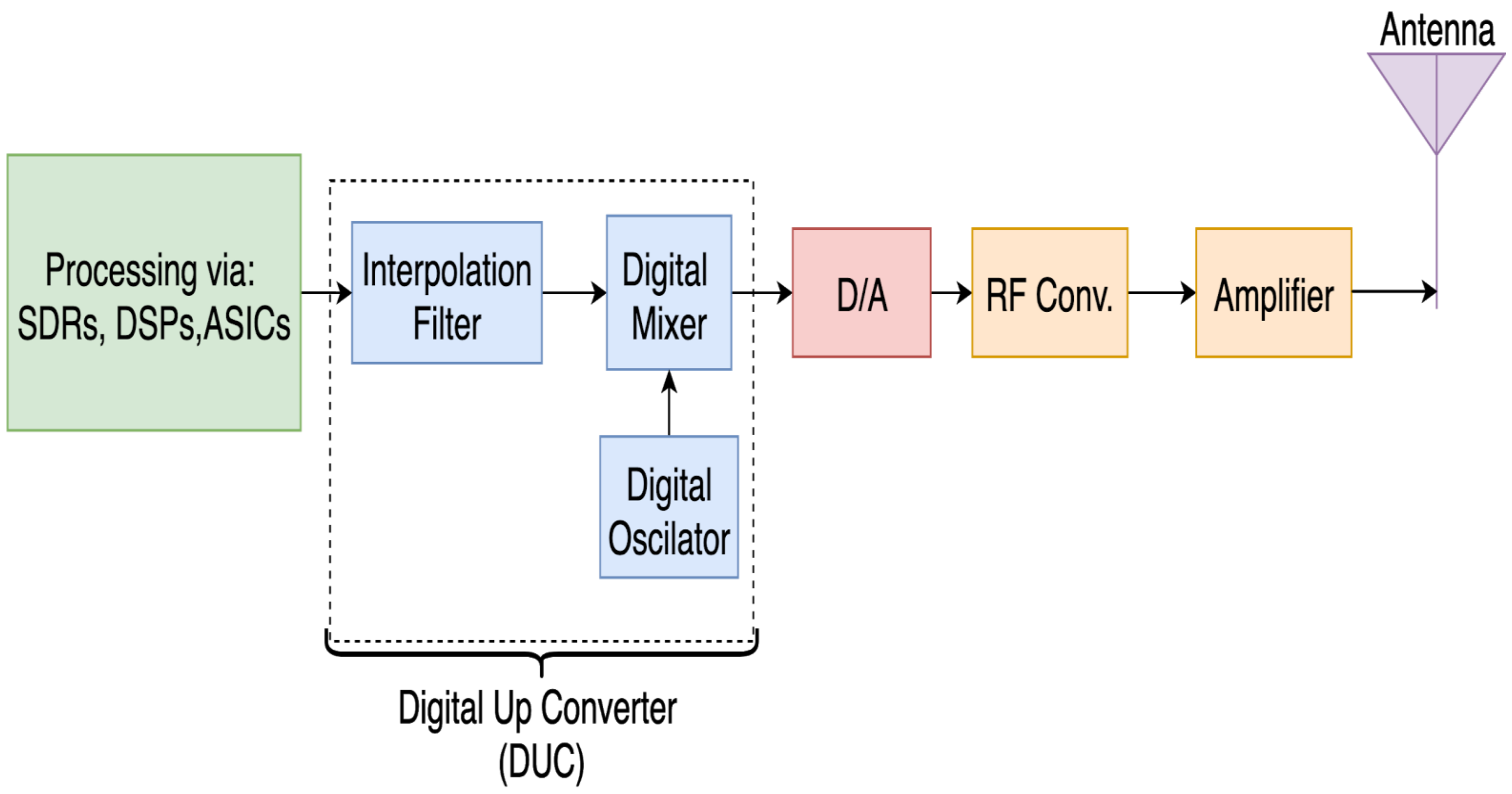


Figure 2.8: Block Diagram of the Transmission Functionality of SDR. Adapted from [32]

An SDR is a radio that contains an RF front end where there is typically a transmit and receive chain that can operate seamlessly together. All of its radio functions are implemented through the use of programmable processing components that include Field-programmable Gate Arrays (FPGAs), Digital Signal Processors (DSPs), General Purpose Processors (GPPs), and Systems on Chip (SoC). This allows an SDR to easily modify its parameters without any physical intervention [33].

One of the industry leaders in SDR development is Ettus Research, which is a subsidiary of National Instruments [34]. Their product line includes types of SDRs for different RF applications that range from DC to 6 GHz, and are the platform of choice for many different engineers and scientists in the wireless community.

In order to interact with and use SDRs, it is important to understand that there are many different implementations, each having pros and cons. This project chose to use the OAI platform due to ease of use and the team being very familiar with C/C++ and Python. Also, the ability to use GNU Radio/Python as the software development platform was favorable. GNU Radio is an open-source toolkit that uses signal processing blocks to program an SDR. It handles all the digital data and can be used to create demodulators, filters, decoders, spectrograms, and many others. It provides a user-friendly graphic user interface (GUI) for the user and is very powerful in its features. GNU Radio also generates a Python file from the flowgraph that can be modified based on user needs. For example, if you so choose to do something that cannot be done with the signal processing blocks available, you can create your own block to perform the task you desire your SDR to achieve [34].

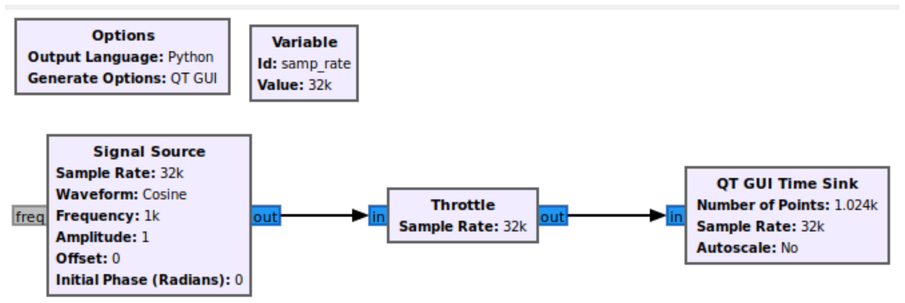


Figure 2.9: GNU Radio Flowgraph Example [35]

2.6 Chapter Summary

This chapter explained the evolution of cellular data technology and the current fourth generation LTE. The architecture of an LTE system was detailed and the physical layer channels and signals were outlined, along with current error detection methods implemented in the technology. This section concludes with an explanation of the different components that can be used to implement an LTE system, including SDRs and OpenAirInterface.

3. Proposed Approach

This chapter described the MQP team's proposed approach, including the outlined process for developing an LTE testbed as well as exploring jamming mitigation techniques. Moreover, the project scheduling and division of labor between teammates is described, along with key milestones that were identified. The proposed metrics for the project's success are defined and evaluation methods are explained.

3.1 Project Scope and Metrics for Success

DoS attacks can be difficult to defend against, and this MQP explores mitigation techniques regarding real-time jamming detection and feedback to the eNodeB in the event of narrow-band jamming that corrupts key downlink signals. This project will explore the PHY layer of the LTE system and specifically monitor the downlink signal of the base station. The anti-jamming method will not be able to defend against attacks targeted at the uplink signal. Moreover, the scope of this project will not explore security for the application and transport layers, as it will solely focus on the received power from the jammer.

The success of this project is determined based on how quickly the code can process the signal and alert the eNodeB of detected jamming. This will be assessed based on metrics from Wireshark, in which the amount of dropped packets will be monitored. The MQP seeks to evaluate how quickly the system detects the presence of an unwanted signal and whether or not the base station is able to evade the jamming attack. If this is accomplished, the UE should be able to reestablish the connection with the eNodeB and this will be verified by analyzing the packet receiving rate before and after the jammer runs.

3.2 Anti-Jamming Method

Developing jamming detection and mitigation schemes to be implemented in real-time requires a process that detects the occurrence of jamming and acts to correct it with very little latency. The goal of the project is to create a system that can recognize when jamming is present and successfully bypasses it without causing a noticeable disruption to the user. Please note that the jamming has been accomplished previously by two other group members and is already implemented through the use of random frequency hopping [25]. In order to solve the issue of jamming, there are many different techniques to choose from. One potential solution is to use machine learning to learn the random pattern of the jamming and predict which subcarrier is going to be targeted and prevent the interference. However, due to time constraints and limited resources this quickly became an unattainable goal.

There is also the possibility of using an FPGA to handle all the data processing and using it to determine if there is a jamming source present and how to fix the jamming. After researching the current state of the art regarding this topic, it was determined that this complexity was not required, as one can handle all the data processing in Python with a similar runtime as that of an FPGA for this application. Hence, using GNU Radio flowgraphs was the approach that was ultimately chosen. This allows for the capturing of all the in-phase and quadrature (IQ) data by a separate SDR. Using this method presents two different options: (1) Exporting the data into a file to then be read and processed, or (2) modify the generated Python file from the flowgraph to process the data directly and return an output.

After investigating these possibilities, it was determined that using an external file would provide a very heavy computational load on the Linux machine and therefore fail to perform in real-time. As a result, the proposed solution involves creating a flowgraph within GNU Radio and expanding on the Python file. The Fast Fourier Transform (FFT) plot of the received signal can be used to find the power levels and iterating through these values will allow for the detection of any unwanted interference. If this is the case, then the SDR can send a signal to notify the base station to change the frequency that it is transmitting on. Once a signal is sent, then the SDR will revert to acting as a receiver. However, if there is no jamming occurring, then the SDR will not transmit a signal and the base station will continue to run normally. Figure 3.1 depicts a logic diagram for this project's proposed approach.

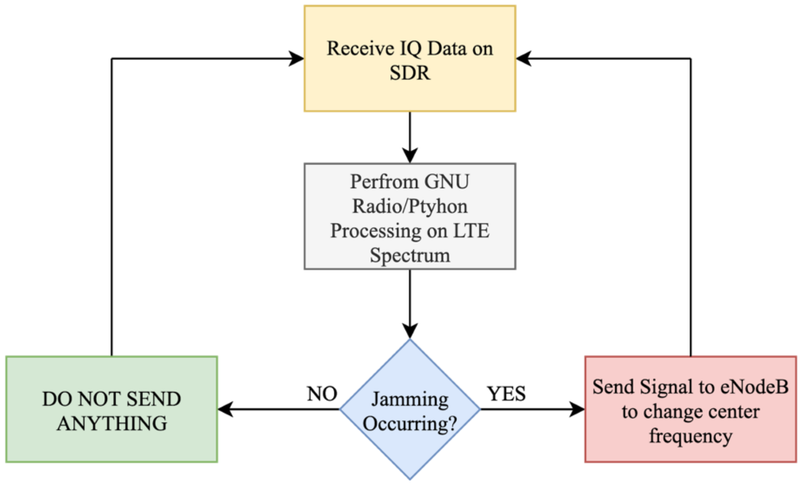

Receive IQ Data on SDR
Perfrom GNU Radio/Ptyhon Processing on LTE Spectrum
Jamming Occurring?
NO
YES
DO NOT SEND ANYTHING
Send Signal to eNodeB to change center frequency

Figure 3.1: Logic Diagram for Proposed Approach

A successful demonstration will be one that has an established LTE system, a jammer that performs narrow-band jamming, and a radio that detects jamming signals and alerts the eNodeB in real-time to mitigate DoS from the UE. Figure 3.2 shows a concept diagram of the final testbed, that includes: a B210 acting as the eNodeB, a Samsung S4 acting as the UE, an N210 acting as the jammer, and an X310 acting as the jammer detector.

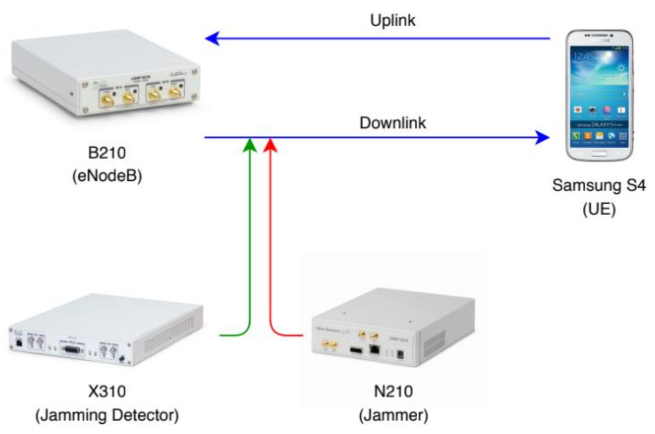


Figure 3.2: Concept Diagram of LTE Jamming Detection Testbed. USRP Images Taken from [25]

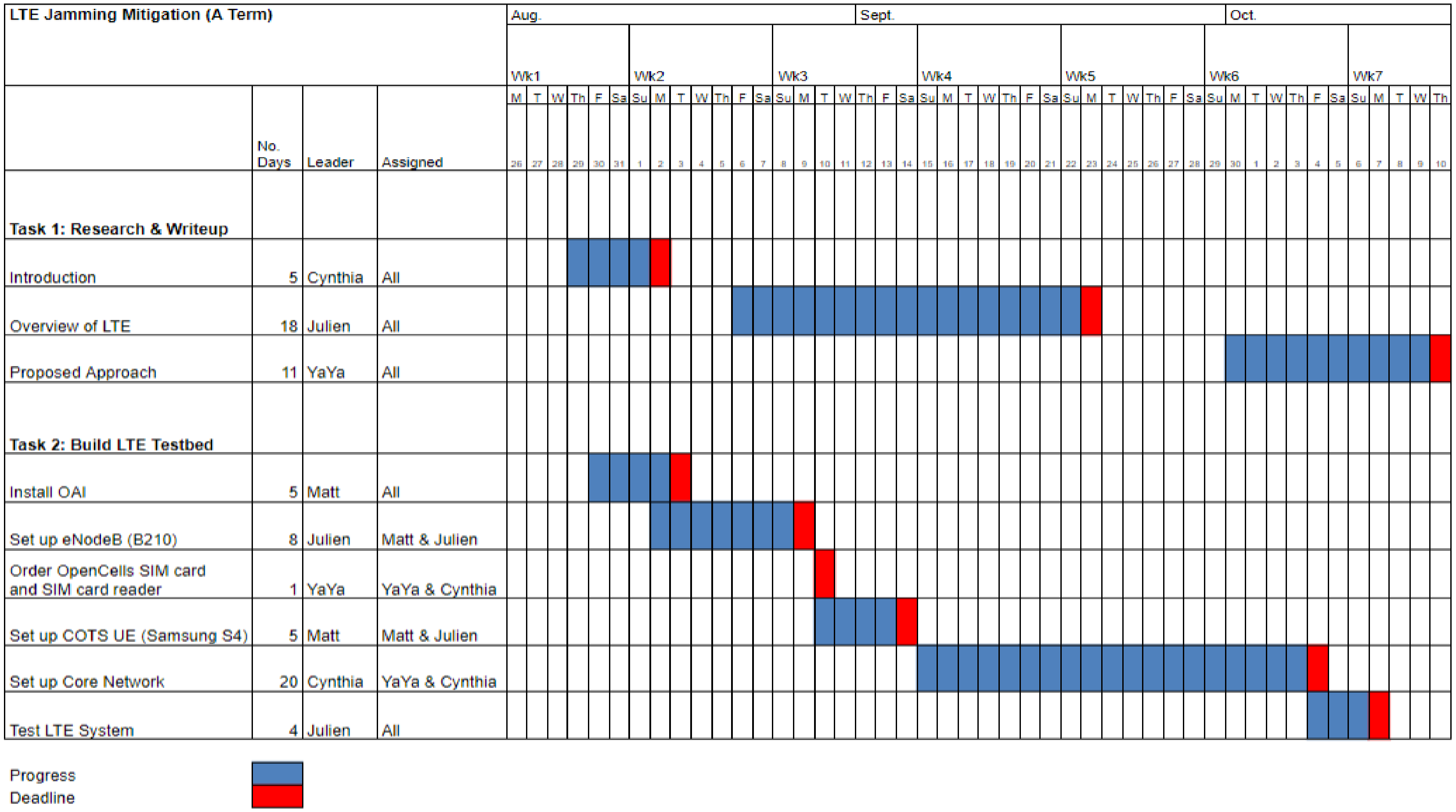
3.3 Project Planning

The project tasks and milestones were outlined using Gantt Charts to set deadlines for each goal and determine which members were responsible for their completion. The MQP project was divided into three 7-week long stages, corresponding to the terms in WPI's quarter system. The team consisted of 4 initial members for the first two terms and then turned into two members for the third term.

