2019-12-13

LTE Frequency Hopping Jammer

Cynthia Teng
Worcester Polytechnic Institute

YaYa Mao Brown
Worcester Polytechnic Institute

LTE Frequency Hopping Jammer

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 on: December 13, 2019

Submitted by:

YaYa Brown,

Electrical and Computer Engineering

Cynthia Teng,

Electrical and Computer Engineering

Project Advisor:

Doctor Alexander Wyglinski,

Electrical and Computer Engineering

Worcester Polytechnic Institute

Abstract

The goal of this project was to show that communication with a cellular base station and user equipment could be interfered with using narrowband jamming. Specifically, a randomized frequency hopping jammer was used as the main method to disrupt service. The testbed was built with OpenAirInterface, software-defined radios, and a Samsung s4 phone. It was found to be possible to greatly disrupt communications in an LTE system with a jammer.

Acknowledgments

The success of this project was only made possible by the support and contributions of other members of the WPI community. We would like to thank those people for their help these past few months. We would like to thank our faculty advisor, Professor Alexander Wyglinski, for his guidance and help throughout the course of this project. We would also like to thank Kuldeep Gill for his advice on the LTE system, OpenAirInterface, and ways to verify our results. We would also like to thank Julien Ataya and Matthew Farah for their help during A term in the development of the testbed.

Executive Summary

Mobile devices have become integrated into all aspects of our life due to their convenience and versatility. Effective jamming can deny many people from gaining wireless network access, which can drastically affect their productivity and in some instances their safety. Most studies that are related to jamming attacks in the physical layer evaluate the threats and effectiveness of jamming by analyzing the system models. There is a need for more practical experiments on jamming attacks that can simulate malicious jamming towards different signals in the physical layer. The main contributions of this project are to implement testbed using software-defined radios, build a Single Tone Jammer (STJ), and learn potential vulnerabilities in LTE Network for jamming.

Different jamming approaches were evaluated in order to find an effective and feasible jamming approach for the testbed, and the decision was made to do frequency hopping as the primary form of jamming. Frequency hopping used narrowband jamming signals and it was easier to accomplish since there was no need for synchronization. The testbed was built with Ubuntu 18.04.3 LTS using a USRP B210 for the eNodeB, a Samsung s4 for the UE, and a USRP N210 for the jammer. After downloading OpenAirInterface, the eNodeB, SIM card, and EPC were programmed using two online tutorials; one of the tutorials was provided by Open Cells Project and the other by a Ph.D. student Chance Tarver on his own website. GNU Radio was used to implement the frequency hopping. In order to test whether the jammer was having an effect on the LTE system, a program called Wireshark was used as a way to validate the results.

The wideband jammer was tested first in order to verify jamming capabilities. The overall result was a Denial of Service (DoS) once the jammer was turned on. This was seen in Wireshark by the retransmission messages and eventual loss of packets. The UE detached itself from the network due to the quality of the channel being of such poor quality. Figure ES. 1 shows the number of packets per second across time. The graph has spikes of a large number of packets and then a time when no packets are being exchanged. This was due to the buffer that YouTube videos use. When a video was being streamed there

is not a constant stream of packets needing to be received continuously because this would cause bad video quality if just one packet was lost or needed to be retransmitted. YouTube uses a buffer to combat this problem by getting a certain number of packets to fill the buffer and then when the video plays packet transmission is stopped until the buffer has depleted to a certain level. This is shown in Figure ES.1 by the spikes, when the buffer is being filled, and then the time of no packets because the buffer was filled.

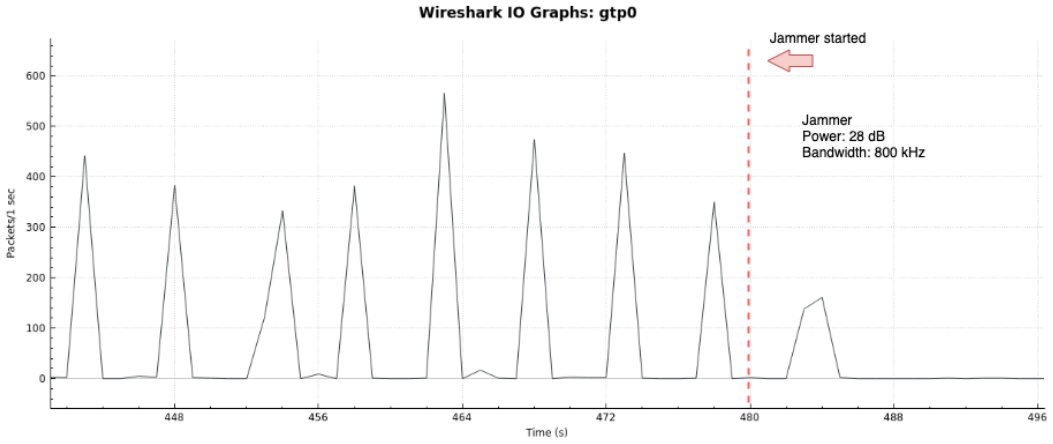


Figure ES. 1 Wideband jammer packet analysis results. After the jammer is on, the number of packets successfully transmitted drops to zero and the UE detached from the network.

The frequency hopping jammer had two different effects on the LTE system; packet corruption and Denial of Service (DoS). The first effect was packet corruption that caused the loss of a few packets and showed a decrease in the number of packets being successfully transmitted. The time between packets increased once the jammer starts transmitting because a few packets are being dropped and retransmissions occurred which caused more time between packets. Packet corruption occurred but not DoS. Throughout the time of testing, even though packets were being dropped, the video quality remained the same and the user was unable to notice a difference in service. This is due to the frequency hopping being randomized.

DoS was achieved by running the experiment multiple times until the phone disconnected in one case. This is shown in Figure ES. 3. The dashed red line indicates when the jammer was started and then there is one spike of delay. Then the number of packets per second dropped significantly until it went to zero and the UE disconnected from the network.

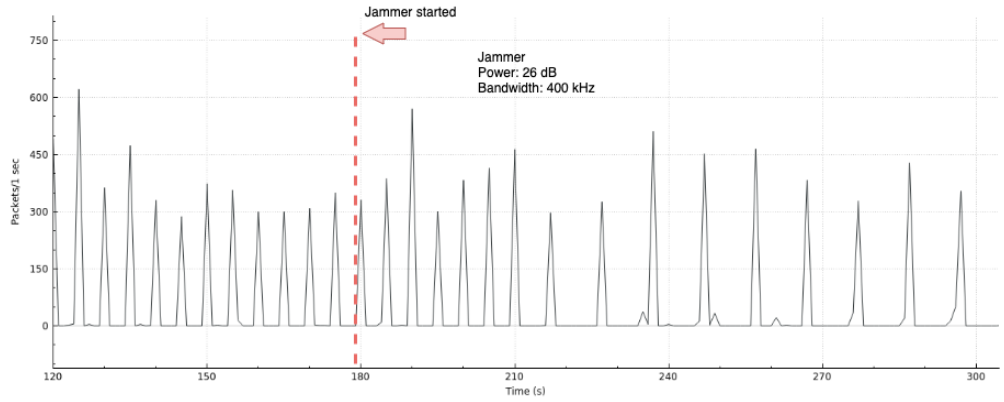


Figure ES. 2 The packet analysis result from frequency hopping jamming at 26dB, a bandwidth of 400 kHz, and at a clock rate of 50ms and a polling rate of 25ms.

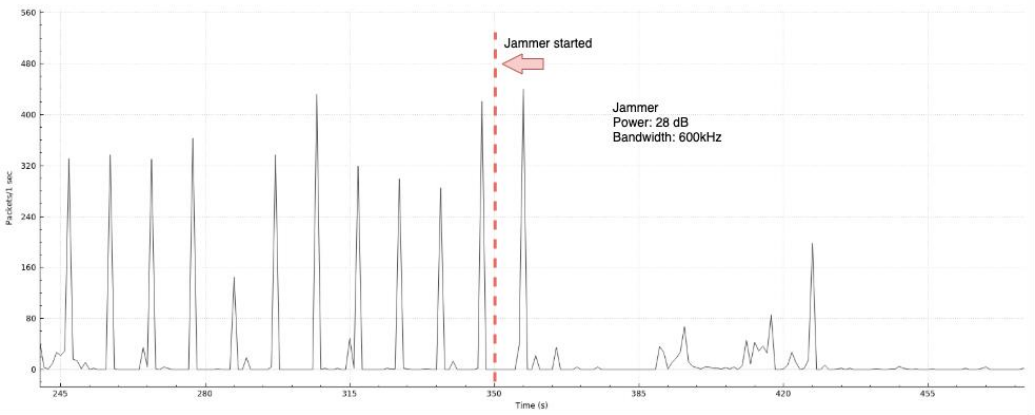


Figure ES. 3 Graph showing successful DoS caused by jammer at 28dB, a bandwidth of 600 kHz, and the clock at 50ms and a polling rate of 25ms.