The first term, A term, included the initial stages of the project: the development of an LTE testbed with an LTE signal being transmitted (eNodeB) and a smartphone (UE) receiving the signal and successfully demonstrating full use of it. This was accomplished in a fast and efficient manner by assigning sub-tasks to each team member and having deadlines in order to meet goals. For example, one person was responsible for implementing the LTE EPC, and another person for setting up the eNodeB. Figure 3.3 shows the team's Gantt chart for A term.

In B term, the team split into two sub-teams for the remaining two terms that included a red team responsible for implementing a unique jamming technique to jam the LTE signal and a blue team responsible for creating anti-jamming measures. The details of each task are shown in the Gantt charts in Figures 3.4 and 3.5. The red team implementation is explained in [25]. The blue team approach is discussed in this report.

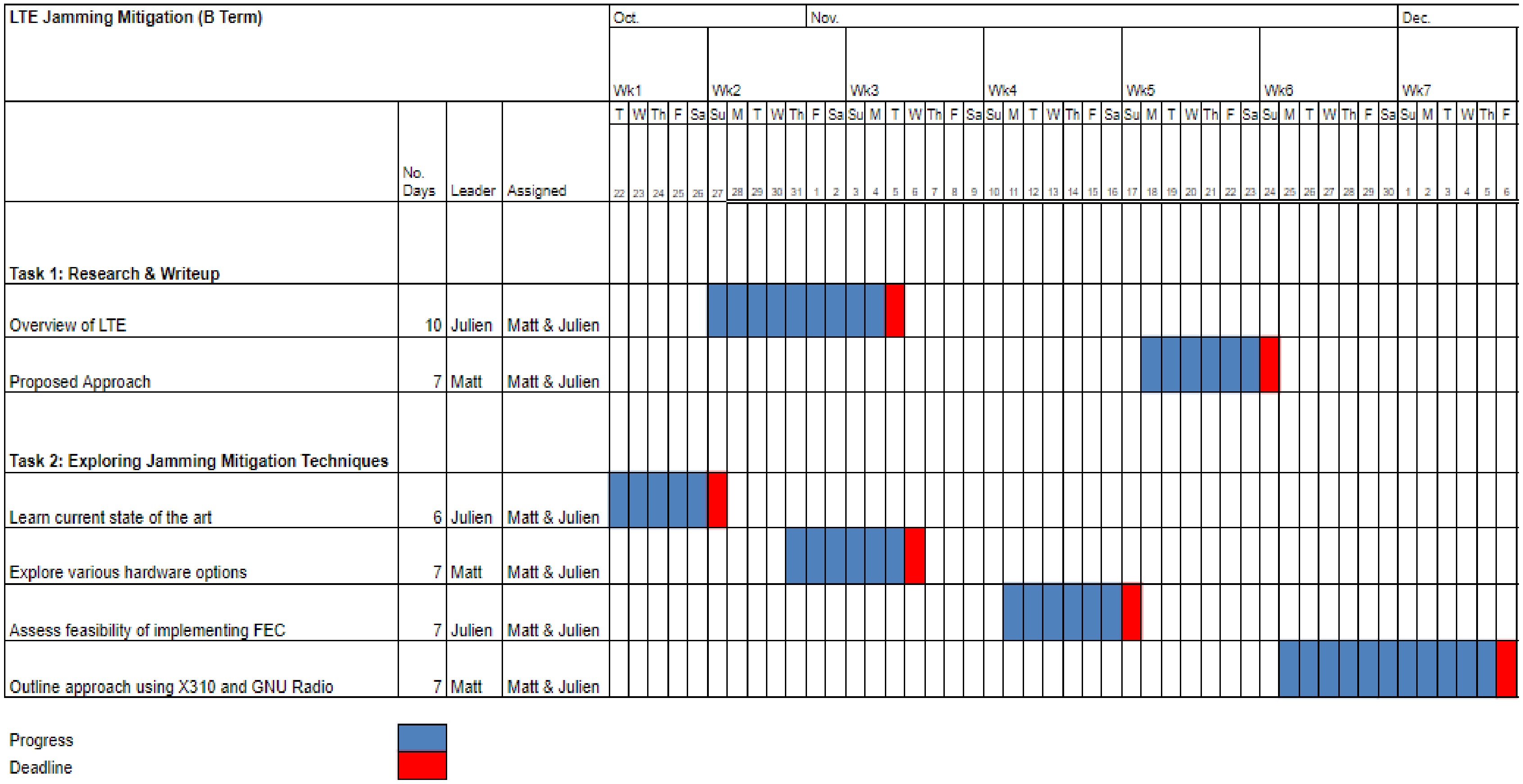
In C term, the blue team implemented the proposed approach of the anti-jamming protocols. The first step was to process the received data with the X310 in real-time, and determine whether or not jamming was occurring across the LTE spectrum. The next step was to interface with the OAI code repository and notify the eNodeB of jamming being present. One issue that arose was detecting jamming in an efficient way, as initially the team struggled to come up with an adaptable way to determine a threshold power level for jamming. Once this was completed, then the eNodeB would change the downlink frequency in real-time. One way that this approach can still be improved is by creating two separate scripts, as opposed to one, where one determines the threshold value alone and outputs that value. The next script would take that output as an input and use it to implement anti-jamming.



LTE Jamming Mitigation (A Term)	No. Days	Leader	Assigned
Task 1: Research & Writeup			
Introduction	5	Cynthia	All
Overview of LTE	18	Julien	All
Proposed Approach	11	YaYa	All
Task 2: Build LTE Testbed			
Install OAI	5	Matt	All
Set up eNodeB (B210)	8	Julien	Matt & Julien
Order OpenCells SIM card and SIM card reader	1	YaYa	YaYa & Cynthia
Set up COTS UE (Samsung S4)	5	Matt	Matt & Julien
Set up Core Network	20	Cynthia	YaYa & Cynthia
Test LTE System	4	Julien	All

Progress	🟦
Deadline	🟥

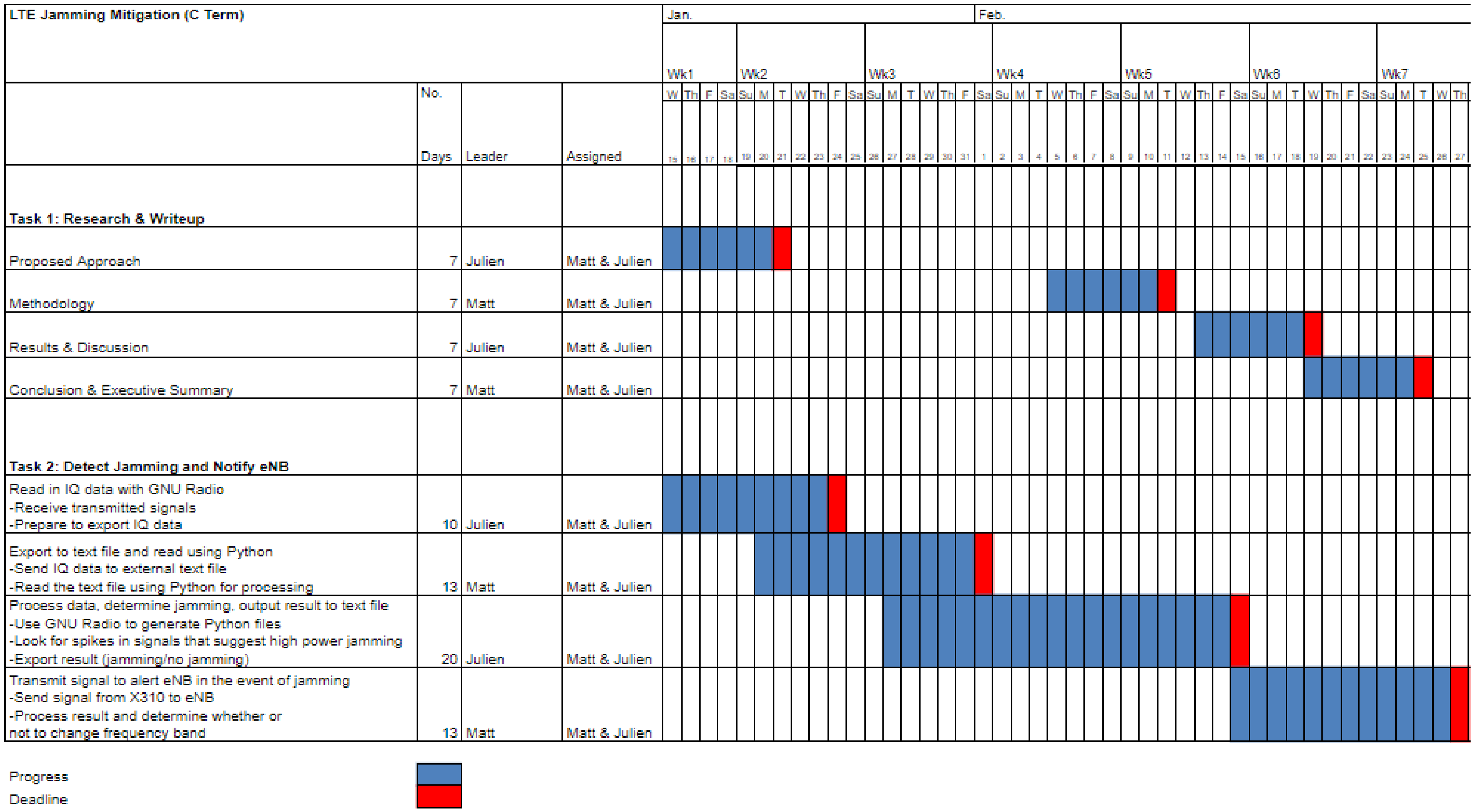
Figure 3.3: A Term Gantt Chart



LTE Jamming Mitigation (B Term)	No. Days	Leader	Assigned
Task 1: Research & Writeup			
Overview of LTE	10	Julien	Matt & Julien
Proposed Approach	7	Matt	Matt & Julien
Task 2: Exploring Jamming Mitigation Techniques			
Learn current state of the art	6	Julien	Matt & Julien
Explore various hardware options	7	Matt	Matt & Julien
Assess feasibility of implementing FEC	7	Julien	Matt & Julien
Outline approach using X310 and GNU Radio	7	Matt	Matt & Julien

Progress	🟦
Deadline	🟥

Figure 3.4: B Term Gantt Chart



LTE Jamming Mitigation (C Term)

	No. Days	Leader	Assigned
Task 1: Research & Writeup			
Proposed Approach	7	Julien	Matt & Julien
Methodology	7	Matt	Matt & Julien
Results & Discussion	7	Julien	Matt & Julien
Conclusion & Executive Summary	7	Matt	Matt & Julien
Task 2: Detect Jamming and Notify eNB			
Read in IQ data with GNU Radio -Receive transmitted signals -Prepare to export IQ data	10	Julien	Matt & Julien
Export to text file and read using Python -Send IQ data to external text file -Read the text file using Python for processing	13	Matt	Matt & Julien
Process data, determine jamming, output result to text file -Use GNU Radio to generate Python files -Look for spikes in signals that suggest high power jamming -Export result (jamming/no jamming)	20	Julien	Matt & Julien
Transmit signal to alert eNB in the event of jamming -Send signal from X310 to eNB -Process result and determine whether or not to change frequency band	13	Matt	Matt & Julien

Progress	🟦
Deadline	🟥

Figure 3.5: C Term Gantt Chart

3.4 Technical Deliverables

Throughout the course of the project, the following will have been delivered:

1. Fully functioning LTE testbed that includes an eNodeB, a UE, and an EPC. The purpose of developing an LTE testbed is to establish a reliable and fast connection between a base station and a cellular phone that can represent a real-time environment. This helped the project deliver an anti-jamming protocol that has the potential to be implemented in the real world, rather than just creating security measures that will have no standing for further development in society.
2. A jamming detector that detects in real-time whether or not jamming is being performed on the LTE spectrum. This is the goal of this project, as LTE use has increased, while DoS attacks have also seen a rise in recent years. It is vital to build more security protocols that go beyond current 3GPP-LTE standards, and this jamming detector will propose one way to do it.
3. eNodeB that randomly hops to a different frequency in LTE Band 7 where there is no jamming occurring. This step is important successfully evade the narrow-band jammer. Having the ability to change to a different LTE downlink frequency is an essential step for this jamming mitigation approach and this system does it very quickly.

3.5 Chapter Summary

This chapter begins by explaining the problems facing LTE security and how this team will evaluate the success of the project. The team's plan for implementing the LTE testbed and jamming detection is outlined using Gantt charts, and a detailed explanation of the milestones is provided so that it is very clear how the jamming will be detected and bypassed. Many approaches were considered during the planning stage, but it was ultimately decided that using an SDR as a receiver and GNU Radio will achieve real-time results.