Overall, the results showed that LTE communication is vulnerable to partial band jamming, which could directly create the disconnection of user equipment (UE). And frequency hopping jammer could affect LTE and result in packet retransmission, packet disruption, and even DoS when the power of the jamming signals was no less than 28 dB.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Abbreviation	Term
1G	First-Generation Systems
3PP	3rd Generation Partnership Project
ACK	Acknowledgement
AMTJ	Asynchronous Multi Tone Jamming
ASTJ	Asynchronous Single Tone Jamming
BW	Bandwidth
C-RS	Cell-specific Reference Signal
CFI	Control Format Indicator
D-RS	Demodulation Reference Signal
DCI	Downlink Control Information
DDoS	Distributed Denial of Service
DFT	Discrete Fourier Transform
DL	Downlink
DoS	Denial of Service
eNB/ eNodeB	Evolved node B
EPC	Evolved Packet Core
FDD	Frequency Division Duplex
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HARQ	Hybrid Automatic Repeat Request
HHS	Home Subscriber Server
HSS	Home Subscriber Server
IFFT	Inverse Fast Fourier Transform
IP	Internet Protocol
LTE	Long Term Evolution
LTE-A	Long Term Evolution Advanced
MAC	Medium Access Control
MBMS	Multimedia Broadcast Multicast Service
MBSFN-RS	Multimedia Broadcast multicast service Single Frequency Network Service Ref Sig
MCH	Multicast Channel
MCH	Multicast Channel
MIB	Message Information Block
MIMO	Multiple-Input Multiple-Output
MME	Mobility Management Entity
MTJ	Multi Tone Jamming
NACK	Negative Acknowledgment
OAI	OpenAirInterface
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	Orthogonal Frequency-Division Multiple Access
P-RS	Positioning Reference Signal
PBCH	Physical Broadcast Channel
PBJ	Partial Band Jamming
PCFICH	Physical Control Format Indicator Channel
PDCCH	Physical Downlink Control Channel
PDCP	Packet Data Convergence Protocol

PDN	Packet Data Network
PDN-GW	Packet Data Network Gateway
PDSCH	Physical Downlink Shared Channel
PGW	Packet Data Network Gateway
PHCH	Physical Multicast Channel
PHICH	Physical Hybrid Arq Indicator Channel
PHY	Physical Layer
PRACH	Physical Random Access Channel
PSS	Primary Synchronization Signal
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
RACH	Random Access Channel
RB	Resource Block
RLC	Radio Link Control
RRC	Radio Resource Control
RRM	Radio Research Management
S-RS	Sounding Reference Signal
SC-FDMA	Single Carrier Frequency Multiple Access
SDR	Software-Defined Radio
SFN	System Frame Number
SGW	Serving Gateway
SIM	Subscriber Identity Module
SMS	Short Message Service
SSS	Secondary Synchronization Signal
STJ	Single Tone Jamming
TDD	Time Division Duplex
UCI	Uplink Control Information
UE	User Equipment
UE-RS	UE-Specific Reference Signal
UL	Uplink
UL-SCH	Uplink-Shared Channel
UMTS	Universal Mobile Telecommunication System
USRP	Universal Software Radio Peripheral

1. Introduction

1.1 Motivation

Mobile devices have become integrated into all aspects of our life due to their convenience and versatility. The applications on them have contributed to socio-economic development in areas of health, education, and finances with everything being accomplished by a few taps on a screen [1]. The number of cellphone users has increased partly due to the introduction of smartphones with their user-friendly interface [2]. The increase in people's usage of data and dependence on it creates a need for further advancement in technology [2]. Long Term Evolution (LTE) is a protocol for wireless broadband communications and was created by the 3rd Generation Partnership Project (3PP) in order to meet the data usage and total traffic demands in cellular networks. It enables faster data transmissions, lower latency, and increased bandwidth efficiency as some of its improvements [2]. As shown in Figure 1.1, the amount of wireless data traffic is increasing exponentially [3].

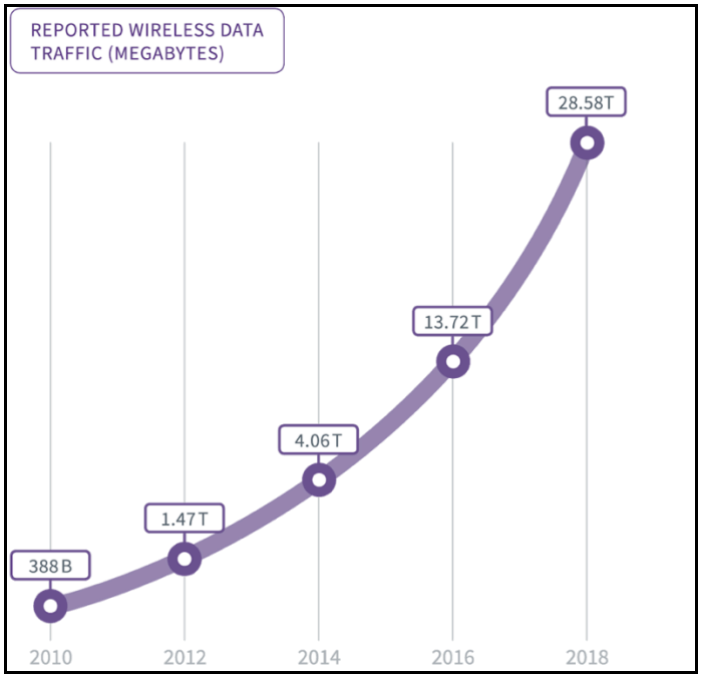


Figure 1. 1 The amount of reported wireless traffic in 2019 by CTIA and the image was adapted from [3].

The focus for the development of LTE was technical improvements that could be seen at the user end. Security against jamming attacks that can cause interruptions of service was not part of the development, and thus LTE networks are vulnerable to these types of attacks [2]. The motivation of this project is to show how jamming of an LTE network can be accomplished utilizing software-defined radios (SDR) and knowledge of network configurations [4]. People depend on their cellphones in everyday life, from getting directions to being their sole form of contact with others. Effective jamming can deny many people from gaining wireless network access, which can drastically affect their productivity and in some instances their safety.

Denial of Service (DoS) attacks have proven to be the biggest threat to LTE networks in recent years; it prevents legitimate users from accessing specific services by targeting hosting computer systems, network resources or the user devices [5]. On October 21st, 2016, a Distributed Denial of Service (DDoS) attack blocked Internet services for millions of subscribers on the Eastern seaboard of the United States [6]. The company that provides backbone services, Dyn, was under sustained attack against their DNS infrastructure, causing serious interference with users' access to major services such as Twitter, Amazon, Tumblr, Reddit, Spotify, and Netflix. Dyn estimated that there were up to 100,000 malicious endpoints, and most of which were originated from Mirai-based botnets. Although this scenario was a network, it gives an example of how losing access to wireless communications can affect people's lives on a large scale. Military LTE networks could also experience DoS attacks that could potentially result in serious security damages on a larger scale. Researchers suggest that LTE technology could be used for several military applications, including for garrison, strategic core, and tactical edge [7]; LTE is increasingly playing a crucial role in supporting military operations with vital enterprise services.

1.2 Current State of the Art

Open-source software, such as srsLTE [8] and OpenAirInterface [9], has enabled the testing, analysis, prototyping, and commercialization of LTE systems for researchers and other interested parties

[8], [9]. In combination with SDRs, these tools allow for the creation of a local LTE network that can be used to implement and analyze the effects of certain jamming techniques.

Jamming can be accomplished through four main methods; partial band, single tone, multi tone, and asynchronous [10]. These different types of jammers are detailed in Table 1.1 and illustrated in Figure 1.2. The timing of the jamming attacks can also vary by being either constant, random, or reactive. This means that the jammer could be constantly sending noise and trying to interfere, randomly interjecting noise, or listening to when the channel is in use to create an attack [11].

Table 1. 1 Description of different types of wireless jammers. Adapted from [10].

Jammer Type	**Description**
Partial Band Jamming (PBJ)	Transmission of noise over a specific LTE band.
Single Tone Jamming (STJ)	Jam a single subcarrier by creating an impulse of noise.
Multi Tone Jamming (MTJ)	Jam multiple subcarriers by creating multiple impulses of noise.
Asynchronous Single Tone Jamming (ASTJ) or Asynchronous Multi Tone Jamming (AMTJ)	Uses a signal with frequency offset to the subcarrier in order to create Inter Carrier Interference.

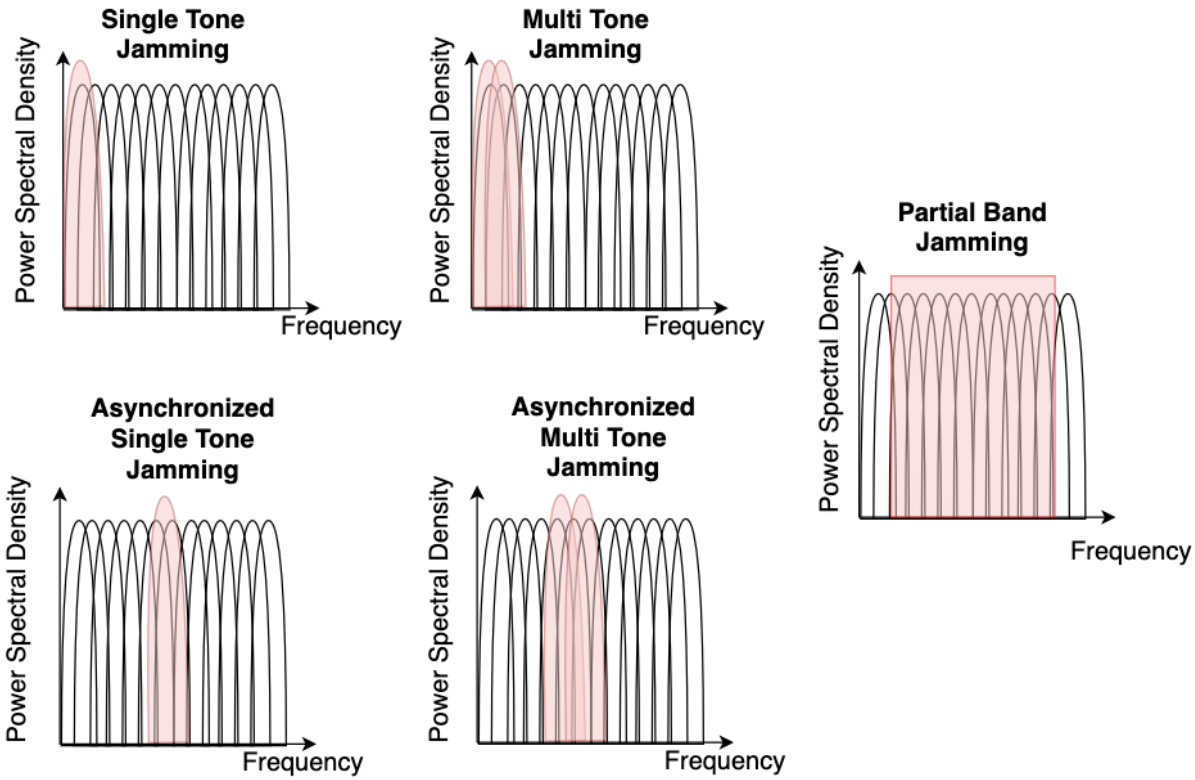


Figure 1. 2 The different types of jammers. Imaged was adapted from [10].

There are many theoretical jammers that could exist given the possibilities. Practical jamming has also been implemented and studied. In reference [12], analysis of jamming utilizing SDRs on LTE systems found that certain theoretical methods were not possible. The primary findings included that the main synchronization signals are the most resilient to jamming while the cell reference signals, Physical Downlink Control Channel (PDCCH), and Physical Control Format Indicator Channel (PCFICH) are the most vulnerable [5]. Reference [13] studied the areas that LTE could be susceptible to jamming attacks in its physical channels and signals [13], while in [14] provided details about the possible places for jamming attacks and then implemented a jammer by raising a created signal's power level until the connection was dropped for the user.

1.3 Technical Challenges

Most studies that are related to jamming attacks in the physical layer evaluate the threats and effectiveness of jamming by analyzing the system models [15], [16]. There needs to be more practical experiments on jamming attacks that can simulate malicious jamming towards different signals in the physical layer. This will help future researchers to acquire a better understanding of the vulnerability of LTE network downlink channel and how a STJ can affect communication.

1.4 Contributions

Implement Software-Defined Radios:

The LTE network and jammer testbed will be implemented using SDRs. The open-source software OpenAirInterface was used to create and program the eNodeB, User Equipment (UE), and core network. The jammer used GNURadio [17] to interfere with the downlink communication between the UE and eNodeB.

Build a Single Tone Jammer (STJ):

STJ was used instead of Partial Band Jamming Partial (PBJ) as previously mentioned. Despite the fact that PBJ requires no synchronization to the network and lower power pulse, STJ would be harder to be detected while jamming [11]. STJ also provided higher efficiency for jamming because of its characteristic of consuming lower power jamming and high levels of DoS that could be possibly be achieved by jamming one cell-specific reference signals, such as Primary Synchronization Signal (PSS) or Secondary Synchronization Signal (SSS) and could reduce the overall system capacity.

Learn Potential Vulnerabilities in LTE Network for Jamming:

The STJ method requires transmitting on top of specific physical channels, and thus, requires synchronization with the network to determine where the physical channels and specific signals exist in the frequency and time domain.

1.5 Report Organization

This report is organized into six chapters, the Introduction, Tutorial on LTE, Proposed Approach, Methodology, Results, and Conclusions. The Introduction chapter presents the motivation for the project, the current state of the art for LTE technology, as well as a brief analysis of the current technical challenges to provide context for how this project is related to prior and ongoing research. The Tutorial on LTE chapter provides information on different uplink and downlink channels and signals in the LTE physical layer and an analysis of the their different vulnerabilities against jamming. In the Proposed Approach chapter, the report introduces the process of identifying and evaluating possible solutions, the general approach to implement the project and the logistics for the project regarding time planning. The Methodology chapter lays out the detailed implementation plan of the project and the test for result verification. The Result chapter presents the findings. Finally, the Conclusion chapter summarizes the project contributions and provide recommended directions for future work.

2. Tutorial on LTE

This chapter begins with an overview of the history of cellular communications. It then goes into detail about the different aspects of an LTE system. The physical layer with its various signals and channels are explained in greater detail. The next part of the chapter focuses on the vulnerabilities that LTE systems may have to jamming attacks. The last part presents more information about how OpenAirInterface operates.

2.1 Overview of LTE

Mobile telecommunication systems began in the 1980s and continues to evolve from then to present day. Figure 2.1 shows this evolution and highlights the main functions in each generation. The first generation systems (1G) used analog techniques to transmit. Its market was limited to business use due to its small capacity and the user equipment (UE) being expensive as well as bulky. More consumer friendly products came in the early 1990s with 2G, which consisted of voice transmission and later included short message service (SMS). Global System for Mobile Communications (GSM) became the most used 2G system [2]. 2.5G was built in order to accommodate the growth of the internet that was also happening during this time. It introduced a core network packet switched domain and modified the air interface to handle data as well as voice. 3G was developed due to the demand for increased data rates, and was dominated by the Universal Mobile Telecommunication System (UMTS). UMTS was developed from GSM with changes made to the air interface but core network remained the same. It used a different technique for radio transmission and reception than 2G. Upon first launch it was not able to deliver what was promised until 3.5G which took off around 2005. The main improvement was enhancements for data applications through high speed packet access [2].

The need for data was rapidly increasing, especially since the UEs available in the market were becoming more user friendly. 2G and 3G networks were getting congested and had to maintain two core

networks for voice and data packets [2]. The reduction of latency was also needed since delay times reached 100ms [2]. UMTS and GSM also were extremely complex due to the need to be backwards compatible with the addition of new features [2]. In 2004 3GPP began developing what is LTE today. The goal being to have higher data rates and reduced latency [2].

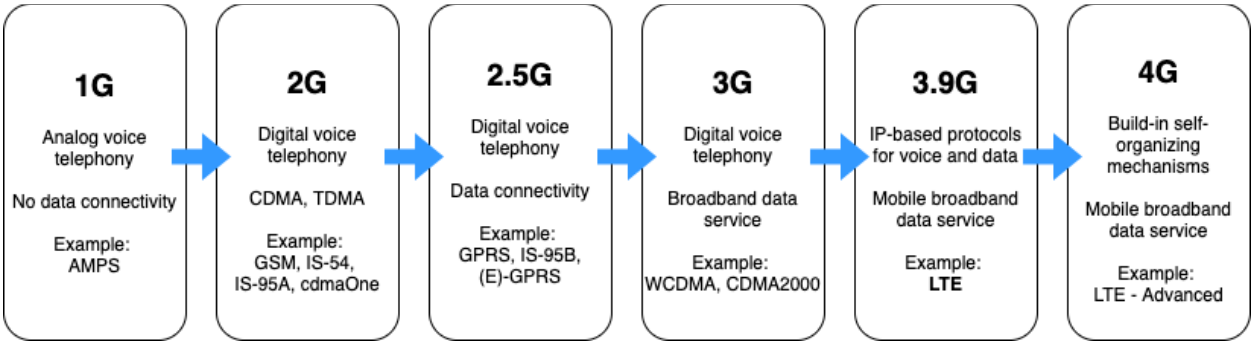


Figure 2. 1 Mobile telecommunications has improved and expanded its functionalities with each new generation. This summarization of the key features was adapted from [16].

The main components of an LTE network includes the UE, evolution Node B (eNodeB), and the Evolved Packet Core (EPC) [2]. The UE can include any mobile device that the user possesses that can connect to the network. Figure 2.1 shows how these components are connected.

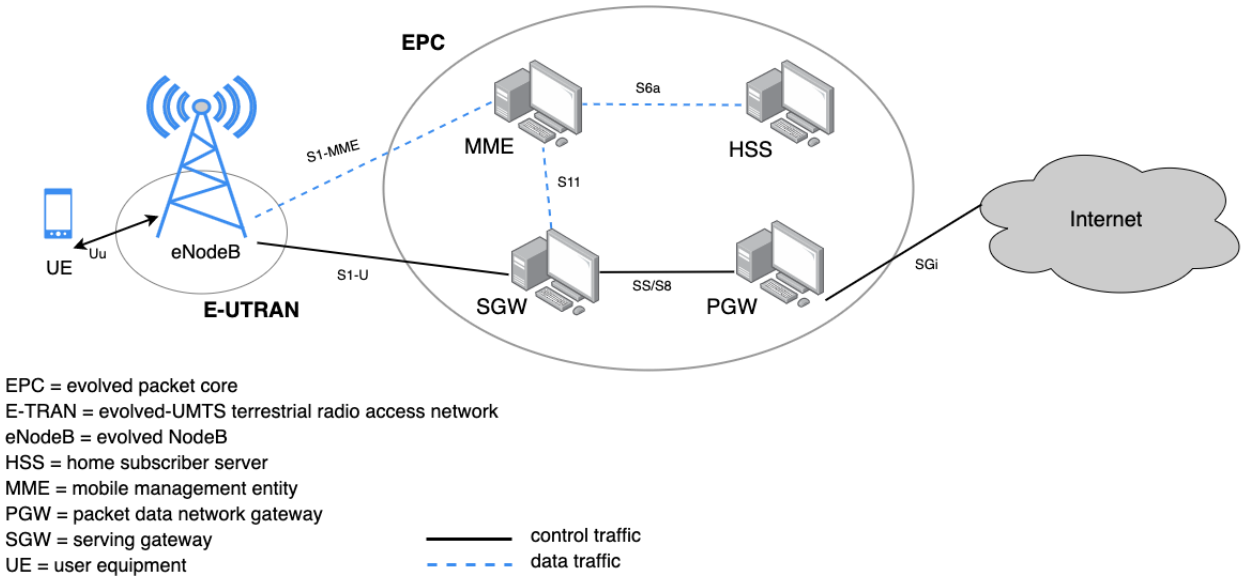


Figure 2. 2 The overall LTE system has three main components: the UE, eNodeB, and EPC. Adapted from [2].

The eNodeB is where the decision making and processing of data from the user occurs. It translates the UE data into a format that can be transmitted by the EPC and reformats data from the EPC to the UE. The air interface functionality is controlled by eNodeB. It handles multiple UEs and optimizes the quality of wireless link for each UE. The high data rates are achieved through the use of Multiple-Input Multiple-Output (MIMO) and multiple antennas. The eNodeB does the complex computations to reduce the amount that would need to be performed by the UE. The framework includes the physical layer abstraction, medium access control (MAC), radio link control (RLC), packet data convergence protocol (PDCP), and radio resource control (RRC) which can be seen in Figure 2.3.