4. Project Implementation

This section discusses the steps needed to successfully implement anti-jamming protocols into an LTE base station. To establish an LTE system, a testbed was designed and implemented using OAI and consisted of an eNodeB and a UE. A narrow-band jammer was then added to analyze the effects of random frequency hopping [25]. The final step involved utilizing an SDR to perform anti-jamming security measures, which resulted in a mitigation of DoS between the base station and UE.

4.1 Testbed Implementation

The LTE testbed includes a 4-core computer running Linux with Ethernet and USB port connectivity that were used to establish a connection to all the SDRs in the system. A USRP B210 was used to act as the LTE system's eNodeB, a USRP N210 was used to jam the connection, and a USRP X310 was used as the jamming detector. The UE was a Samsung Galaxy S4 and a programmed SIM card from Open-Cells was used to allow for the phone to form an LTE connection with the eNodeB [36]. The LTE testbed was located within a Faraday cage, which shields all RF wireless signals and must be used in order to comply with federal regulations. Figure 4.1 shows the setup of the testbed, and Figure 3.2 shows the concept diagram of the project setup as well.

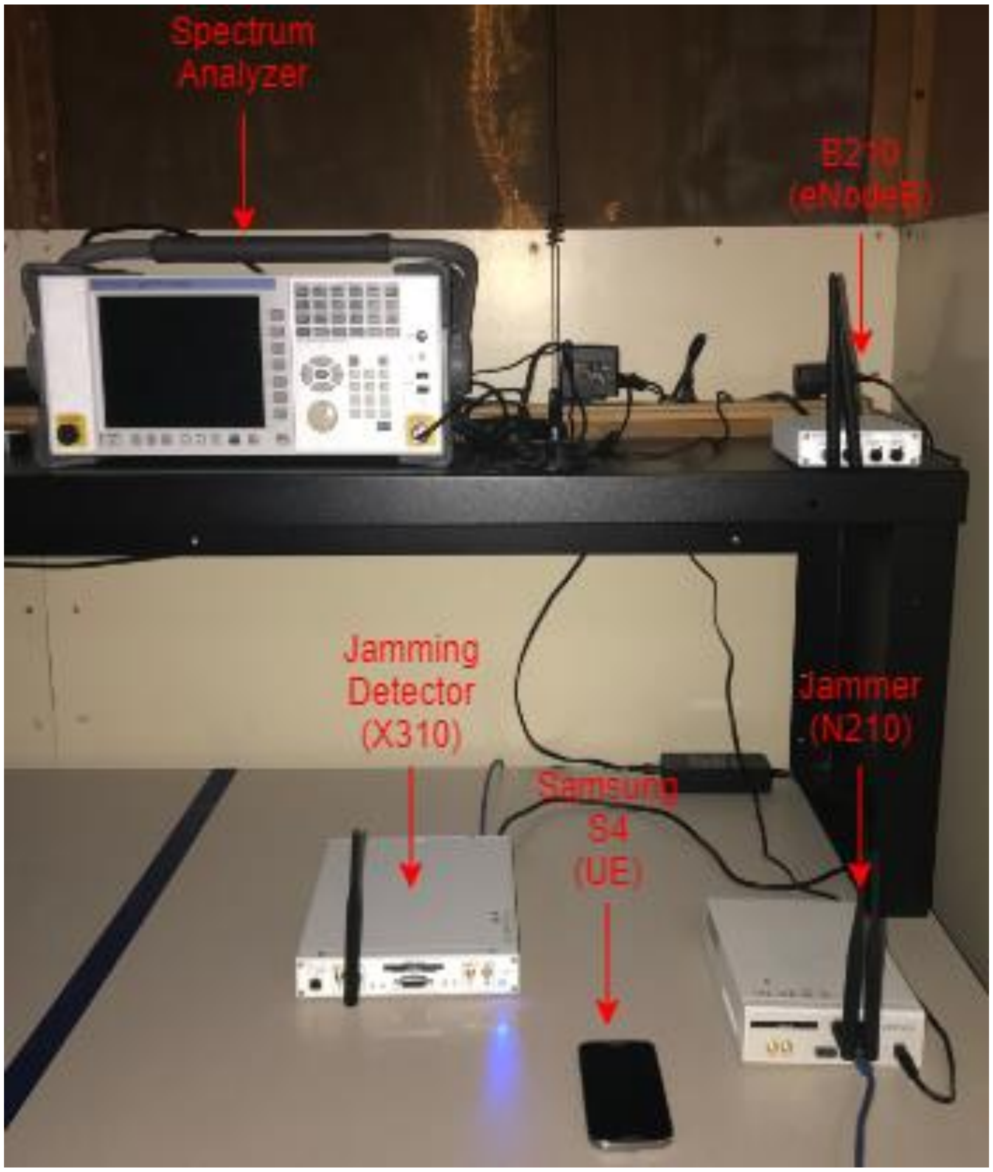


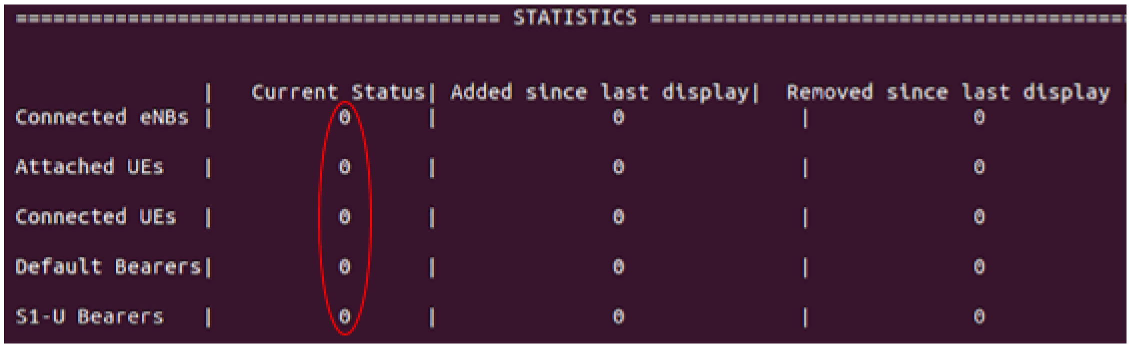
Figure 4.1: Hardware Setup of Testbed in Faraday Cage

Setting up the LTE base station required extensive research and debugging of OAI, as discussed in the background. In order to perform the setup properly with a functioning eNodeB, EPC, and UE, the steps outlined in Table 4.1 were performed. The SDRs in the system and the EPC were all controlled and performed on a four-core Linux machine running on Ubuntu 18.04. First, the eNodeB was setup by copying the GIT repository of *OpenAirInterface5G* to the local machine. All the dependencies were installed and the eNodeB was compiled. Then the EPC was setup on the local machine. The *openair-cn* repository was copied to the local copy of *OpenAirInterface5G* and the software was installed via building the HSS, MME, and SPGW. Lastly, the UE was then formatted properly to connect to the network. This was done by configuring the SIM card and installing it into the phone.

Table 4.1: LTE Testbed Setup Process. Adapted from [36] and [37]

eNodeB Steps	EPC/Core Network Steps	UE Steps
1. Copy the Git Repository, *OpenAirInterface5G,* to the local Linux Machine 2. On the local Linux Machine: a. Install the dependencies b. Compile the eNodeB 3. Edit the configuration file for the eNodeB (*i.e.* change the center frequency to 2.68 GHz, and setup the correct SDR to act as the base station) 4. Everything is all set, and the LTE base station can now be run	1. Copy the Git Repository for openair-cn from the OAI repository 2. Install the software for the EPC: a. Build the HSS b. Build the MME c. Build the SPGW 3. The EPC is setup and ready for use	1. Connect the SIM Card to your computer using the UICC tool provided by opencells 2. Program the SIM Card with an: adm, authentication key, opc, and spn 3. Remove SIM card from opencells board and place into LTE compatible phone 4. Setup the APN protocol on the phone to IPv4 and the bearer to LTE 5. The phone (UE) is now ready for integration

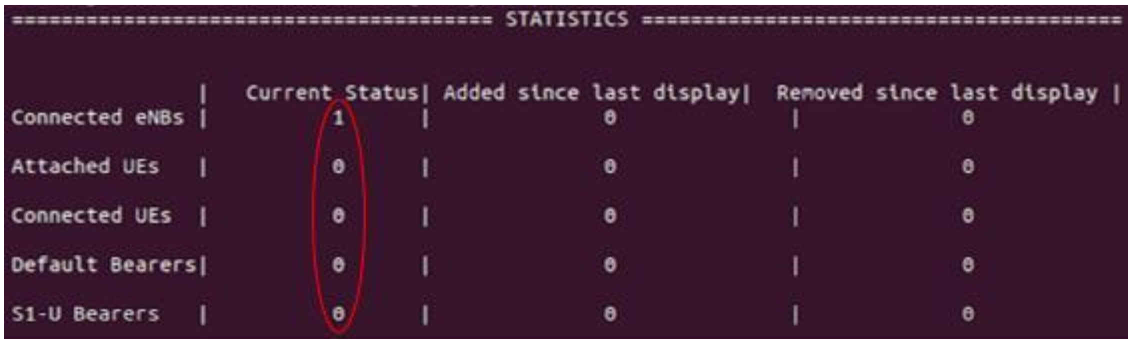
The team referenced Chance Tarver's OAI Tutorial for the development of the eNodeB and EPC and Open-Cells to complete the programming of the UE. Once all of these steps were completed, the LTE testbed became fully functional. The EPC was then run in the command terminal by using the commands '*./run_hss*', '*./run_mme*', and '*./run_spgw*' in the openair-cn directory. The eNodeB was run in a similar way using the command '*cmake_targets/lte_build_oai/build. /lte-softmodem -O ~/opencells-mods/enb. 10MHz.b200*'. The eNodeB would transmit the downlink signal at the user designated 2.68 GHz frequency. As soon as the eNodeB was running, the phone connected to the base station and the team was able to browse the Internet, seeing a successful LTE connection. When the base station was off, there were no connected eNodeB according to the MME statistics, shown in Figure 4.2. This changed and one eNodeB was detected when the B210 began transmitting, shown in Figure 4.3. Once the UE connected to the base station, the MME gave the statistics shown in Figure 4.4, showing a successful LTE connection.



```
======================================= STATISTICS =======================================

                 |   Current Status| Added since last display|  Removed since last display
Connected eNBs   |         0       |           0              |            0
Attached UEs     |         0       |           0              |            0
Connected UEs    |         0       |           0              |            0
Default Bearers  |         0       |           0              |            0
S1-U Bearers     |         0       |           0              |            0
```

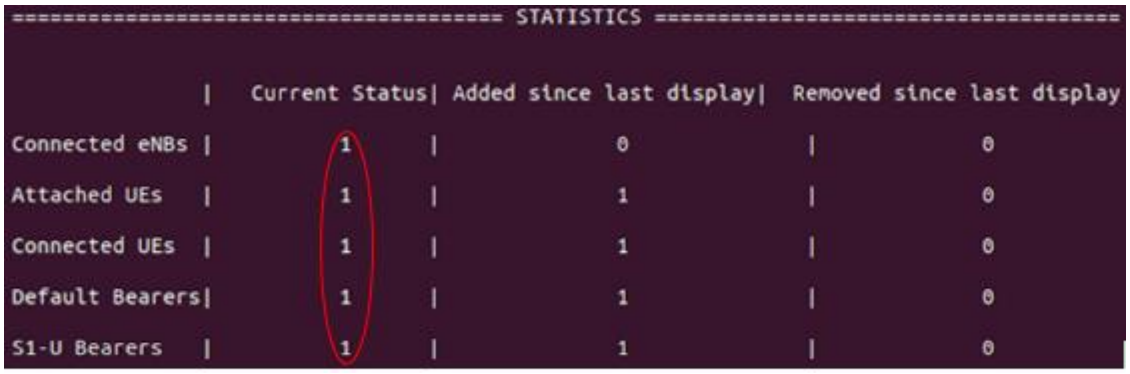
Figure 4.2: MME Statistics without eNodeB Connection



```
======================================= STATISTICS =======================================

                 |   Current Status| Added since last display|  Removed since last display |
Connected eNBs   |         1       |           0              |            0
Attached UEs     |         0       |           0              |            0
Connected UEs    |         0       |           0              |            0
Default Bearers  |         0       |           0              |            0
S1-U Bearers     |         0       |           0              |            0
```

Figure 4.3: MME Statistics with eNodeB Connection



```
====================================== STATISTICS ======================================

                |   Current Status| Added since last display|  Removed since last display
Connected eNBs  |          1      |            0            |            0
Attached UEs    |          1      |            1            |            0
Connected UEs   |          1      |            1            |            0
Default Bearers|           1      |            1            |            0
S1-U Bearers    |          1      |            1            |            0
```

Figure 4.4: MME Statistics with eNodeB and UE Connection

In order to develop security measures to defend the downlink signal from jamming attacks, a jammer was created and introduced to the LTE system. An in-depth explanation of the creation of the random frequency hopping jammer that was used in this project is outlined in [25]. Once this jammer disrupted the system, the team needed to detect its presence and notify the eNodeB.

4.2 Jamming Detection

Before the eNodeB could successfully bypass the jamming, the team first analyzed the behavior of the N210 to understand how it would disrupt random subcarriers. Using GNU Radio, a flowgraph was created to display the FFT plot and spectrogram at the center frequency of the downlink signal. Figure 4.5 shows the flowgraph implementation. The team used the USRP X310 to act as a receiver and another GNU Radio flowgraph to implement a 1024-point FFT for the 10MHz wide downlink signal. This means that each of the 1024 bins were 9.765kHz wide and were searched through by probing the results, which returned the power levels at each bin. The first block, Stream to Vector, was used to concatenate all the IQ data into one vector, which was then provided as an input to the FFT block. Once this FFT provided a result, the Complex to Mag^2 block and Log10 block were used to compute the power of the signal. This is the value that was probed for the jamming detection.