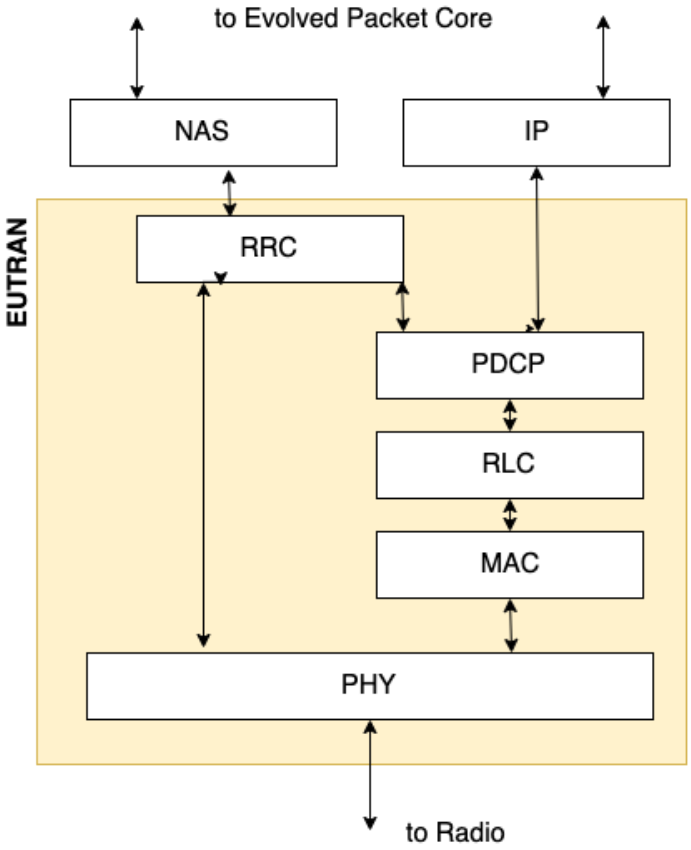


Figure 2. 3 The network layer of the LTE network illustrating the communication lines between the different framework components. This image was adapted from [16].

The EPC has four main components that each perform different functions. The Serving Gateway (SGW) handles user data packets through routing and forwarding. The Packet Data Network Gateway (PDN-GW) provides connection to the internet by being a point of exit and entry of traffic for the UE.

The Mobility Management Entity (MME) is the key control node for the LTE access networks. The Home Subscriber Server (HSS) performs mobility management, user authentication, and access authorization. The type of information processed is handled by the MME and the HSS while the user data is handled by the SGW and the PDN-GW [16].

2.2 LTE Physical Layer

The LTE physical (PHY) layer creates the connection between the eNodeB and the UE through signals and control channels. There are two main forms of communication which are the downlink (DL), which is from the eNodeB to the UE, and the uplink (UL) communication from the UE to the eNodeB [13]. Communication in the downlink and uplink are accomplished through a variety of different signals and control channels as shown and described in the following tables.

Table 2. 1 Description of major downlink physical channels. Adapted from [10].

Physical Downlink Shared Channel (PDSCH)	Carries the data and signaling messages from the Downlink-Shared Channel and paging messages from the Paging Channel.
Physical Broadcast Channel (PBCH)	Contains the Master Information Block
Master Information Block (MIB)	Carries information about the operating bandwidth, frame number, and PHICH.
Physical Multicast Channel (PHCH):	Carries multimedia broadcast/multicast service data of the Multicast Channel (MCH).
Physical Downlink Control Channel (PDCCH):	Carries the Downlink Control Information (DCI) which consists of mainly scheduling commands and scheduling grants.
Physical HARQ Indicator Channel (PHICH)	Carries ACK/NACK information about the uplink which allows UE to decide whether to transmit new data or retransmit the previous data
Physical Control Format Indicator Channel (PCFICH)	Carries information about the frame structure.

Table 2. 2 Description of major downlink physical signals. Adapted from [10].

Primary Synchronization Signal (PSS)	Carries information about the physical layer cell identity and is used by the UE for connecting to the cell.
Secondary Synchronization Signal (SSS)	Carries information about the physical layer cell identity and is used by the UE to connect to the cell.
Cell-Specific Reference Signal (C-RS)	Sent by eNodeB to support channel estimation at the UE.
MBSFN Service Ref Sig (MBSFN-RS)	The reference signal for the Multimedia Broadcast Multicast Service (MBMS) and is used for channel estimation to the UE.
UE-Specific Reference Signal (UE-RS):	For channel estimation to the UE.
Positioning Reference Signal (P-RS)	Sent by eNodeB for location based service.
Control Format Indicator (CFI)	Carries information about the number of symbols allocated for channel control.

Table 2. 3 Description of major uplink physical channels. Adapted from [10].

Physical Uplink Shared Channel (PUSCH)	Carries the data and signaling messages from the Uplink-Shared Channel (UL-SCH) and carries Uplink Control Information (UCI) to ensure the UE is not transmitting at the same time.
Physical Uplink Control Channel (PUCCH)	Carries the UCI in case the UE needs to send only control information.
Physical Random Access Channel (PRACH)	Carries the random access transmissions from the Random Access Channel (RACH).

Table 2. 4 Description of major uplink physical signals. Adapted from [10].

Demodulation Ref Signal (D-RS)	Sent by the UE to the eNodeB for channel estimation.
Sounding Ref Signal (S-RS)	Configured by the eNodeB for power reference in order to support frequency dependent scheduling

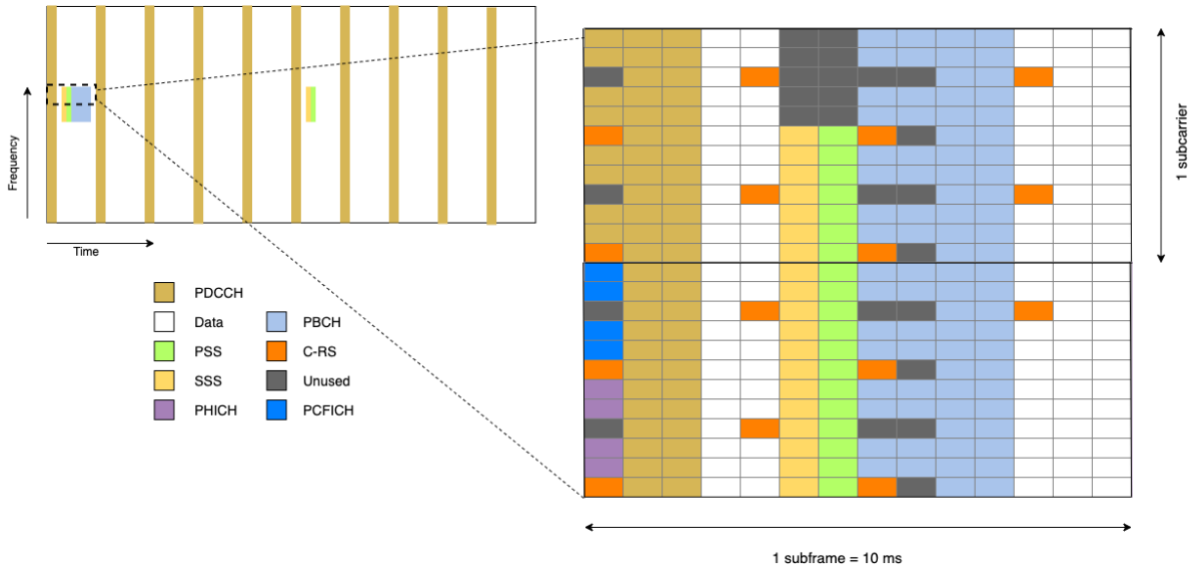


Figure 2. 4 The general downlink frame structure of key signals in an LTE system. Adapted from [13].

The downlink uses orthogonal frequency-division multiplexing (OFDM) modulation on the downlink and an orthogonal frequency-division multiple access scheme (OFDMA) [14]. OFDM, is a spectrally efficient way to transmit data because it uses different subcarriers to transmit a set amount of symbols. This is accomplished on the transmitter by first taking in a data stream from the upper layers and doing a serial to parallel conversion after modulation. This results in multiple different sine waves that can be added together to transmit one signal carrier waveform that contains all the data information. Figure 2.5 shows what an OFDM signal looks like in the frequency domain.

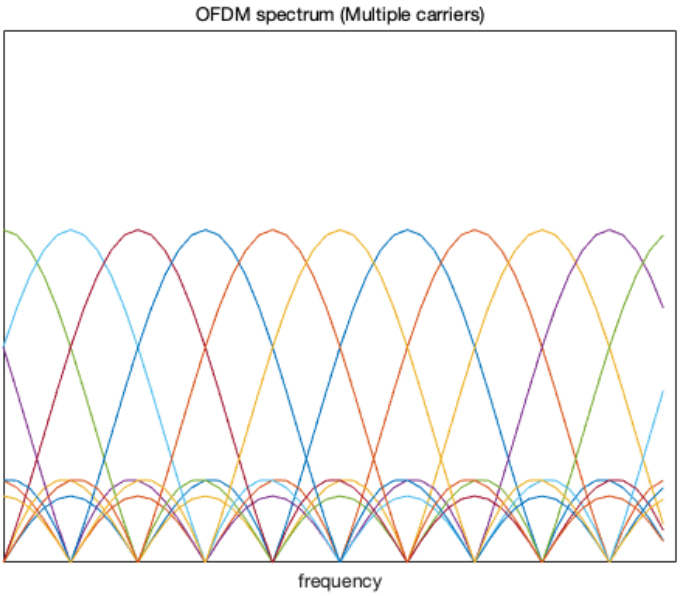


Figure 2. 5 The OFDM spectrum showing different subcarriers and how they look together in the frequency domain.

The frame structure of the DL is shown in Figure 2.2. It consists of multiple subcarriers, each a 180 kHz block, that are spaced 15 kHz apart from each other with a cyclic prefix of 5 us. OFDM symbols are transmitted in 0.5 ms slots and two slot forms a 1ms subframe and 10 subframes is a 10 ms frame [10]. All the symbols transmitted in a single slot on all the subcarriers forms a Resource Block [14]. The bandwidth can vary from 1.4 MHz to 20 MHz and thus the number of resource blocks available in a single slot varies from 6 to 110 [14].

The frame structure of the downlink in LTE consists of the control channels, reference signals, and synchronization signals transmitted at specific times and intervals. The PSS and SSS are transmitted two times per frame at the center 62 subcarriers and the PBCH is transmitted once in a frame at the center 72 subcarriers [14]. The RS are distributed throughout the frame to enable channel estimation in both the time and frequency domain. The control channels are transmitted at the start of every subframe with a width varying from 1 to 4 OFDM symbols depending on the channel bandwidth and the control information in that subframe. This structure can be seen in Figure 2.2 and shows the different signals and what they look like in a frame and then at a specific subcarrier.

The UE goes through a procedure in order to connect to the eNodeB. The PSS is first searched for and must be obtained in order to detect the Physical Layer Identity and symbol timing. The UE, once gaining information about the symbol timing, can detect the SSS in order to get the cell identity group and subframe number in order to derive the cell-ID of the eNodeB to which it is attached. Once the cell-ID has been obtained, the UE performs channel estimation and equalization by finding the C-RS. After, it decodes the PBCH to get the MIB which carries information about the operating bandwidth, frame number, and PHICH duration and resources configurations. The UE now has broadcast messages available to decode the PCFICH to get the CFI value. The UE has enough information about the lattice in the frame where it has to search for PHICH and PDCCH. The UE then decodes the PDCCH in order to gain information about the resources allocated in the PDSCH and will be able to decode the data [13].

The uplink uses single carrier frequency multiple access (SC-FDMA) [18]. This is similar to OFDMA, except the discrete Fourier transform (DFT) is done prior to the inverse fast Fourier transform (IFFT) causing the spread of data symbols over all subcarriers creating a virtual single carrier structure. This results is a lower peak-to-average power ratio which benefits the UE in its power efficiency [18]. The frame structure of the UE is shown in Figure 2.3. The D-RS, which is utilized by the PUSCH and PUCCH for channel estimation, is distributed throughout the frame. The SRS is transmitted in one OFDM symbol to help the eNodeB to measure the received power across a wide transmission bandwidth. The PUCCH symmetrically transmits at both edges of the BW and the PRACH transmits over a bandwidth of 6 resource blocks with a duration from 1 to 3 subframes with its position being configured by the eNodeB [10]. The rest of the block consists of the PUSCH.

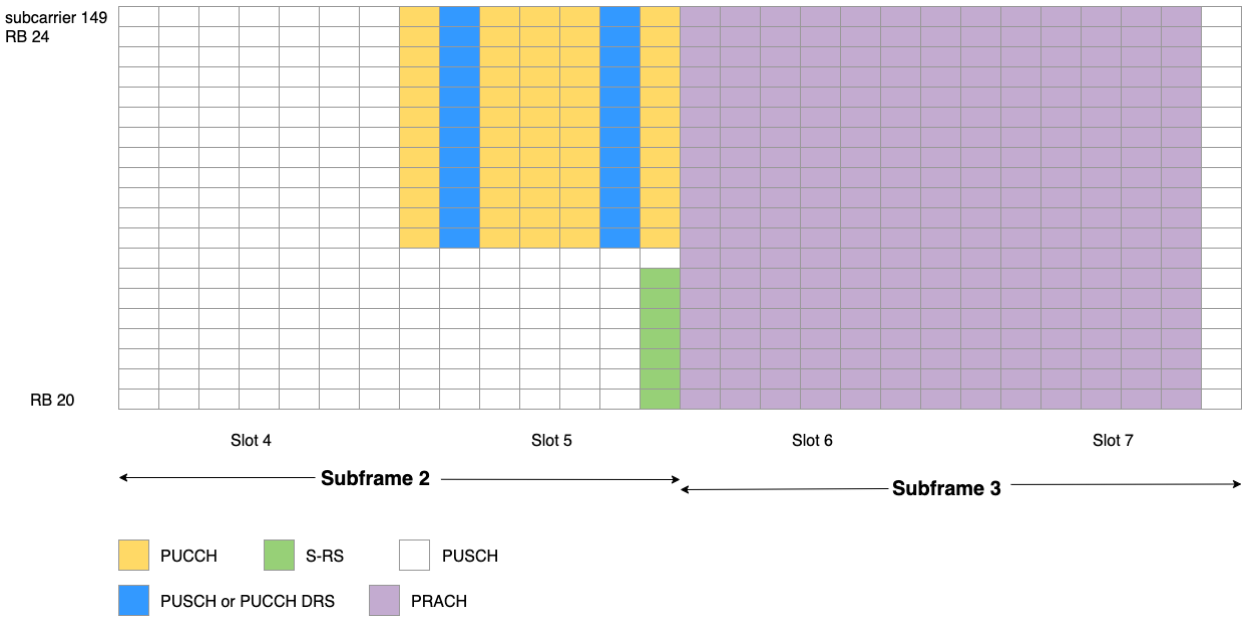


Figure 2. 6 The LTE Uplink Frame Structure that includes key signals. Adapted from [10].

2.3 LTE Vulnerabilities

As mentioned in the previous chapter, several ways of jamming are partial band jamming (PBJ), single tone jamming (STJ), multi tone jamming (MTJ), asynchronous single tone jamming (ASTJ), and asynchronous multi tone jamming (AMTJ). In this section, the vulnerabilities of different channels and signals in both uplink (UL) and downlink (DL) are analyzed, as well as potential methods of jamming and their evaluation, in order to find the suitable "weak spot" for efficient jamming in the LTE communication.

Two potential signals that could be jammed are the PSS and SSS which are the synchronization signals in the downlink. Since the positions of these two signals are fixed in the frame structure, PSS and SSS can be jammed by a continuous jammer that transmits at the desired frequency [10]. However, this strategy requires the highest power of the jamming signal, which makes it easy to detect the jammer. A more effective method of corrupting the PSS, mentioned in [13] and [14], is RF spoofing, referring to transmitting a fake signal meant to masquerade as an actual signal. The jammer will transmit a bogus PSS asynchronously to the LTE frame at higher power that prevents the terminal from detecting the SSS or decoding the Master Information Block (MIB) of the network. Even though disrupting the PSS or SSS

will not cause an immediate Denial of Service (DoS), but it could instead prevent new UEs from accessing the cell and idle UEs from re-synchronizing with the cell [12].

Another potential weakness in LTE signals are the Cell-Specific Reference Signals (C-RS) which carries downlink pilot symbols that are used for channel estimation, quality assessment, and equalization [10]. C-RS is located throughout the frame in both time and frequency domain, based on the cell ID as well as antenna port number (MIMO). If C-RS is jammed, the bit error rate of the complete network would increase tremendously. Studies in [12] have proven that configuring perfect synchronization is difficult to achieve for a sparse and non-contiguously distributed signal such as the C-RS. It is suggested that the jammer can target CRS subcarriers with MTJ instead since the C-RS subcarrier locations depend on the cell ID and remain constant for all users of the cell. Jamming C-RS also requires the jammer to synchronize with the LTE network and knowledge of PSS and SSS. Due to the long symbol duration (71 microseconds), one benefit for jamming C-RS is that there will be a short propagation delay, for example, if there are approximately 5 miles between the jammer and the UE, there would only be a propagation delay of 27 microseconds [19]. Researchers have recommended that the jammer could start transmitting a fraction of a symbol early to compensate for this delay [16].

Channels that contain important information are also vulnerable to jamming attacks. The Physical Broadcast Channel (PBCH) contains the Master Information Block (MIB), which is required by the UE to gain information about the downlink bandwidth, resource length of the Hybrid ARQ (HARQ) Indicator Channel (PHICH), and the System Frame Number (SFN) for frame synchronization. Thus, the UE could get initial access to the cell [12]. As shown in Figure 2.2, PBCH is located at center 72 subcarriers and appears in the first subframe of every frame. Without PBCH, the UE will not be able to decode PHICH, which carries ACK/NACK information for the uplink. Researchers in [10] and [20] suggest that jamming PBCH does not necessarily require the jammer to synchronize with the cell if the jammer is continuously transmitting at center 72 subcarriers, and this jamming attack is characterized by a low duty cycle and a fairly low bandwidth. Since the characteristics of the PBCH requires a fairly high-power interfering signal to deny the service to non cell edge users, the jammer using the method will be bounded by the large

transmitted power at the eNodeB and the potentially low transmitted power of the jamming device in order to outpower the legitimate signal [20].

Another channel that is susceptible to jamming attacks is the Physical Uplink Control Channel (PUCCH) which is used to send the eNodeB a variety of control information. Due to its feature of mapped to the resource blocks on the edges of the system bandwidth jamming attacks can be implemented after acquiring knowledge of the LTE system bandwidth and center frequency [13], [14], [19] (see Figure 2.3). It is important to note that uplink jamming has an impact on the entire cell as opposed to locally around the jammer [13]. This is because when the jammer attacks PUCCH, it prevents the eNodeB from receiving essential uplink signaling messages required for the correct operation of the cell, and overwhelms eNodeB reception. In such a way, the jammer will effectively prevent the base station to communicate with every user equipment (UE) in the cell, and thus, extending the range of the attack to the entire cell, as shown in Figure 2.7 below.

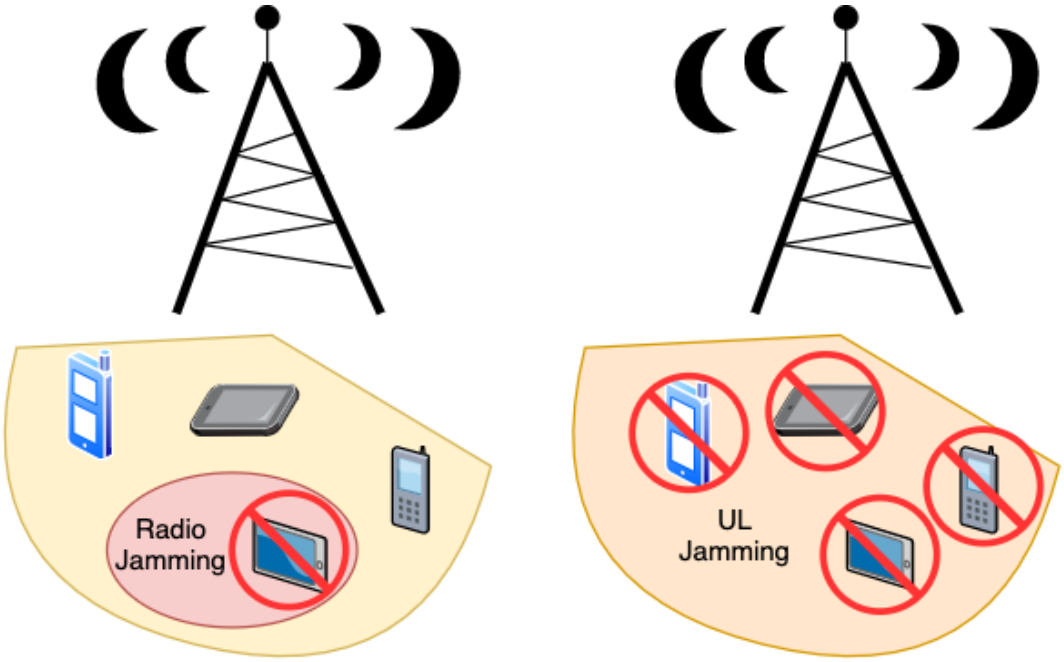


Figure 2. 7 Impact range of regular radio jamming versus uplink jamming. The impact of jamming the network can cause severe consequences. Adapted from [20].