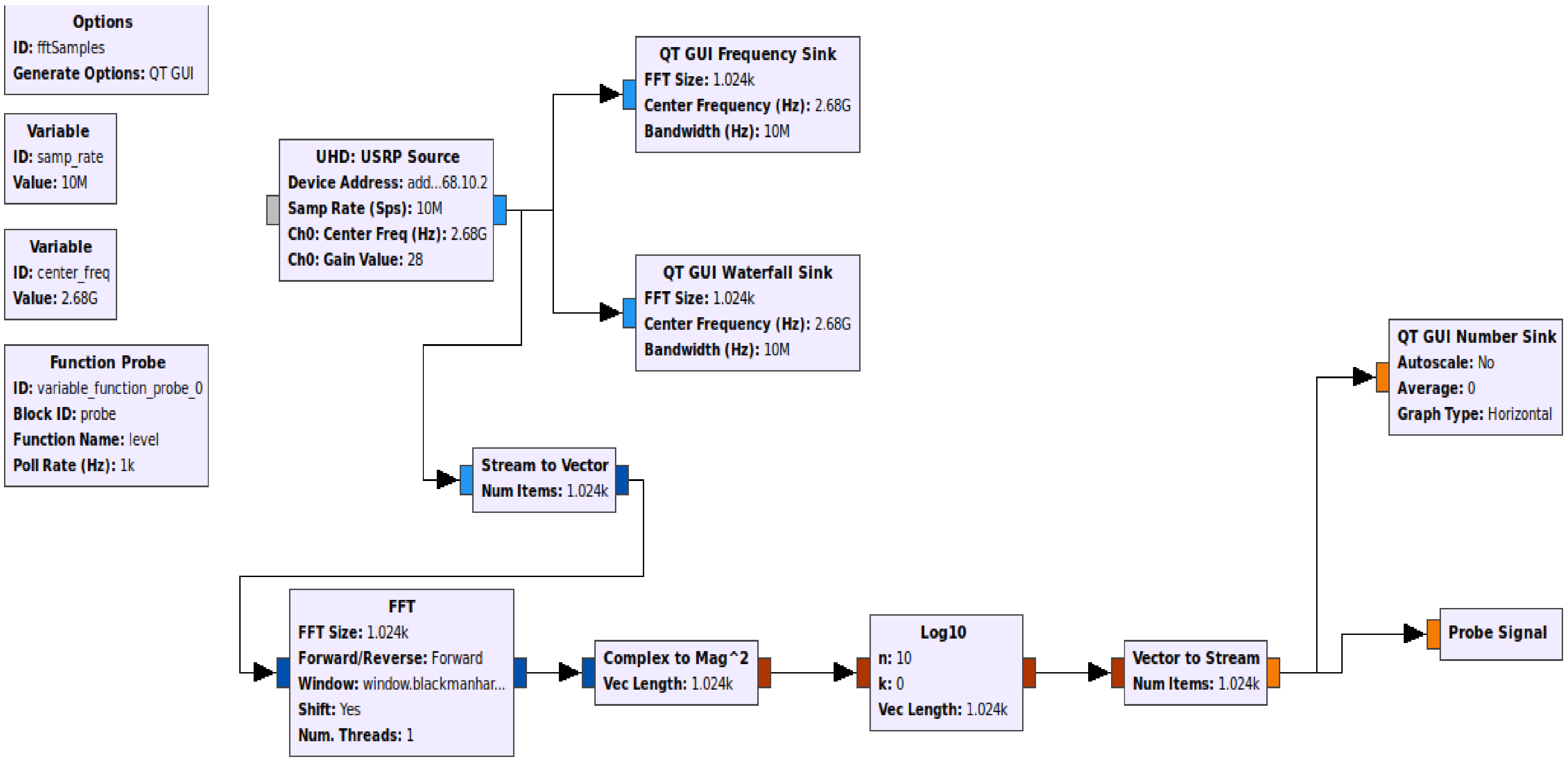


Figure 4.5: GNU Radio Flowgraph for Monitoring LTE Downlink Signal

The method for detecting jamming was to continuously check this value in real-time to see if it increases above a certain threshold, signifying the presence of an unwanted signal. The default rate at which the power level is probed is 0.2 Hz, resulting in a return value every 5 seconds, and this was increased to 10 Hz to sample the power every 0.1s.

To determine the necessary threshold value, the team added code to the GNU Radio generated Python script to monitor the power levels of the downlink signal and record a sufficient threshold. Once the base station began transmitting, the X310 would receive the power levels and then keep track of the first 5000 readings, which took about 5 seconds to complete. The maximum value of this list was then used as a threshold, as any power level above this was not consistent with the LTE signal and it was determined that an interfering signal was present. The calibration period to determine the threshold was necessary for this application because the power of the wireless signal would change from time to time and this implementation would account for these situations. After this was completed, jamming was successfully detected and the X310 needed to notify the eNodeB.

4.3 Alerting the eNodeB

Once the jamming was successfully detected by the X310, the modified Python script alerted the eNodeB. This was done by communicating with the OAI code, which resulted in the base station changing the frequency band of the downlink signal. Figure 4.6 shows a conceptual flow diagram of the anti-jamming process.

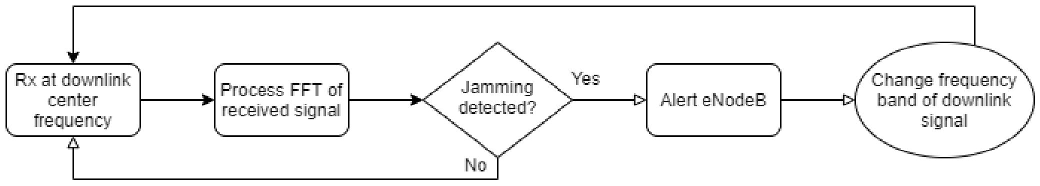

Rx at downlink center frequency
Process FFT of received signal
Jamming detected?
Yes
No
Alert eNodeB
Change frequency band of downlink signal

Figure 4.6: Conceptual Flow Diagram of Jamming Detector

The OAI server is constantly listening for requests to determine if any action is required. Through a UDP socket, the Python script connects to the server and triggers a change in the downlink frequency after sending the new transmission frequency. The frequency hopper targets frequencies from 2.66 GHz to 2.7 GHz, so the downlink frequency value that was sent was randomly selected to avoid these channels while also remaining in LTE Band 7 so that the UE can reconnect to it. Once this was accomplished, the X310 then changed the center frequency it receives at to match the new downlink frequency, resulting in a continuous and adaptive jamming detection method.

4.4 Chapter Summary

This chapter explains the LTE testbed along with the jamming mitigation performed using an SDR and GNU Radio. The LTE connection was verified by connecting the UE to the eNodeB and browsing the Internet through it. Once the jammer was introduced, the signal was monitored using a spectrogram to learn how to bypass it. The real-time detection of the X310 was successful and the eNodeB changed frequency bands when the jammer disrupted the LTE system.

5. Results & Discussion

This chapter explains the results of the jamming detection and its effect on the LTE system. The outcome of the anti-jamming is measured through analyzing the packets and determining if the eNodeB is alerted of jamming in real-time.

5.1 Testbed

Once the testbed was successfully implemented using the OAI platform, a functioning LTE system was available for testing purposes. The verification of this result was proven through the observation of an established Internet connection between the eNodeB and UE. Further investigation of the system could be done through analysis of the spectrogram and FFT plot at the downlink frequency. Figure 5.1 shows a 10 MHz wide signal being transmitted to the UE without any interference to the signal. The eNodeB is transmitting at a center frequency of 2.68 GHz because that is the default value of the LTE Band 7 configuration file. The signals are transmitted consistently and these plots help understand the regular behavior of the LTE downlink.

Once the jammer was turned on, the graphs showed different results seen in Figure 5.2, clearly indicating the presence of an interfering signal. The spectrogram shows the energy detected over time, and the N210 can be seen to transmit at high-power for 10 seconds at a frequency of 2.678 GHz then randomly hop to a 2.677 GHz. The FFT depicts how the power level is an outlier that can be detected when compared to the strength of the 10 MHz downlink signal.

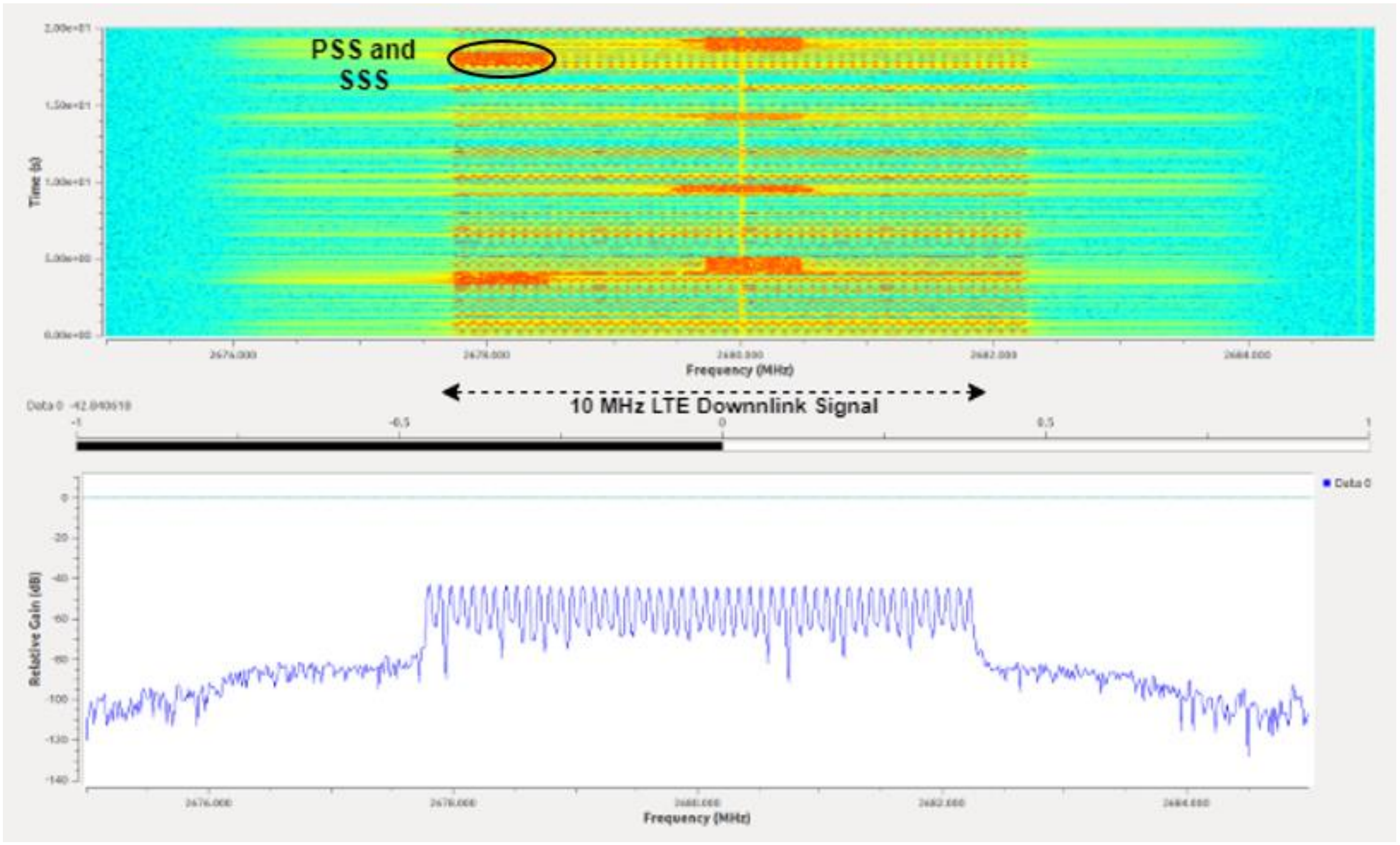


Figure 5.1: FFT Plot and Spectrogram at Center Frequency of 2.68 GHz with Jammer Off