LTE has three control channels in downlink, namely; Physical Control Format Indicator Channel (PCFICH), Physical Downlink Control Channel (PDCCH), and Physical Hybrid ARQ Indicator Channel (PHICH). Although it is possible to jam the PDCCH directly, we will first analyze the possible jamming method of the PCFICH. PCFICH is the key to the process of transmitting and decoding the information as mentioned above. It appears infrequently in the downlink frame structure; only in the first orthogonal frequency-division multiple access (OFDM) symbol in each subframe and occupies a total of 16 resource elements (RE) [13]. Since the location of the 16 subcarriers is determined by the eNodeB's complete cell ID, jamming the PCFICH requires the jammer to synchronize to both downlink synchronization signals PSS and SSS. This also limits the jamming to only one cell [19]. The PDCCH carries critical control information, [13] and [20] have suggested that successful jamming the PDCCH requires transmitting at high power as well as synchronization with the cell. In addition, since the PDCCH size varies between one and three OFDM symbols, the jammer needs to decode the PCFICH first in order to launch an effective attack with the least amount of power [13]. Through experiments, [12] implies that since jamming the PCFICH has the same outcome as jamming the PDCCH, jamming the PDCCH seems to be more impractical for the jammer since the PCFICH's sparsity makes the jamming attack more complex, and thus, less likely to be anti-jammed. If PHICH is jammed, the downlink communication will not suffer, but the uplink communication will because it carries ACK/NACK information of the uplink which enables the UE to make decisions [10]. It is worth mentioning that although PHICH jamming requires the jammer to synchronize with the cell, its sparsity location on the frame structures makes the jamming more complex and threatening.

The user data could also be potentially jammed. The Physical Downlink Shared Channel (PDSCH) and Physical Uplink Shared Channel (PUSCH) are used to transmit user data to and from the eNodeB, which means that the jammer can eavesdrop and take the advantage to get the complete system information. While it is possible to jam these two channels, it is suggested that the jamming process requires the jammer to extensively decode the protocol with control information and user information, which almost equals the effort to jam the entire LTE signal [10], [13]. This makes it an extremely

complex attack that might be considered a combination of jamming and cyber-attack. Therefore, it is not recommended to jam the PDSCH and PUSCH [13].

2.4 Open Air Interface (OAI)

The OpenAirInterface (OAI) emulation platform is an open-source software platform developed based on real 3GPP LTE protocols that provide a better approach to testing, evaluating, and validating wireless communications and signal processing. It is written in C and provides a complete wireless protocol stack that implements the PHY, MAC, RLC, PDCP, RRC as well as providing an IPv4/IPv6 network device interface under Linux [21]. There are two different modes that can be used, either single state machine or multi-state machine emulation [4]. Single state machine has virtualization of network nodes within one physical computer. The multi-state machine uses distributed deployment on so that multiple machines can be used and transmits information via IP address. For example, the eNodeB could be on one machine while the UE is on another and they would need to be on the same local network and communicate via their machine's IP address.

The source code can be found on OpenAirInterferace's Github and is divided into three main folders. The PHY layer and other related parameters resides in the *Openair1* folder and it also provides interface to the MAC layer. The hardware interface can be found in this folder and its main function is to realize the baseband signal process. The *Openair2* folder contains the realization for the upper network layer including the MAC, RLC, PDPC and RRC. The folder *Openair3* has network modules that uses the IP protocol. It offers interfaces to the applications which makes the whole platform more complete and practical. The EPC resides in another folder called *openair-cn* and this is where the MME, SPGW, and HSS can be run.

The working procedure of the emulator can be divided into four steps, as shown in Figure 2.5 [22]. The four consecutive can be defined as Emulation Scene Description, Initialization, Execution, and Output.

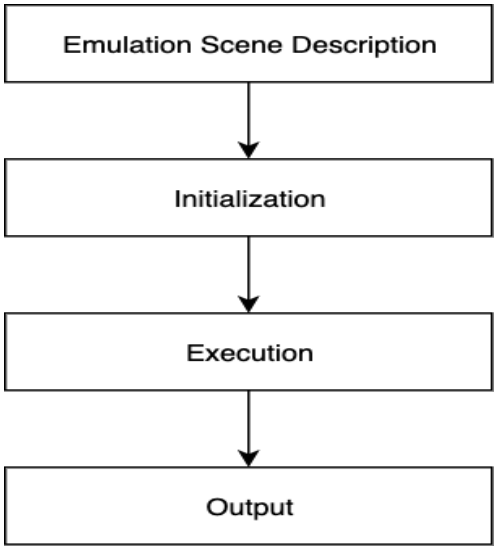


Figure 2. 8 The emulation workflow of OpenAirInterface is divided up into these four steps emulation scene, initialization, execution, and output. Adapted from [23].

The first step of emulation is to build an emulation scenario depending on the users' input. The parameters include single or multi- channel modes, the number of eNodeB and user equipment (UE), working mode (TDD, FDD) and many other systems and environment variables [23]. The next step is initialization. This is the emulation platform procedure of configuration which uses the user defined parameters from the previous step [4]. It contains emulation scenario initialization and each layer of eNB and UE configuration [4]. Also, it includes traffic and mobility initialization [24]. The execution step indicates the execution of the emulator, and the synchronization of the emulated nodes will be run in the experiment. The last step is the monitoring which is done through an outputted log file. The file contains the whole procedure of emulation that has been collected, labeled and archive for the usage of analyzing the emulator and evaluating the experiment. Through experiments, [23] tested the feature of OAI on testing LTE systems and concluded that this platform can be useful and helpful for the learning and researching of LTE and LTE-A standards, performance evaluation of new scheduling algorithms, improving the accuracy of link-level simulation by using the whole protocol stack, mobility management protocols in cellular networks, etc.

3. Proposed Approach

3.1 Problem Statement

In this project, the objective was to effectively jam an LTE system to disrupt or deny service to the user equipment (UE) using Software Defined Radios (SDRs). The selection process for which types of jamming to implement depended on its efficiency and complexity. Efficiency refers to the power level of the jamming signal that was being transmitted in comparison to the effectiveness of jamming, namely the amount of transmission disruption that was caused by it. Complexity measured the level of difficulty that the jamming would cause to mitigate LTE communication or anti-jam. Despite the goals, due to the budget, time limitation, and feasibility consideration was given to factors such as whether synchronization was needed when selecting the most suitable jamming approach.

3.2 Evaluation

In order to effectively jam an LTE system to cause a denial of service to the UE, there are several possible approaches:

1. Sniffing
2. Spoofing
3. Jamming uplink channels
4. Jamming downlink channels

Sniffing involves eavesdropping on important signals which could potentially give valuable information leading to more efficient attacks. Spoofing is able to create a jamming attack by making a signal that masquerades as legitimate LTE signal. Jamming of the uplink channels would involve creating another signal and transmitting at the same time as the actual signals to cause interference. Jamming of the downlink channels is similar to that of the uplink signals except the downlink signals would be targeted.

Each approach was analyzed by its efficiency, complexity, and feasibility. Spoofing was recommended to target the synchronization signals in the downlink, primary synchronization signal (PSS) and secondary synchronization signal (SSS) [13]. In the simulation, [13] observed that the UE lose connection with the eNodeB when the spoofing attack maintained at a high power level. Thus, spoofing was not considered efficient due to its high ratio of the received jamming signal power to the received LTE signal power. On the other hand, sniffing created a denial of service (DoS) scenario by producing an eavesdropping environment using other low-cost software radios. It was recommended in [13] that sniffing techniques should be used during the downlink Physical Broadcast Channel (PBCH) phase in order to acquire system information block (SIB) that indicated the complete configuration of the cell and other critical information of the mobile network, and thus, to identify the specific cells that are deployed for critical communications and distinguish them from mobile operator eNodeBs. However, due to the fact that there was only one UE in the LTE testbed, user-targeted DoS would not provide additional benefits compared with other approaches. Furthermore, the decoding process of the SIB would be more focused on software rather than the LTE physical layer.

As mentioned in the previous chapter, the LTE physical layer possesses many vulnerable channels to jamming attacks. The uplink jamming would be able to effectively prevent the base station to communicate with every UEs in the cell, as well as extending the range of the attack to the entire cell. However, due to the fact that there was only one UE in the LTE testbed, the uplink jamming would not be as effective as it should be. Additionally, successfully jamming the uplink control channels requires transmitting at high power as well as synchronization with the cell, which makes this approach not efficient enough compared with others [13], [24].

As mentioned in the previous chapter, downlink jamming possesses several options that included PSS, and SSS, Physical Broadcast Channel (PBCH). PSS carries information about the physical layer cell identity and plays an essential role in the process of UE connecting to the cell. Consequently, jamming the PSS would prevent new UEs from accessing the cell and re-synchronizing with the cell, which assures the complexity of the attack. Due to the fact that it is transmitted once in a frame at the center of 72

subcarriers, the PSS jamming attack only would require narrowband jamming signals, which means this attack is highly efficient [10]. Therefore, it was decided to use synchronized signal-tone jamming on the PSS in the downlink.