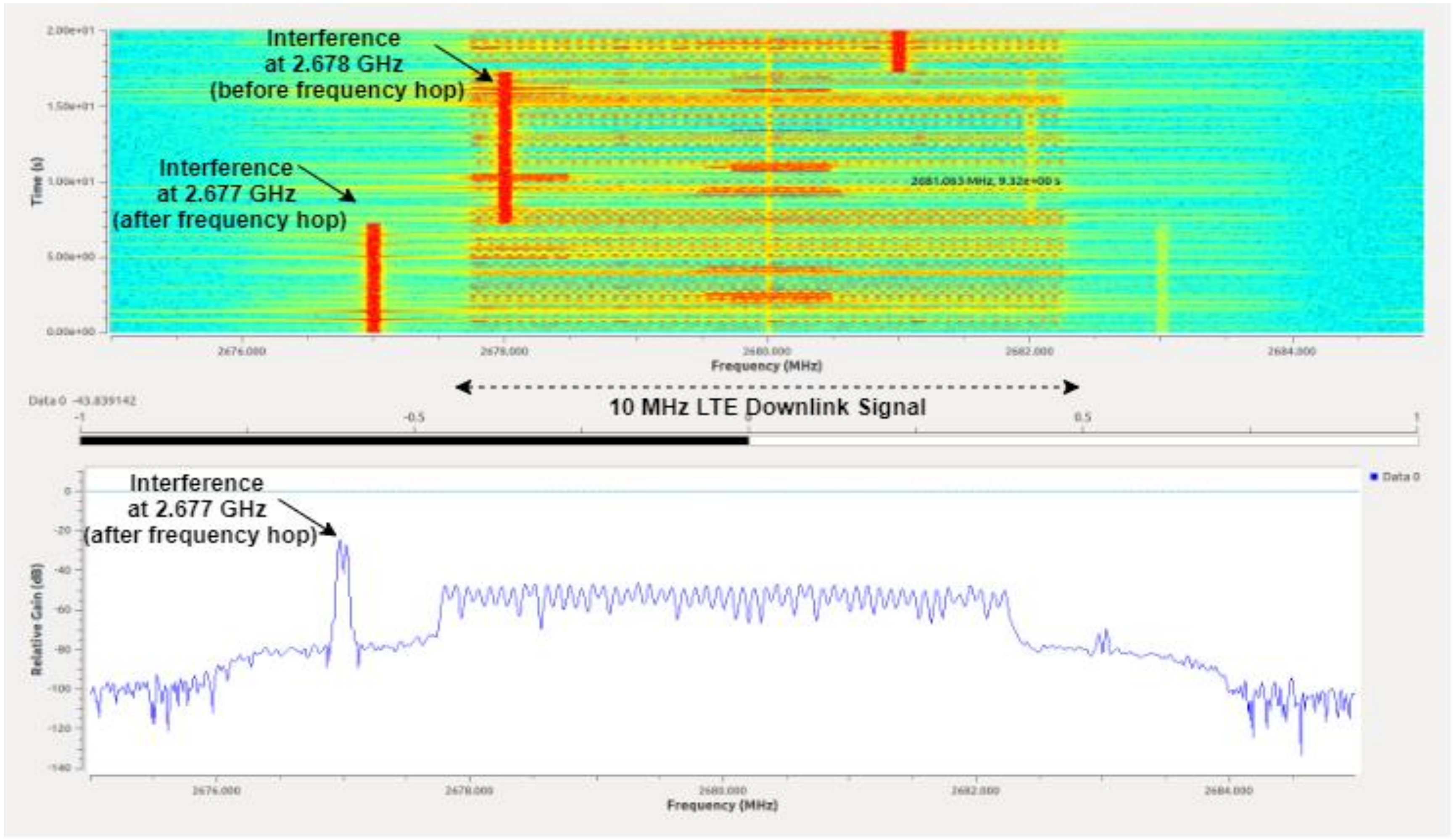
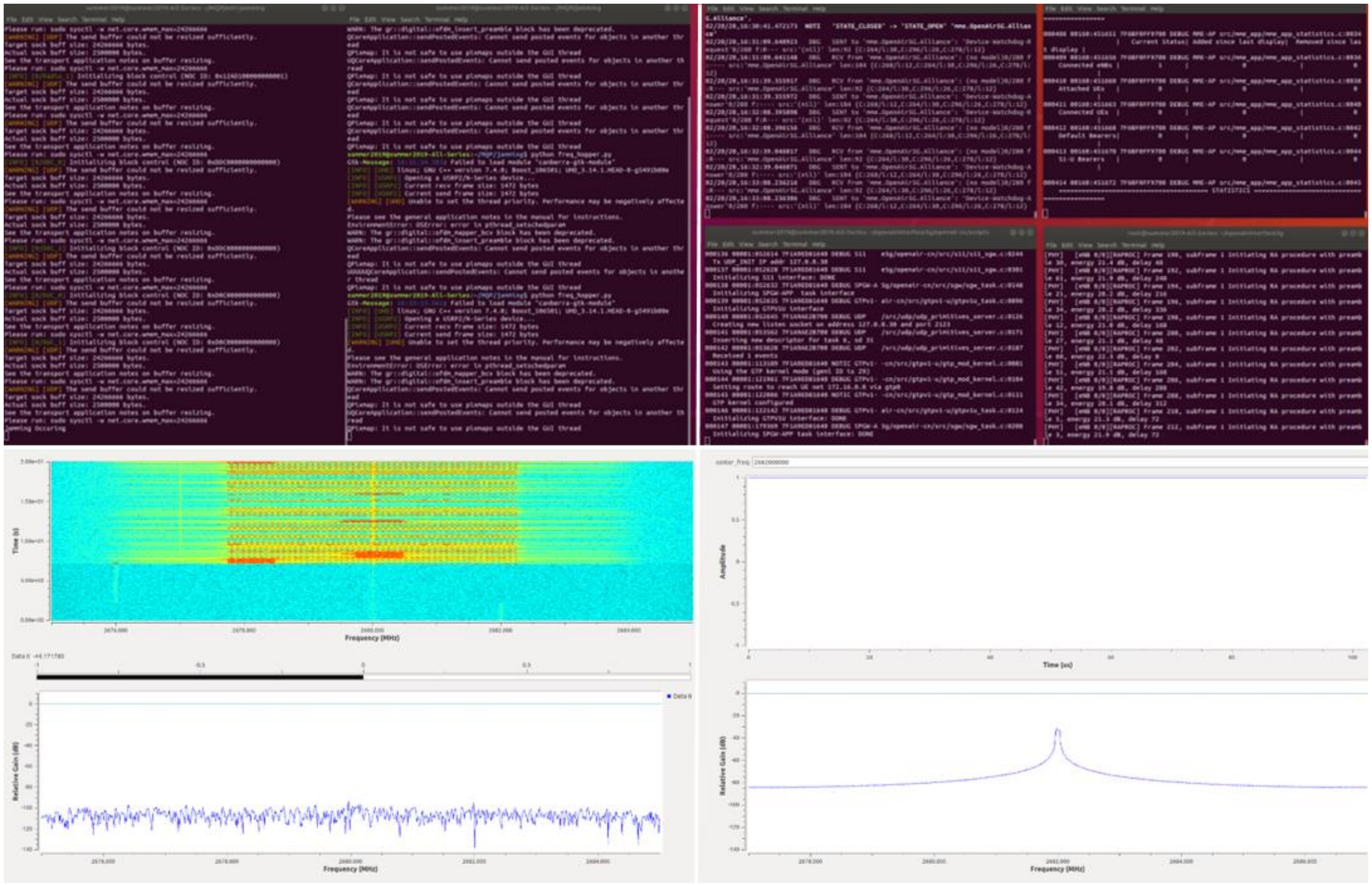


Figure 5.2: FFT Plot and Spectrogram at Center Frequency of 2.68 GHz with Jammer On

This project used these tools to monitor the behavior of the LTE downlink spectrum to determine a sufficient approach for bypassing the random frequency hopper. Figure 5.3 illustrates the team’s setup for the implementation of the LTE testbed and jamming detector. The screens are top-left: jammer and jamming detector terminals, top-right: LTE eNodeB and EPC terminals, bottom-left: jamming detector running, bottom-right: jammer running.

The jamming and jamming detector terminal were used to run the jamming and anti-jamming scripts, respectively. The terminal with the eNodeB and EPC were used to run the LTE system. To make sure the UE was connected to the Internet, the HSS, MME, and SGW were ran. Once the eNodeB was transmitting, the UE was able to identify it and establish a connection.

The jamming detector script played two roles. It would provide a visual at the center frequency through a spectrogram and an FFT plot, but it would also continuously monitor the power levels and report to the eNodeB in the event of jamming. The jamming script shows the FFT plot of the frequency hopper, verifying that the N210 was transmitting at high power and randomly changing frequencies.



```
Please run: sudo sysctl -w net.core.wmem_max=24862464
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000000001)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24862464 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24862464
Jamming Occurring
```

```
WARN: The gr::digital::ofdm_insert_preamble block has been deprecated.
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in another thr
ead
QPixmap: It is not safe to use pixmaps outside the GUI thread
QQCoreApplication::sendPostedEvents: Cannot send posted events for objects in another th
read
QPixmap: It is not safe to use pixmaps outside the GUI thread
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in another thr
ead
QPixmap: It is not safe to use pixmaps outside the GUI thread
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in another thr
ead
QPixmap: It is not safe to use pixmaps outside the GUI thread
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in another thr
ead
QPixmap: It is not safe to use pixmaps outside the GUI thread
summer2019@summer2019-All-Series:~/MQP/Jamming$ python freq_hopper.py
Gtk-Message: 16:31:34.3022: Failed to load module "canberra-gtk-module"
[INFO] [UHD] linux; GNU C++ version 7.4.0; Boost_106501; UHD_3.14.1.HEAD-0-g5491b80e
[INFO] [USRP2] Opening a USRP2/N-Series device...
[INFO] [USRP2] Current recv frame size: 1472 bytes
[INFO] [USRP2] Current send frame size: 1472 bytes
[WARNING] [UHD] Unable to set the thread priority. Performance may be negatively affecte
d.
Please see the general application notes in the manual for instructions.
EnvironmentError: OSError: error in pthread_setschedparam
WARN: The gr::digital::ofdm_mapper_bcv block has been deprecated.
WARN: The gr::digital::ofdm_insert_preamble block has been deprecated.
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in another thr
ead
QPixmap: It is not safe to use pixmaps outside the GUI thread
UUUUUQCoreApplication::sendPostedEvents: Cannot send posted events for objects in anothe
r thread
QPixmap: It is not safe to use pixmaps outside the GUI thread
summer2019@summer2019-All-Series:~/MQP/Jamming$ python freq_hopper.py
Gtk-Message: 16:31:53.3022: Failed to load module "canberra-gtk-module"
[INFO] [UHD] linux; GNU C++ version 7.4.0; Boost_106501; UHD_3.14.1.HEAD-0-g5491b80e
[INFO] [USRP2] Opening a USRP2/N-Series device...
[INFO] [USRP2] Current recv frame size: 1472 bytes
[INFO] [USRP2] Current send frame size: 1472 bytes
[WARNING] [UHD] Unable to set the thread priority. Performance may be negatively affecte
d.
Please see the general application notes in the manual for instructions.
EnvironmentError: OSError: error in pthread_setschedparam
WARN: The gr::digital::ofdm_mapper_bcv block has been deprecated.
WARN: The gr::digital::ofdm_insert_preamble block has been deprecated.
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in another thr
ead
QPixmap: It is not safe to use pixmaps outside the GUI thread
QQCoreApplication::sendPostedEvents: Cannot send posted events for objects in another th
read
QPixmap: It is not safe to use pixmaps outside the GUI thread
```

```
G.Alliance'.
02/28/20,16:30:41.472171  NOTI   'STATE_CLOSED' -> 'STATE_OPEN' 'mme.OpenAir5G.Allian
ce'
02/28/20,16:31:09.040923   DBG   SENT to 'mme.OpenAir5G.Alliance': 'Device-Watchdog-R
equest'0/280 f:R--- src:'(nil)' len:92 {C:264/l:36,C:296/l:20,C:278/l:12}
02/28/20,16:31:09.041148   DBG   RCV from 'mme.OpenAir5G.Alliance': (no model)0/280 f
:---- src:'mme.OpenAir5G.Alliance' len:104 {C:268/l:12,C:264/l:36,C:296/l:20,C:278/l:
12}
02/28/20,16:31:39.333917   DBG   RCV from 'mme.OpenAir5G.Alliance': (no model)0/280 f
:R--- src:'mme.OpenAir5G.Alliance' len:92 {C:264/l:36,C:296/l:20,C:278/l:12}
02/28/20,16:31:39.333972   DBG   SENT to 'mme.OpenAir5G.Alliance': 'Device-Watchdog-A
nswer'0/280 f:---- src:'(nil)' len:104 {C:268/l:12,C:264/l:36,C:296/l:20,C:278/l:12}
02/28/20,16:32:08.395898   DBG   SENT to 'mme.OpenAir5G.Alliance': 'Device-Watchdog-R
equest'0/280 f:R--- src:'(nil)' len:92 {C:264/l:36,C:296/l:20,C:278/l:12}
02/28/20,16:32:08.396158   DBG   RCV from 'mme.OpenAir5G.Alliance': (no model)0/280 f
:---- src:'mme.OpenAir5G.Alliance' len:104 {C:268/l:12,C:264/l:36,C:296/l:20,C:278/l:
12}
02/28/20,16:32:39.040817   DBG   RCV from 'mme.OpenAir5G.Alliance': (no model)0/280 f
:R--- src:'mme.OpenAir5G.Alliance' len:92 {C:264/l:36,C:296/l:20,C:278/l:12}
02/28/20,16:32:39.040871   DBG   SENT to 'mme.OpenAir5G.Alliance': 'Device-Watchdog-A
nswer'0/280 f:---- src:'(nil)' len:104 {C:268/l:12,C:264/l:36,C:296/l:20,C:278/l:12}
02/28/20,16:33:08.236216   DBG   RCV from 'mme.OpenAir5G.Alliance': (no model)0/280 f
:R--- src:'mme.OpenAir5G.Alliance' len:92 {C:264/l:36,C:296/l:20,C:278/l:12}
02/28/20,16:33:08.236306   DBG   SENT to 'mme.OpenAir5G.Alliance': 'Device-Watchdog-A
nswer'0/280 f:---- src:'(nil)' len:104 {C:268/l:12,C:264/l:36,C:296/l:20,C:278/l:12}
```

```
=================

000408 00108:451651 7F08F0FF9700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0034
                  |   Current Status| Added since last display|  Removed since las
t display |
000409 00108:451656 7F08F0FF9700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0036
    Connected eNBs |          1     |              0          |            0
           |
000410 00108:451660 7F08F0FF9700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0038
    Attached UEs   |          0     |              0          |            0
           |
000411 00108:451663 7F08F0FF9700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0040
    Connected UEs  |          0     |              0          |            0
           |
000412 00108:451668 7F08F0FF9700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0042
    Default Bearers|          0     |              0          |            0
           |
000413 00108:451670 7F08F0FF9700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0044
    S1-U Bearers   |          0     |              0          |            0
           |

000414 00108:451672 7F08F0FF9700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0045
    ====================================== STATISTICS ===============================
=================
```

```
000136 00001:052614 7F1A901B1640 DEBUG S11    efg/openair-cn/src/s11/s11_sgw.c:0246
  Tx UDP_INIT IP addr 127.0.0.30
000137 00001:052628 7F1A901B1640 DEBUG S11    efg/openair-cn/src/s11/s11_sgw.c:0301
  Initializing S11 interface: DONE
000138 00001:052632 7F1A901B1640 DEBUG SPGW-A 5g/openair-cn/src/sgw/sgw_task.c:0146
  Initializing SPGW-APP  task interface
000139 00001:052635 7F1A901B1640 DEBUG GTPv1- air-cn/src/gtpv1-u/gtpv1u_task.c:0896
  Initializing GTPV1U interface
000140 00001:052645 7F1A9A628700 DEBUG UDP    /src/udp/udp_primitives_server.c:0126
  Creating new listen socket on address 127.0.0.30 and port 2123
000141 00001:053562 7F1A9A628700 DEBUG UDP    /src/udp/udp_primitives_server.c:0171
  Inserting new descriptor for task 6, sd 31
000142 00001:053628 7F1A9A628700 DEBUG UDP    /src/udp/udp_primitives_server.c:0187
  Received 1 events
000143 00001:113109 7F1A901B1640 NOTIC GTPv1- -cn/src/gtpv1-u/gtp_mod_kernel.c:0081
  Using the GTP kernel mode (genl ID is 29)
000144 00001:121961 7F1A901B1640 DEBUG GTPv1- -cn/src/gtpv1-u/gtp_mod_kernel.c:0104
  Setting route to reach UE net 172.16.0.0 via gtp0
000145 00001:122806 7F1A901B1640 NOTIC GTPv1- -cn/src/gtpv1-u/gtp_mod_kernel.c:0111
  GTP kernel configured
000146 00001:122142 7F1A901B1640 DEBUG GTPv1- air-cn/src/gtpv1-u/gtpv1u_task.c:0124
  Initializing GTPV1U interface: DONE
000147 00001:179369 7F1A901B1640 DEBUG SPGW-A 5g/openair-cn/src/sgw/sgw_task.c:0200
  Initializing SPGW-APP task interface: DONE
```

```
[PHY]   [eNB 0/0][RAPROC] Frame 190, subframe 1 Initiating RA procedure with preamb
le 30, energy 21.4 dB, delay 48
[PHY]   [eNB 0/0][RAPROC] Frame 192, subframe 1 Initiating RA procedure with preamb
le 61, energy 21.9 dB, delay 248
[PHY]   [eNB 0/0][RAPROC] Frame 194, subframe 1 Initiating RA procedure with preamb
le 21, energy 20.2 dB, delay 216
[PHY]   [eNB 0/0][RAPROC] Frame 196, subframe 1 Initiating RA procedure with preamb
le 34, energy 20.2 dB, delay 336
[PHY]   [eNB 0/0][RAPROC] Frame 198, subframe 1 Initiating RA procedure with preamb
le 12, energy 21.8 dB, delay 168
[PHY]   [eNB 0/0][RAPROC] Frame 200, subframe 1 Initiating RA procedure with preamb
le 27, energy 21.1 dB, delay 48
[PHY]   [eNB 0/0][RAPROC] Frame 202, subframe 1 Initiating RA procedure with preamb
le 60, energy 22.3 dB, delay 8
[PHY]   [eNB 0/0][RAPROC] Frame 204, subframe 1 Initiating RA procedure with preamb
le 51, energy 21.1 dB, delay 168
[PHY]   [eNB 0/0][RAPROC] Frame 206, subframe 1 Initiating RA procedure with preamb
le 42, energy 19.8 dB, delay 288
[PHY]   [eNB 0/0][RAPROC] Frame 208, subframe 1 Initiating RA procedure with preamb
le 34, energy 20.1 dB, delay 312
[PHY]   [eNB 0/0][RAPROC] Frame 210, subframe 1 Initiating RA procedure with preamb
le 5, energy 21.3 dB, delay 72
[PHY]   [eNB 0/0][RAPROC] Frame 212, subframe 1 Initiating RA procedure with preamb
le 3, energy 21.9 dB, delay 72
```

Figure 5.3: Testbed Setup with Jamming Detected

5.2 Jamming Detection and Alerting the eNodeB

Probing the power levels at the center frequency at a fast rate allowed for the fast detection of jamming. While running the modified Python script shown in Appendix A with the jammer was turned on, the interfering signal was identified in real-time and the frequency channel was then changed to minimize the disruption to the LTE system. Figure 5.4 shows the X310 responding to the detection of jamming by changing the frequency channel at which it acts as a receiver. Figure 5.5 shows the output signaling jamming detection. In these figures, the X310 has not yet notified the eNodeB of jamming and it only changed the receiving frequency. The purpose of this is to demonstrate its fast response time, as it reacted before the jamming signal was plotted on the spectrogram. The FFT results shows how the received power was decreased to 0. The terminal notifies the user that jamming was detected by the X310.

Once the receiver alerted the eNodeB of the jamming, the downlink frequency was changed and the plots depicted an undisrupted LTE signal. Figure 5.6 shows the new spectrogram and FFT plot after the jamming detector notified the eNodeB to hop frequencies. Figure 5.7 shows the command terminal output signifying that the UDP socket has been sent. The spectrogram shows the key LTE signals such as the PSS and SSS being transmitted. The FFT verifies that the X310 is receiving a 10 MHz wide signal at an expected power level. The command terminal informs the user that the value of the new downlink frequency has been sent to the eNodeB and the system has evaded the jammer.

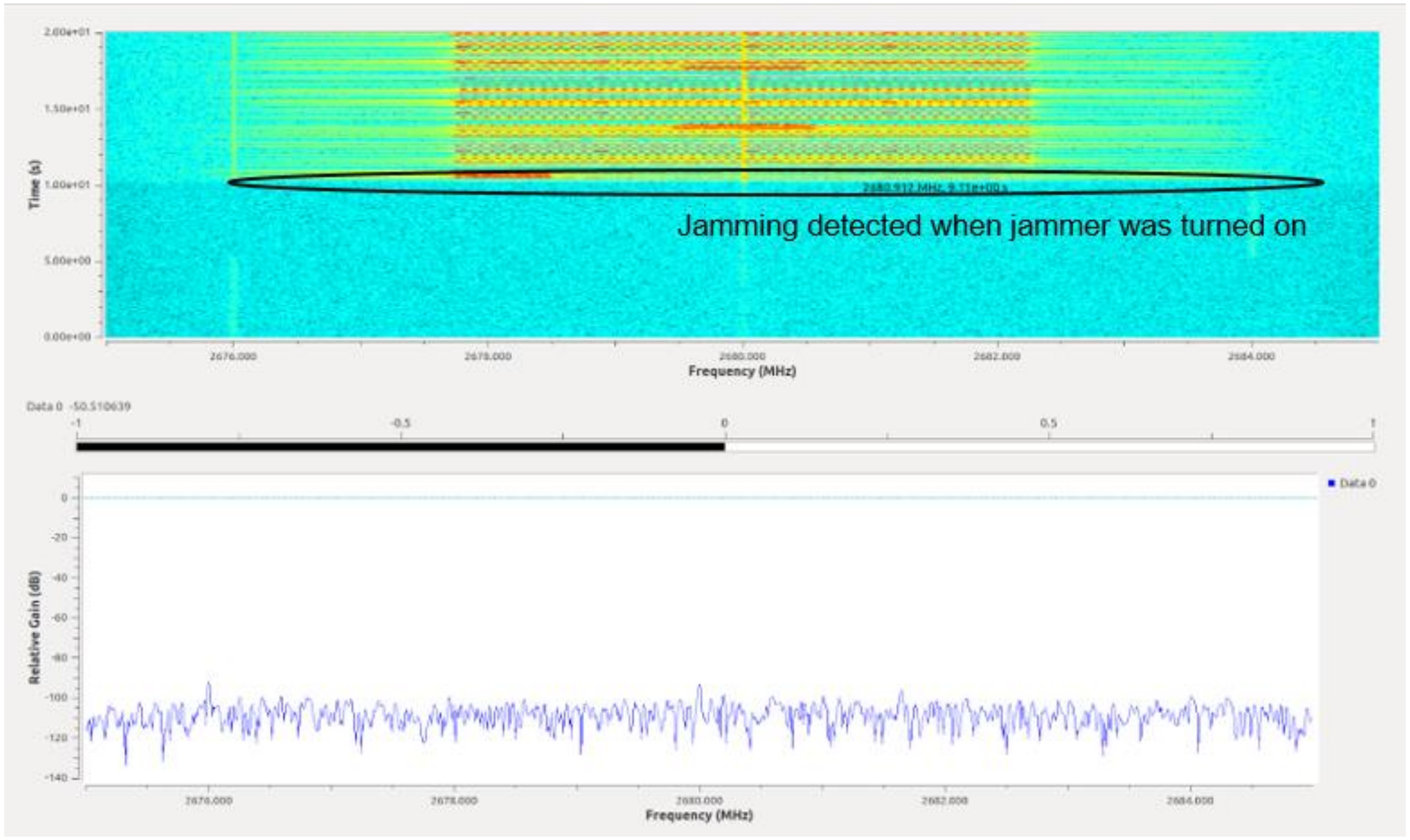
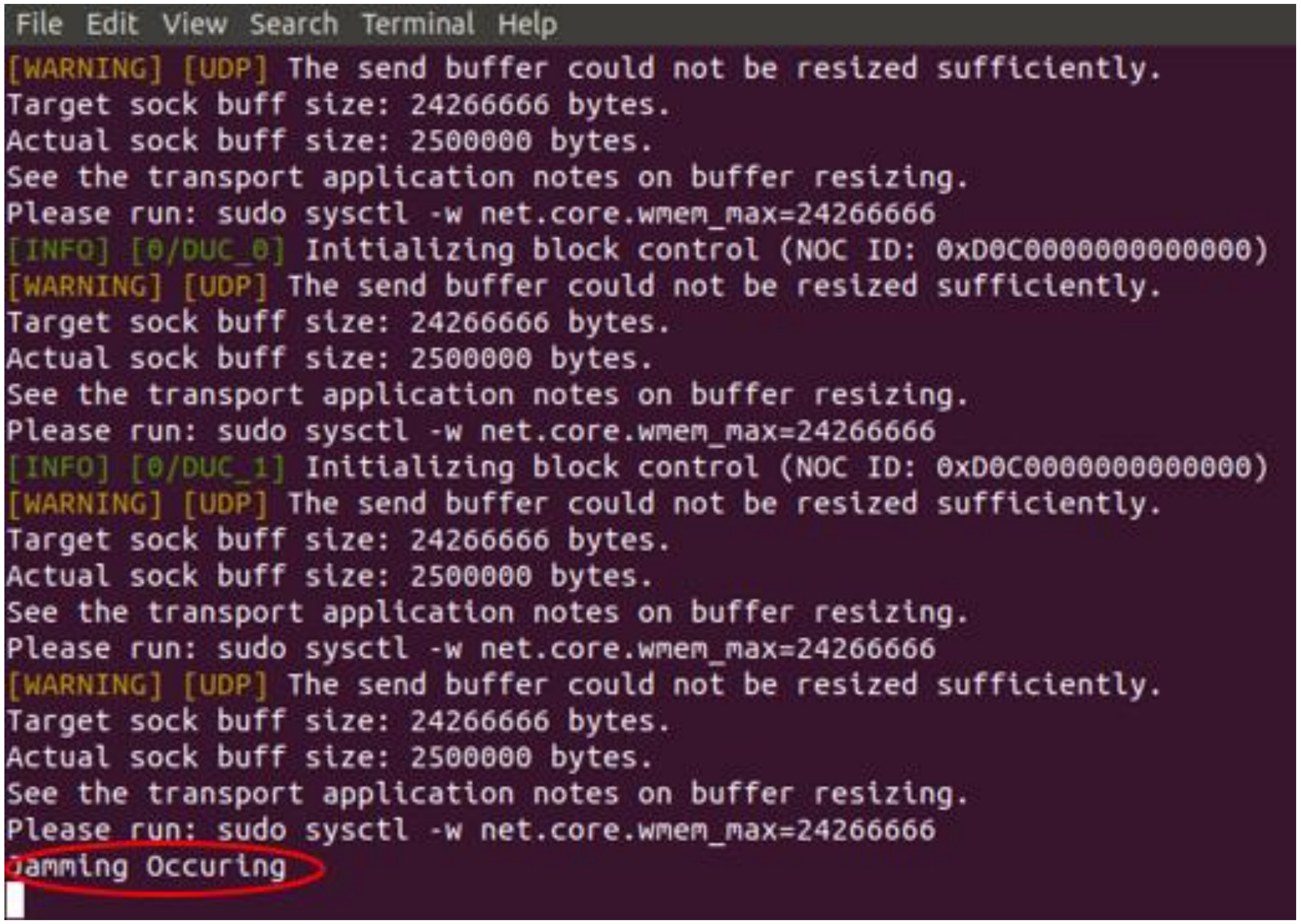


Figure 5.4: Jamming Detector Changing Receiving Frequency after Detecting Interference



```
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24266666 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24266666
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24266666 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24266666
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24266666 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24266666
[WARNING] [UDP] The send buffer could not be resized sufficiently.
Target sock buff size: 24266666 bytes.
Actual sock buff size: 2500000 bytes.
See the transport application notes on buffer resizing.
Please run: sudo sysctl -w net.core.wmem_max=24266666
Jamming Occuring
```