3.3 Proposal Approach

An initial plan was devised after considering the advantages and disadvantages of each potential type of jamming. The first decision was to begin with partial band jamming because it was easy to implement and also allowed us to test and confirm that the equipment was working as anticipated. The next step was to determine how to measure the success of the jamming. The method that was chosen was to count how many packets were received and compare that to the number transmitted. This would have made it possible to determine the effect of the jamming via a quantitative approach. After this initial testing of the verification and jamming were completed, it would then be beneficial to move to more complex jamming attacks. This would involve setting up the synchronization of the jammer to the eNodeB. However, due to time constraints and the complexity of synchronization, different jamming approaches were evaluated again in order to find a more effective and feasible jamming approach for the testbed, as shown below in Table 3.1. After the evaluation, the decision was made to do frequency hopping as the primary form of jamming. Frequency hopping could still be narrow band jamming but it is easier to accomplish since there is no need for synchronization. It would also be more difficult to anti-jam because the frequencies it hops to would to randomized.

Table 3. 1 Evaluation of different jamming types

Types of Jamming	**Pros**	**Cons**
Partial Band Jamming (Wideband Jamming)	• No synchronization required	• Easy to be detected • Need constantly high power
Synchronized Jamming	• Know the signal or channel that is being attacked	• Requires synchronization • Requires the knowledge of the exact frequency of the subcarrier • LTE could easily recover by transmitting at another frequency
Asynchronized Jamming	• No synchronization required	• Do not know the signal or channel that is being attacked
Frequency Hopping Jamming	• No synchronization required • Hard for anti-jam because of its unpredictability	• Do not know the signal or channel that is being attacked

3.4 Project Planning

Microsoft Excel was used to plan out objectives throughout the project. The Gantt Chart in Figure 3.1 and Figure 3.2 shows the tasks and timeline. WPI has a quarter system with each academic year made up of four seven-week terms, and this project was conducted throughout the first two terms, A-term 2019 and B-term 2019. Throughout the design stage of the project, several deadlines are mentioned and noted in red on the Gantt Chart. The project team size changed from four members in the first term to two members in second term. In the first term, the focus was on designing and building the testbed. Specifically, it was broken into multiple subtasks, such as setting up the software interaction between the transceiver, USRP B210, UE, and the jamming device, USRP N210. After each segment has been set up separately, they have to be consolidated into the testbed, in which they were evaluated using a spectrum analyzer in order to ensure the transmission. The Agilent CSA Spectrum Analyzer N1996A was used and it has a frequency range from 100 kHz to 3 GHz. From then on, the project moved to the jamming stage during B-term 2019, as shown in Figure 3.2. After acquiring information on the network, the USRP N210 was synchronized with the eNodeB and partial band jamming (PBJ) was then performed to collect the effect of the jammer. Then, the frequency hopping jamming approach was implemented with GNU Radio.

The results of jamming attempts were collected and evaluated to determine whether it was effective by analyzing the throughput before and after the jammer. During the process of setting up the testbed and testing jamming, the report recorded the work done in the implementation and testing stages. A live demo was shown to conclude the project.

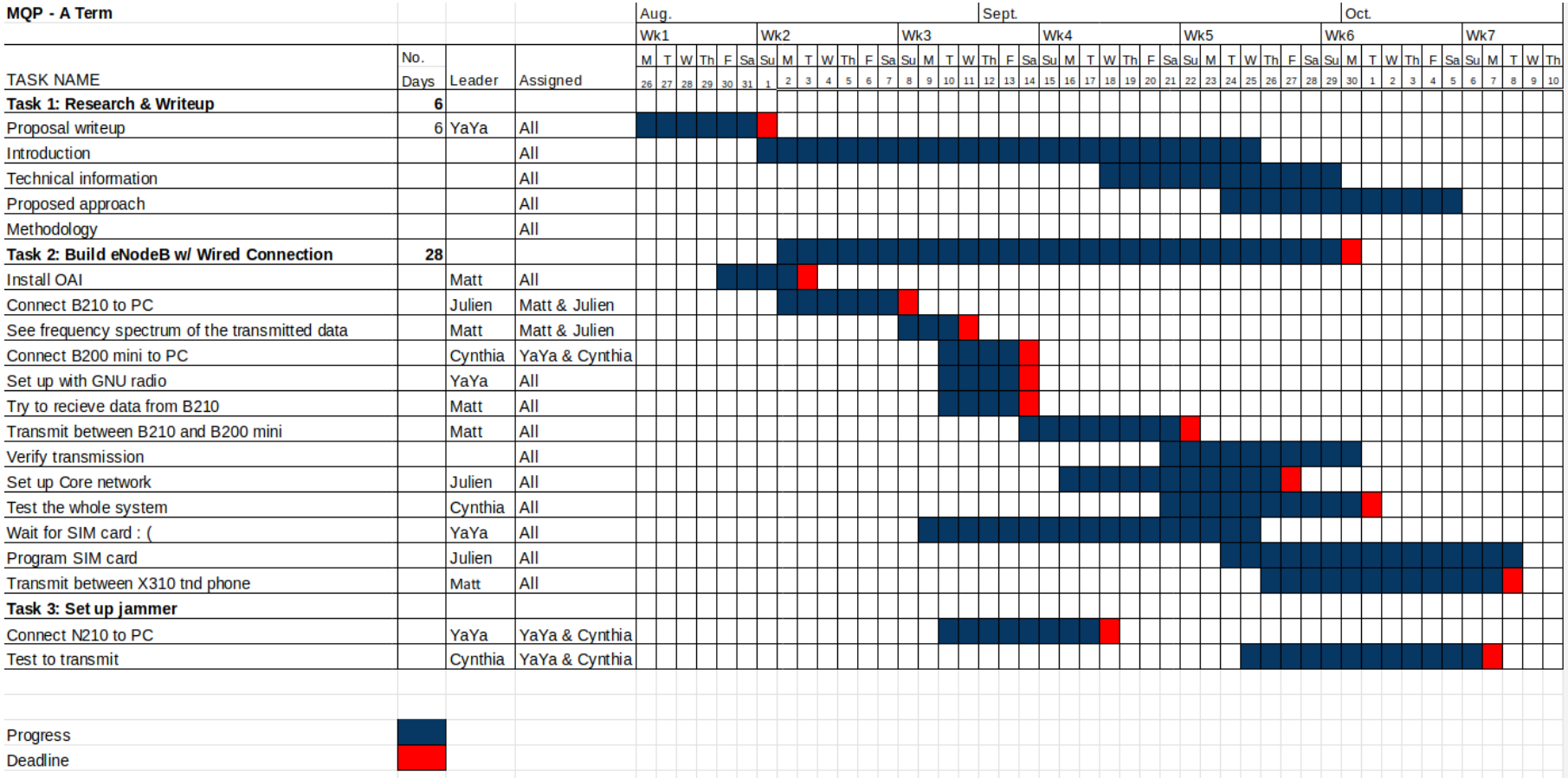


Figure 3. 1 Gantt Chart for A term 2019

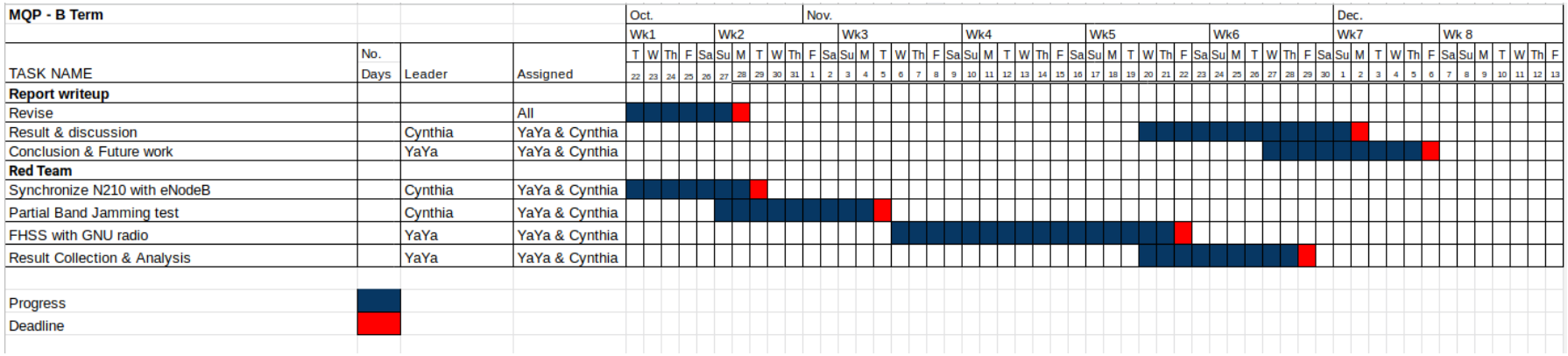


Figure 3. 2 Gantt Chart for B term 2019

4. Methodology

This chapter describes the methods that were used to successfully jam LTE communications. The first section focuses on the creation of an LTE testbed using two USRPs and a Samsung s4 phone. It also includes information about OpenAirInterface. The next section goes into detail about partial band jamming and how that was implemented through GNU Radio. The section afterward explains how frequency hopping was achieved and provides insight into the GNU Radio code. Lastly, the method that was used to verify the results of the jammer is discussed.

4.1 Testbed Implementation

The testbed consisted of an eNodeB, User Equipment (UE), and jammer which is shown in Figure 4.1. A USRP B210 was used for the eNodeB, a Samsung S4 for the UE, and a USRP N210 for the jammer. The whole testbed was built on a 64-bit computer with an Intel Core i7-4770K CPU @ 3.50 GHz x8 processor with Ubuntu 18.04.3 LTS operating system. The B210 was connected to the computer through USB 3.0 and the USRP N210 was connected through a 1 Gigabit ethernet cord. The UE was connected to the network by an open-cell SIM card that was programmed and its information inputted into the Home Subscriber Server (HSS) in OpenAirInterface (OAI). The testbed setup was located in a Faraday cage due to the restriction of frequency bandwidths in the U.S. The Faraday cage works as an enclosure to block electromagnetic fields, and thus the wireless transmission was limited within the cage and not interfering with other signals of the same frequency.