Figure 5.5: Jamming Detector Terminal after Detecting Interference

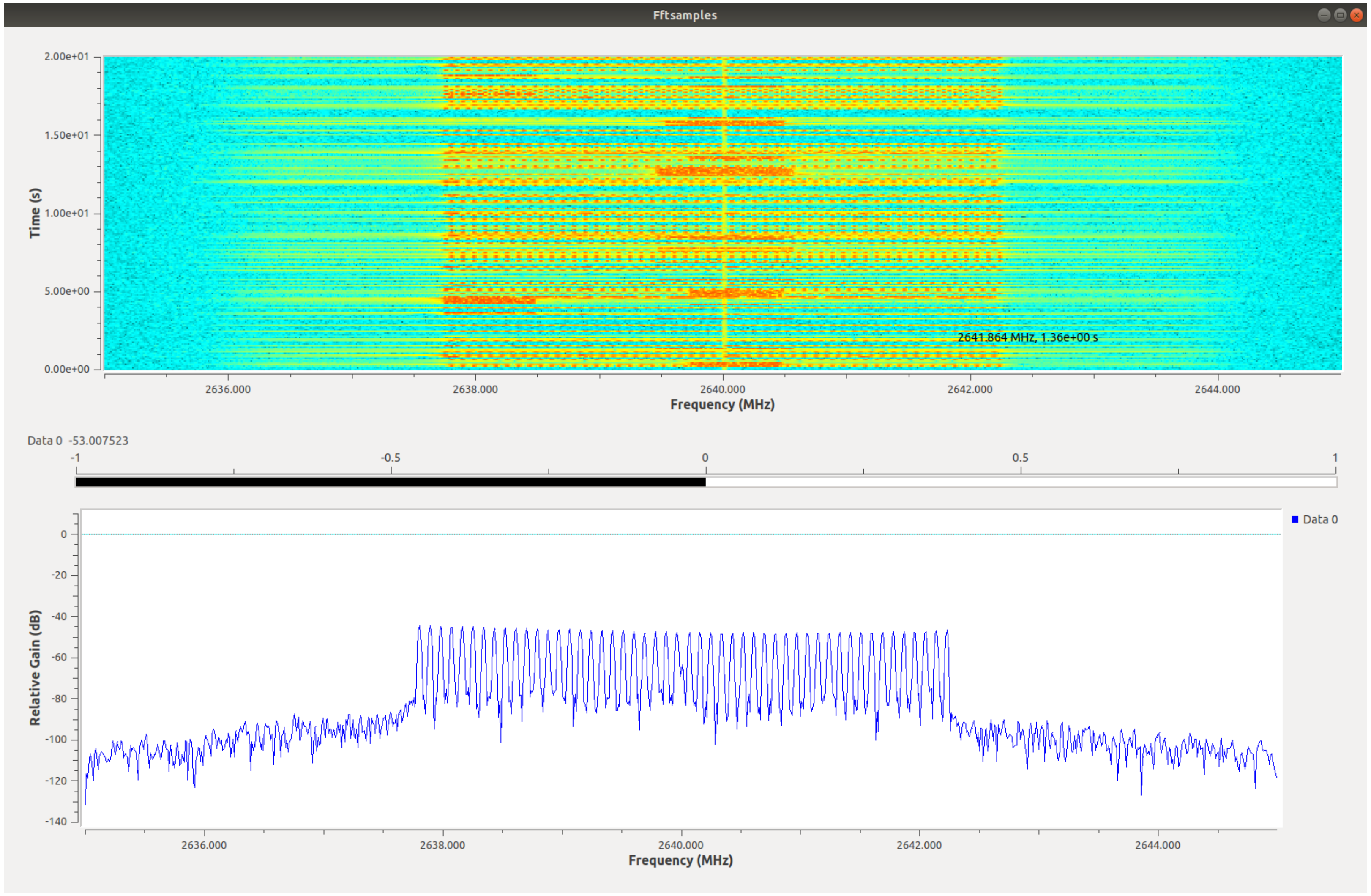
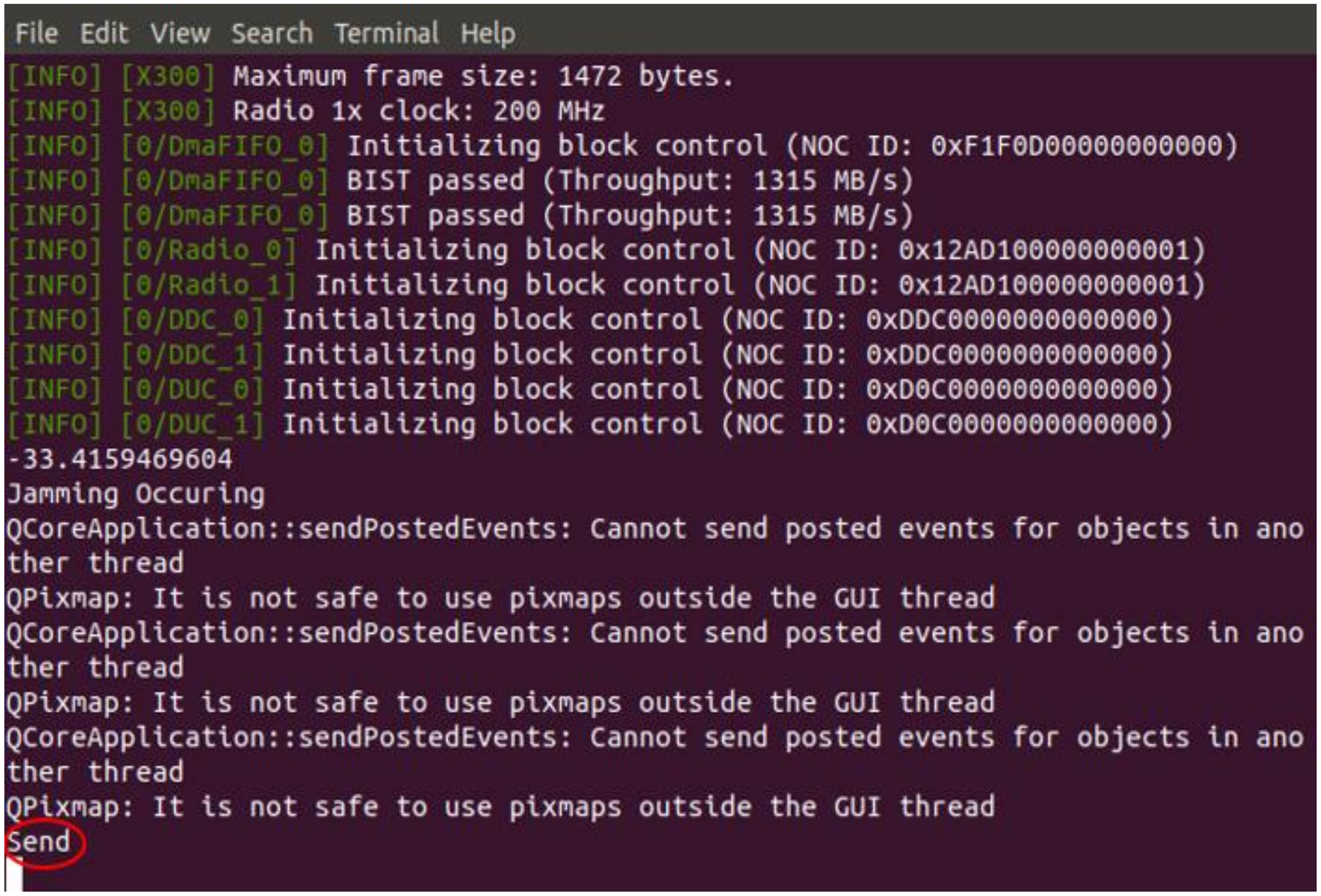


Figure 5.6: FFT Plot and Spectrogram at New Center Frequency to Bypass Jamming



```
[INFO] [X300] Maximum frame size: 1472 bytes.
[INFO] [X300] Radio 1x clock: 200 MHz
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000000)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1315 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1315 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000000001)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000000001)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
-33.4159469604
Jamming Occuring
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in ano
ther thread
QPixmap: It is not safe to use pixmaps outside the GUI thread
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in ano
ther thread
QPixmap: It is not safe to use pixmaps outside the GUI thread
QCoreApplication::sendPostedEvents: Cannot send posted events for objects in ano
ther thread
QPixmap: It is not safe to use pixmaps outside the GUI thread
Send
```

Figure 5.7: Jamming Detector Terminal after Alerting eNodeB of Interference

Once the real-time jamming detection was achieved and the base station was alerted, monitoring the wireless packets was necessary to evaluate the success of the jamming mitigation and ensure that the LTE system was not drastically disrupted. This was done through analyzing the packet receiving rate of the UE. This allowed the team to discover how many packets failed to reach the UE, resulting in slower throughput of the LTE system. Wireshark, an open-source packet analyzer, was used for this evaluation [38]. Figure 5.8 shows the Wireshark calculated packet receiving rate for an uninterrupted connection between the eNodeB and UE and Figure 5.9 shows the packet receiving rate after the eNodeB changed downlink frequencies. Figure 5.6 from [25] shows the packet loss analysis of the LTE system when the jammer was performing DoS, and the packet receiving rate decreased to 0. Once the eNodeB was alerted of the interfering signal, depicted by the red dotted line in Figure 5.8, and it changed the downlink transmission frequency, the packet receiving rate increased. The dotted green line in Figure 5.9 shows the point in time that the base station hopped frequencies. The results indicate that the UE was successful receiving packets and the DoS was evaded. As a result, the UE was still able to connect to the Internet and the LTE system was able to successfully bypass the jamming attack.

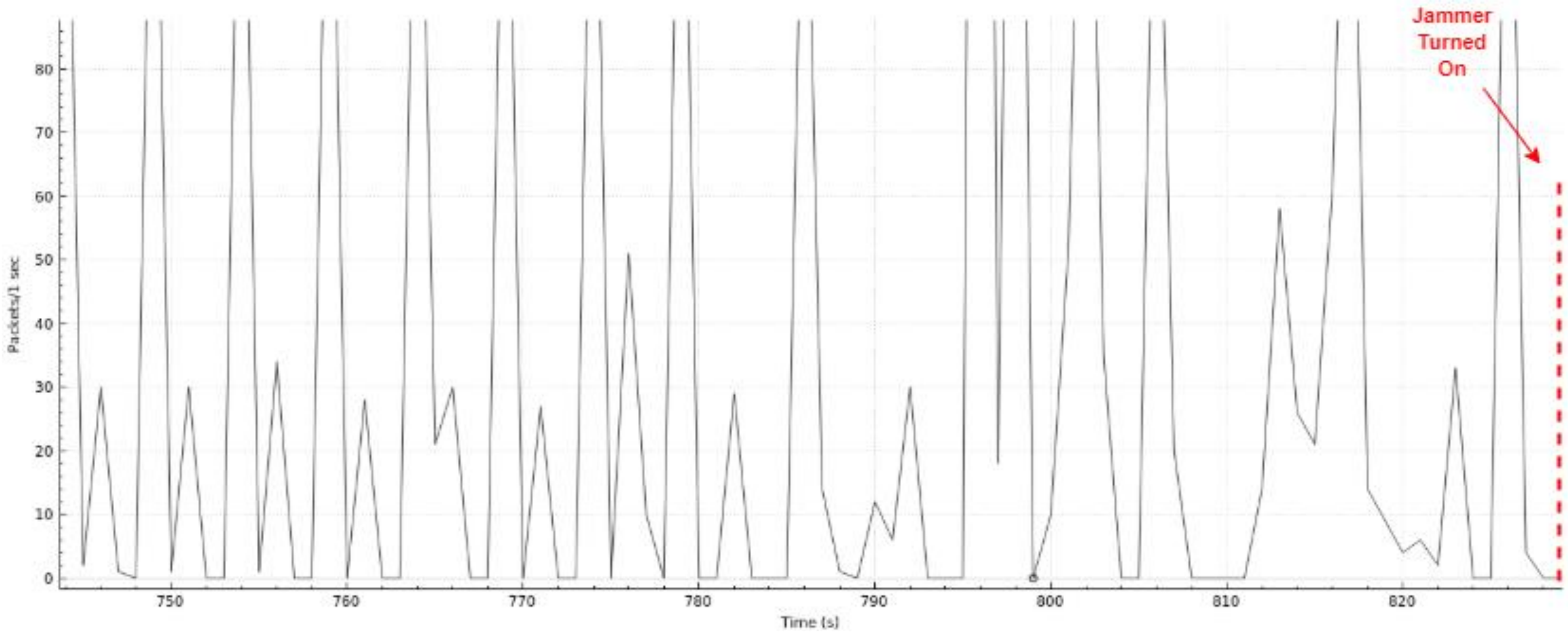


Figure 5.8: Packing Receiving Rate vs. Time at 2.68 GHz Downlink Frequency

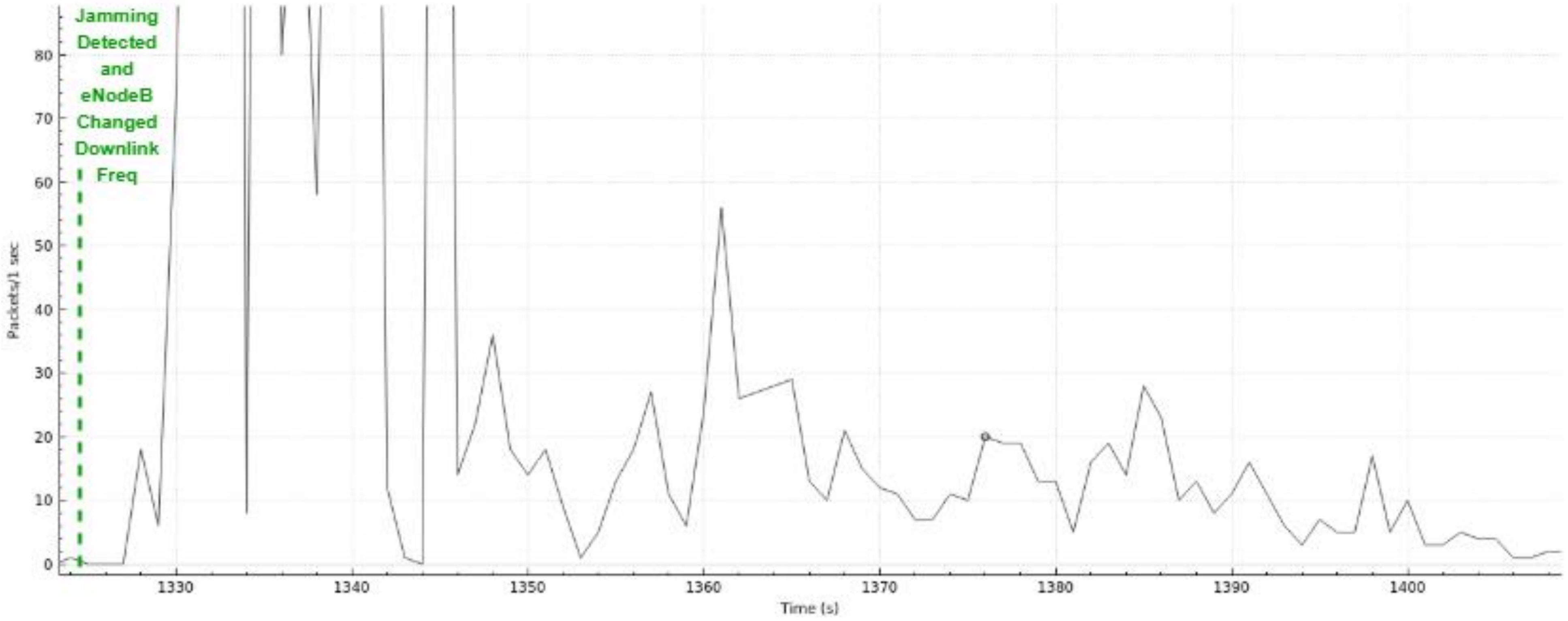


Figure 5.9: Packet Receiving Rate vs. Time After eNodeB Changed Downlink Frequency

5.3 Chapter Summary

This chapter verified the results achieved from the implementation of the LTE jamming detection. The spectrogram and FFT plots of the LTE downlink center frequency of 2.68 GHz were shown when the jammer was both off and on. The response of the X310 was seen through these visual plots. Moreover, Wireshark was used to analyze the packet loss of the LTE system and evaluate the effectiveness of the jamming detector. The base station was able to successful change the downlink frequency and bypass the denial of service attack.

6. Conclusion

This project utilized an LTE testbed and a narrow-band jammer to investigate the effectiveness of an SDR for the detection of an unwanted signal. The use of OAI software allowed for the development of an LTE system that consisted of an eNodeB and a UE with access to the Internet. Although LTE technology is vulnerable to various jamming attacks such as DoS, investigating the transmission frequencies can help develop security protocols. Using a USRP X310 was a reliable hardware component for fast and continuous jamming detection. GNU Radio is an efficient GUI tool to monitor signals and perform computations on them. By implementing a flowgraph and modifying the GNU Radio generated Python script, this project was able to achieve real-time detection of jamming using an X310. This research on LTE security is an important topic in cellular technology that can be expanded upon for future generations of cellular networks.

6.1 Recommendations for Future Work

There are many opportunities to expand on this research and make further advancements towards the security of cellular networks. Regarding the detection method, the way the jamming detection is being done is by looking at the LTE downlink waterfall curve. The maximum power level, in dB, of the downlink signal is then measured in the script. This value is then set as the threshold and if power levels rise above this plus a certain range in the frequency spectrum of the LTE downlink signal, then it is determined that a jammer is present. Finding a more adaptable way that does not slow down the system and is able to constantly update the threshold value while continuing to return real-time jamming detection would be a beneficial addition to this approach.

Another way to improve upon this project is to introduce multiple eNodeBs into the system, as it would be interesting to see how the jamming and anti-jamming protocols developed, would interact with the system. One can also introduce multiple UEs and do further experiments on the impact of using more than one UE for a single eNodeB. Also, how the UEs would react to there being multiple eNodeBs in the system is of importance as well.

This project can also be applied to a 5G cellular network. The use of OAI can be used to implement a 5G testbed similar to the one that was done for LTE. Understanding the physical layer of this new technology is essential to bypassing random frequency hopping, as the downlink signal may have different characteristics and the addition of millimeter waves may require an alternative approach.

Works Cited

[1] L. Silver, “Smartphone Ownership Is Growing Rapidly Around the World, but Not Always Equally,” Pew Research Center, 2019.

[2] R. Ghannam. A. Chung ,“User-targeted Denial-of-Service Attacks in LTE Mobile Networks,” International Conference on Wireless and Mobile Computing, Networking and Communications, 2018

[3] S. O’Dea, “Number of 4G LTE Connections Worldwide from 2012 to 2020,” Statista, 2020.

[4] H. Jensen. J. Sharpe, “LTE infrastructure for today’s military networks,” PICMG Systems and Technology, 2014.

[5] D. Talbot, “One Simple Trick Could Disable a City’s 4G Phone Network,” 2012.

[6] I. Miguelez. P. Sutton, “srsLTE: an open-source platform for LTE evolution and experimentation,” ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization, 2016

[7] J. Demel. S. Koslowski. F. Jondral, “A LTE Receiver Framework Implementation in GNU Radio,” Karlsruhe Institute of Technology, 2013

[8] R. Knopp. L. Gathier, “Demo: OpenAirInterface: an open LTE network in a PC,” EURECOM, 2014.

[9] M. Litchtman and others, “LTE/LTE-A Jamming, Spoofing, and Sniffing: Threat Assessment and Mitigation,” IEEE Communications Magazine, 2016.

[10] R. Jover. J. Lackey. A. Raghavan, “Enhancing the security of LTE networks against jamming attacks, EURASIP Journal on Information Security,” Eurasip Journal on Information Security, 2014.

[11] S. Barros, “LTE Jamming Mitigation Based on Frequency Hopping Strategies,” IEEE Latin-American Conference on Communications, 2016.

[12] V. Sruk. K.Fertalj. V. Zlomislić, “Denial of Service Attacks, Defenses and Research Challenges,” Sage Journals, 2017.

[13] G. Pantziou, “A survey on jamming attacks and countermeasures in WSNs,” IEEE, 2009

[14] K. Grover, “Jamming and Anti-Jamming techniques in Wireless Networks,” Montana State University, 2014.

[15] Li. Song. Liang, “Wireless Communications under Hostile Jamming: Security and Efficiency,” Springer, 2018.

[16] T. Mshvidobadze, “Evolution Mobile Wireless Communication and LTE Networks,” 6th International Conference on Application of Information and Communication Technologies, 2012.

[17] Rajiv, “Evolution of wireless technologies 1G to 5G in mobile communication,” RF Page, 2018.

[18] C. Cox, “An introduction to LTE: LTE, LTE-advanced, SAE,” VoLTE and 4G MobileCommunications. John Wiley &amp; Sons Ltd., 2014.

[19] Netmanias, “LTE Network Architecture: Basic,” Netmanias, 2013.

[20] OpenAirInterface Software Alliance, “About us,” 2020

[21] Alcatel-Lucent, “The LTE Network Architecture,” Alcatel-Lucent, 2009

[22] N. Gompa, “What is LTE?,”,Extreme Tech, 2015.

[23] Y. Chaudhari, “Need of Physical Layer Security in LTE: Analysis of Vulnerabilities in LTE Physical Layer,” 2017 International Conference on Wireless Communications, Signal Processing and Networking, 2017.

[24] Keysight Technologies, “LTE Physical Layer Overview,” Keysight Technologies, 2019.

[25] Y. Brown C. Teng, “LTE Frequency Hopping Jammer,” Worcester Polytechnic Institute, 2019.

[26] Techplayon, “Explain LTE Frame Structure both for FDD and TDD,” Techplayon, 2018.

[27] M. Lichtman, R. P. Jover, M. Labib, R. Rao, V. Marojevic, and J. H. Reed, “LTE/LTE-a jamming, spoofing, and sniffing: Threat assessment and mitigation,” IEEE Commun., 2016.

[28] OpenAirInterface5G, “OpenAirInterface Implementation Code Repository,” GitLab, 2019

[29] GeeksforGeeks, “Error Detection in Computer Networks,” GeeksforGeeks, 2019.

[30] N. Nikaein, “OpenAirInterface: A Flexible Platform for 5G Research,” ACM SIGCOMM Computer Communication Review, 2014.

[31] A. Tato, “Software Defined Radio: A Brief Introduction,” Proceedings, 2018.
J. Fernández, “Software Defined Radio: Basic Principles and Applications,” Faculty of
[32] Engineering Journal, 2014.
Wireless Innovation, “What is Software Defined Radio?,” Wireless Innovation Forum, 2015.

[33] Ettus Research, National Instruments, 2020.

[34] Creative Commons Attribution-ShareAlike, “What is GNU Radio?,” 2019.

[35] GNU Radio, “Guided Tutorial GRC,” GNU Radio, 2020

[36] OpenCells, All in one OpenAirInterface, 2019.

[37] C. Tarver, “OAI Tutorial,” Academic Theme for Hugo, 2019.

[38] Wireshark, 2020.

Appendix A: Python Code for Jamming Detector

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Fftsamples
# Generated: Mon Feb 10 14:32:35 2020
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from PyQt4 import Qt
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import fft
from gnuradio import gr
from gnuradio import qtgui
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.fft import window
from gnuradio.filter import firdes
from optparse import OptionParser
import sip
import sys
import threading
import time
from gnuradio import qtgui
import socket
import random


class fftSamples(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Fftsamples")
        Qt.QWidget.__init__(self)
    self.setWindowTitle("Fftsamples")
    qtgui.util.check_set_qss()
    try:
        self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
    except:
        pass
    self.top_scroll_layout = Qt.QVBoxLayout()
    self.setLayout(self.top_scroll_layout)
    self.top_scroll = Qt.QScrollArea()
    self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
    self.top_scroll_layout.addWidget(self.top_scroll)
    self.top_scroll.setWidgetResizable(True)
    self.top_widget = Qt.QWidget()
    self.top_scroll.setWidget(self.top_widget)
    self.top_layout = Qt.QVBoxLayout(self.top_widget)
    self.top_grid_layout = Qt.QGridLayout()
    self.top_layout.addLayout(self.top_grid_layout)

    self.settings = Qt.QSettings("GNU Radio", "fftSamples")
    self.restoreGeometry(self.settings.value("geometry").toByteArray())


    ##################################################
    # Variables
    ##################################################
    self.variable_function_probe_0 = variable_function_probe_0 = 0
    self.samp_rate = samp_rate = 10e6

try:
    self.center_freq = center_freq = self.get_center_freq()

except:
        self.center_freq = center_freq = 2.68e9

    ##################################################
    # Blocks
    ##################################################
    self.probe = blocks.probe_signal_f()

    def _variable_function_probe_0_probe():
        jamming = False
    has_jammed = False
    values = []
    count = 0
        while True:
            val = self.probe.level()
        if val == 0:    # val is initialized to 0
           continue
        else:
           try:
            if val > threshold+2:    # threshold for power level
               jamming = True
               if (jamming and not has_jammed):
               print ('Jamming Occuring')
               TX_FREQS = [2625, 2630, 2635, 2640, 2645, 2650]
               TX_FREQ = random.choice(TX_FREQS)
               #send signal to base station to change center frequency
               self.uhd_usrp_source_0.set_center_freq(TX_FREQ*1e6, 0)    # change the
center frequency of receiver
               self.qtgui_waterfall_sink_x_0.set_frequency_range(TX_FREQ*1e6,
self.samp_rate)
               self.qtgui_freq_sink_x_0.set_frequency_range(TX_FREQ*1e6,
self.samp_rate)

               #send freq to base station
               oai_ip  ='127.0.0.125'
               port =12000
               oai_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
               oai_sock.connect((oai_ip,port))
               BUFFER = 'd'+ str(int(TX_FREQ))
               oai_sock.send(BUFFER.encode())
               print('Send')

               has_jammed = True
               else:
                  continue

           except:
           values.append(val)
           count = count + 1
           if count == 5000:
               threshold = max(values)
               print threshold

               try:
                  self.set_variable_function_probe_0(val)
               except AttributeError:
                  pass
               time.sleep(1.0 / (1000))
        _variable_function_probe_0_thread =
threading.Thread(target=_variable_function_probe_0_probe)
        _variable_function_probe_0_thread.daemon = True
```

```
print(_variable_function_probe_0_thread.start())

self.uhd_usrp_source_0 = uhd.usrp_source(
    ",".join(("addr=192.168.10.2", "")),
    uhd.stream_args(
        cpu_format="fc32",
        channels=range(1),
    ),
)
self.uhd_usrp_source_0.set_samp_rate(samp_rate)
self.uhd_usrp_source_0.set_center_freq(center_freq, 0)
self.uhd_usrp_source_0.set_gain(28, 0)
self.qtgui_waterfall_sink_x_0 = qtgui.waterfall_sink_c(
    1024, #size
    firdes.WIN_BLACKMAN_hARRIS, #wintype
    center_freq, #fc
    samp_rate, #bw
    "", #name
        1 #number of inputs
)
self.qtgui_waterfall_sink_x_0.set_update_time(0.10)
self.qtgui_waterfall_sink_x_0.enable_grid(False)
self.qtgui_waterfall_sink_x_0.enable_axis_labels(True)

if not True:
  self.qtgui_waterfall_sink_x_0.disable_legend()

if "complex" == "float" or "complex" == "msg_float":
  self.qtgui_waterfall_sink_x_0.set_plot_pos_half(not True)

labels = ['', '', '', '', '',
          '', '', '', '', '']
colors = [0, 0, 0, 0, 0,
          0, 0, 0, 0, 0]
alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
          1.0, 1.0, 1.0, 1.0, 1.0]
for i in xrange(1):
    if len(labels[i]) == 0:
        self.qtgui_waterfall_sink_x_0.set_line_label(i, "Data {0}".format(i))
    else:
        self.qtgui_waterfall_sink_x_0.set_line_label(i, labels[i])
    self.qtgui_waterfall_sink_x_0.set_color_map(i, colors[i])
    self.qtgui_waterfall_sink_x_0.set_line_alpha(i, alphas[i])

self.qtgui_waterfall_sink_x_0.set_intensity_range(-140, 10)
```

```
        self._qtgui_waterfall_sink_x_0_win =
sip.wrapinstance(self.qtgui_waterfall_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_waterfall_sink_x_0_win)
        self.qtgui_number_sink_0 = qtgui.number_sink(
            gr.sizeof_float,
            0,
            qtgui.NUM_GRAPH_HORIZ,
            1
        )
        self.qtgui_number_sink_0.set_update_time(0.10)
        self.qtgui_number_sink_0.set_title("")

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        units = ['', '', '', '', '',
                 '', '', '', '', '']
        colors = [("black", "black"), ("black", "black"), ("black", "black"), ("black", "black"),
("black", "black"),
                  ("black", "black"), ("black", "black"), ("black", "black"), ("black", "black"),
("black", "black")]
        factor = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        for i in xrange(1):
            self.qtgui_number_sink_0.set_min(i, -1)
            self.qtgui_number_sink_0.set_max(i, 1)
            self.qtgui_number_sink_0.set_color(i, colors[i][0], colors[i][1])
            if len(labels[i]) == 0:
                self.qtgui_number_sink_0.set_label(i, "Data {0}".format(i))
            else:
                self.qtgui_number_sink_0.set_label(i, labels[i])
            self.qtgui_number_sink_0.set_unit(i, units[i])
            self.qtgui_number_sink_0.set_factor(i, factor[i])

        self.qtgui_number_sink_0.enable_autoscale(False)
        self._qtgui_number_sink_0_win =
sip.wrapinstance(self.qtgui_number_sink_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_number_sink_0_win)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
            1024, #size
            firdes.WIN_BLACKMAN_hARRIS, #wintype
            center_freq, #fc
            samp_rate, #bw
            "", #name
            1 #number of inputs
        )
        self.qtgui_freq_sink_x_0.set_update_time(0.10)
        self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
        self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
        self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
        self.qtgui_freq_sink_x_0.enable_autoscale(False)
        self.qtgui_freq_sink_x_0.enable_grid(False)
        self.qtgui_freq_sink_x_0.set_fft_average(1.0)
        self.qtgui_freq_sink_x_0.enable_axis_labels(True)
        self.qtgui_freq_sink_x_0.enable_control_panel(False)

        if not True:
          self.qtgui_freq_sink_x_0.disable_legend()

        if "complex" == "float" or "complex" == "msg_float":
          self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                  '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                  "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                  1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_freq_sink_x_0_win =
sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_grid_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.fft_vxx_0 = fft.fft_vcc(1024, True, (window.blackmanharris(1024)), True, 1)
        self.blocks_vector_to_stream_0 = blocks.vector_to_stream(gr.sizeof_float*1, 1024)
        self.blocks_stream_to_vector_0 =
blocks.stream_to_vector(gr.sizeof_gr_complex*1, 1024)
        self.blocks_nlog10_ff_0 = blocks.nlog10_ff(10, 1024, 0)
        self.blocks_complex_to_mag_squared_0 =
blocks.complex_to_mag_squared(1024)


        ##################################################
        # Connections
        ##################################################
        self.connect((self.blocks_complex_to_mag_squared_0, 0),
(self.blocks_nlog10_ff_0, 0))
        self.connect((self.blocks_nlog10_ff_0, 0), (self.blocks_vector_to_stream_0, 0))
        self.connect((self.blocks_stream_to_vector_0, 0), (self.fft_vxx_0, 0))
        self.connect((self.blocks_vector_to_stream_0, 0), (self.probe, 0))
        self.connect((self.blocks_vector_to_stream_0, 0), (self.qtgui_number_sink_0, 0))
        self.connect((self.fft_vxx_0, 0), (self.blocks_complex_to_mag_squared_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.blocks_stream_to_vector_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.qtgui_waterfall_sink_x_0, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "fftSamples")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_variable_function_probe_0(self):
        return self.variable_function_probe_0

    def set_variable_function_probe_0(self, variable_function_probe_0):
        self.variable_function_probe_0 = variable_function_probe_0

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.qtgui_waterfall_sink_x_0.set_frequency_range(self.center_freq,
self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(self.center_freq, self.samp_rate)

    def get_center_freq(self):
        return self.center_freq

    def set_center_freq(self, center_freq):
        self.center_freq = center_freq
        self.uhd_usrp_source_0.set_center_freq(self.center_freq, 0)
        self.qtgui_waterfall_sink_x_0.set_frequency_range(self.center_freq,
self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(self.center_freq, self.samp_rate)


def main(top_block_cls=fftSamples, options=None):
    from distutils.version import StrictVersion
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()
    tb.start()
    tb.show()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


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