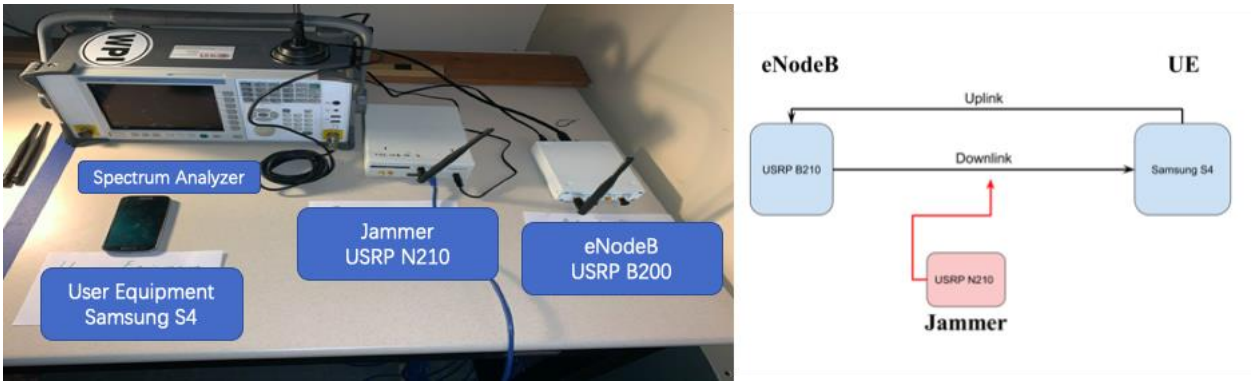


Figure 4. 1 The implemented testbed that was used is shown on the left. The testbed structure (right) shows the theoretical connections from each component of the testbed.

The testbed was built following the two tutorials available online [25], [26], and the overall steps are shown below in the flow diagram Figure 4.2. Reference [25], [26] provided instructions for the installation of the original files on Github, configuration, and Linux command lines to run for each step. For the network setup, each node in the EPC network was on a separate IP address as shown below.

HSS is on localhost: 127.0.0.1
eNodeB is on 127.0.0.10
MME is on 127.0.0.20
SPGW is on 127.0.0.30

Due to the fact that the UE was a smartphone device, a SIM card purchased from Open Cells was used for the UE [27]. The SIM card is configured for 3GPP: GSM, WCDMA, LTE with a mileage algorithm to perform 3GPP standard subscriber identification and it came with a card reader/writer to personalize and configure the card [27]. The SIM card was then programmed (instructions on [25]) and then its information was added into the network match with the HSS database.

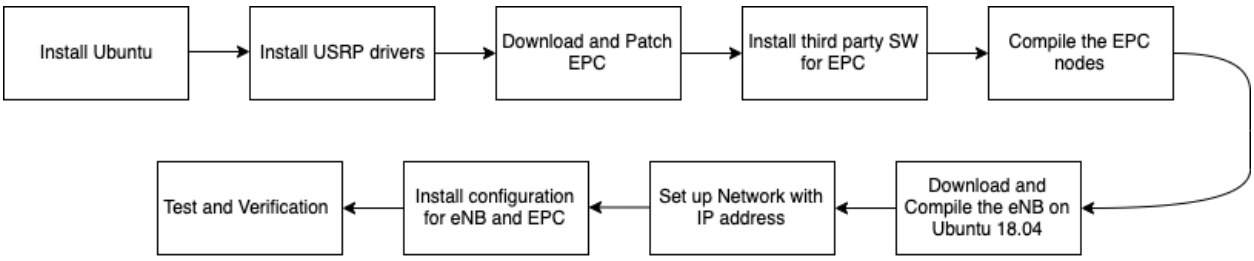


Figure 4. 2 OAI tutorial steps flow diagrams adapted from [26].

In order to run the testbed, each node (HHS, MME, SPGW) needs to be turned on separately before the eNodeB. Four different command windows were opened and each of the command lines are the following [26]:

```
cd openair-cn; source oaienv; cd scripts; ./run_hss
cd openair-cn; source oaienv; cd scripts; ./run_mme
cd openair-cn; source oaienv; cd scripts; sudo -E ./run_spgw
sudo bash; cd ~/openairinterface5g; source oaienv; cd cmake_targets/lte_build_oai/build./lte-softmodem -O ~/opencells-mods/enb.10MHz.b200
```

After which the UE should be connected and it should attach to the network and be able to reach the internet through the OAI network.

4.2 Partial Band Jamming Test

GNU Radio was used to implement the partial band jammer. The output of this jammer was a signal that was spread across many frequencies and jammed any signals that operated at those frequency levels. The flow graph that was created in GNU Radio to generate a particular partial band jamming signal is shown below in Figure 4.3. The sample rate was chosen to be 20 MHz for the USRP N210 to be able to operate properly. If the sampling rate was lowered it caused an error stating that there were not enough samples for the N210 to transmit a signal.

The first block in the flowgraph is a signal source that was chosen to be a cosine wave and its parameters were unchanged. The signal was then passed through the OFDM Modulation block in GNU Radio. This would allow the signal source to be spread across multiple frequencies. The parameters of the block were changed in order to change the bandwidth of the signal. The parameters that changed the bandwidth were the FFT length and occupied tones. The FFT length was set to 4096 because it was a large enough length to have a satisfactory resolution but not enough to overwhelm the computer. The occupied tones parameter was modified to change the bandwidth. Through testing, it was found that as the number of occupied tones decreased the bandwidth of the signal increased. The Frequency Sink GUI block was added to the output of the modulated signal to determine the value of the occupied tones in order to obtain the desired bandwidth. Table 4.1 shows each of the bandwidths that were tested and the

occupied tones that were used accordingly. The general method to obtain a certain bandwidth was adding or subtracting at increments of 30 to the occupied tones. This would then respectively add or subtract 200 kHz to the bandwidth.

The UHD USRP Sink block was connected to the resulting modulated signal which had the address of the N210 was specified in order for GNU Radio to connect to the radio and output the signal. The center frequency was chosen to be 2.56 GHz because that is the center frequency of the LTE downlink signal. The gain value was set to 28 dB because it was the highest power level the N210 could transmit.

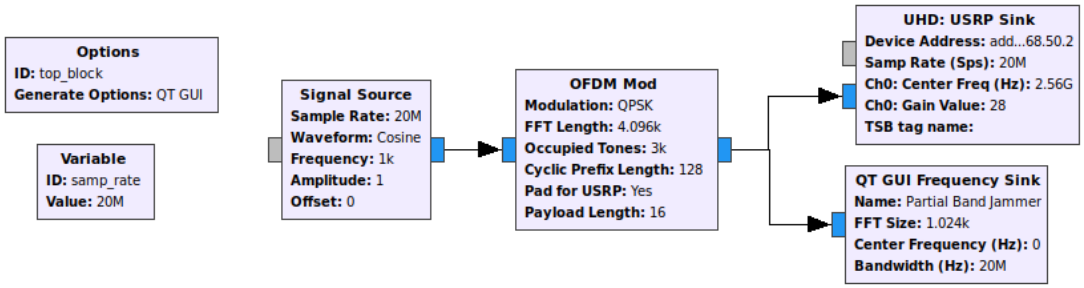


Figure 4. 3 The flow GNURadio flow graph for partial band jamming shows the steps of generating the signal, modulating it, and then sending it to the USRP and GUI.

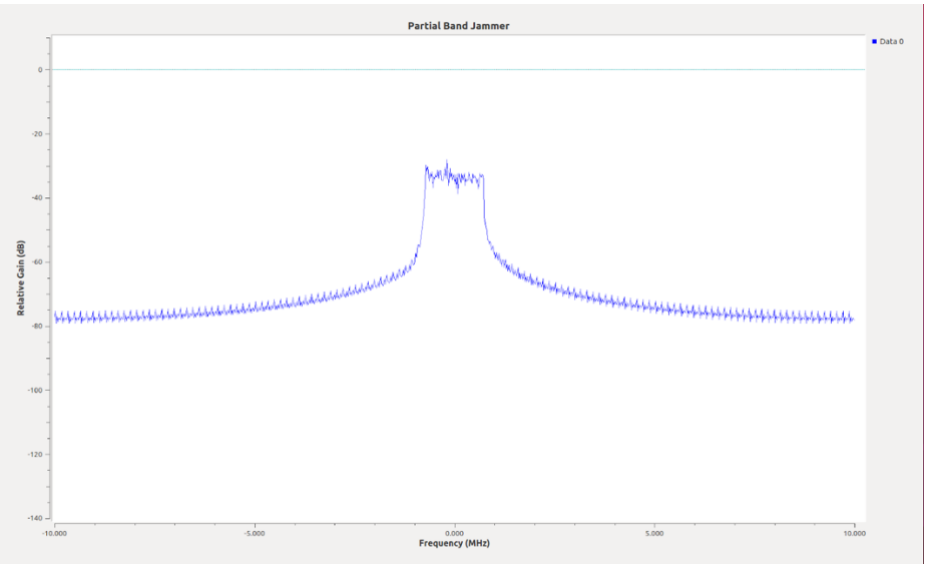


Figure 4. 4 A graph showing the partial band jammer signal that has a bandwidth of 2 MHz.

Table 4. 1 Bandwidth of partial band jamming

Occupied Tone	Jamming Signals Bandwidth
90	600 kHz
120	800 kHz
150	1 MHz
300	2 MHz

4.3 Frequency Hopping Jamming

Frequency hopping was implemented in order to achieve narrowband jamming. Narrowband jamming would decrease the number of frequency levels that the outputted signal occupies and be harder to detect. The wideband jamming GNU Radio blocks were utilized and the bandwidth was set to 400 kHz and 600 kHz for two different tests. A visual of the GNU Radio flowgraph for the frequency hopping jammer can be seen in Figure 4.5. The original flow graph was modified by the addition of another signal source, a probe block, and a function probe block. The probe signal block is able to take measurements of a signal at a poll rate specified in the function probe block. The type of measurement was also specified in the function probe block to be amplitude levels. The signal source added was set to be a square wave and acted as a clock for the probe. This was needed for the probe signal to have a reliable source to take amplitude measurements since a square wave can only have amplitude measures of 1 or 0. A sine wave, if not sampled at exactly the peaks and valleys, might be slightly off from an exact value of 1 or 0. The

values needed to be exact for the function probe block and it stored these values of the polled measurement from the probe signal to a variable called `val`.

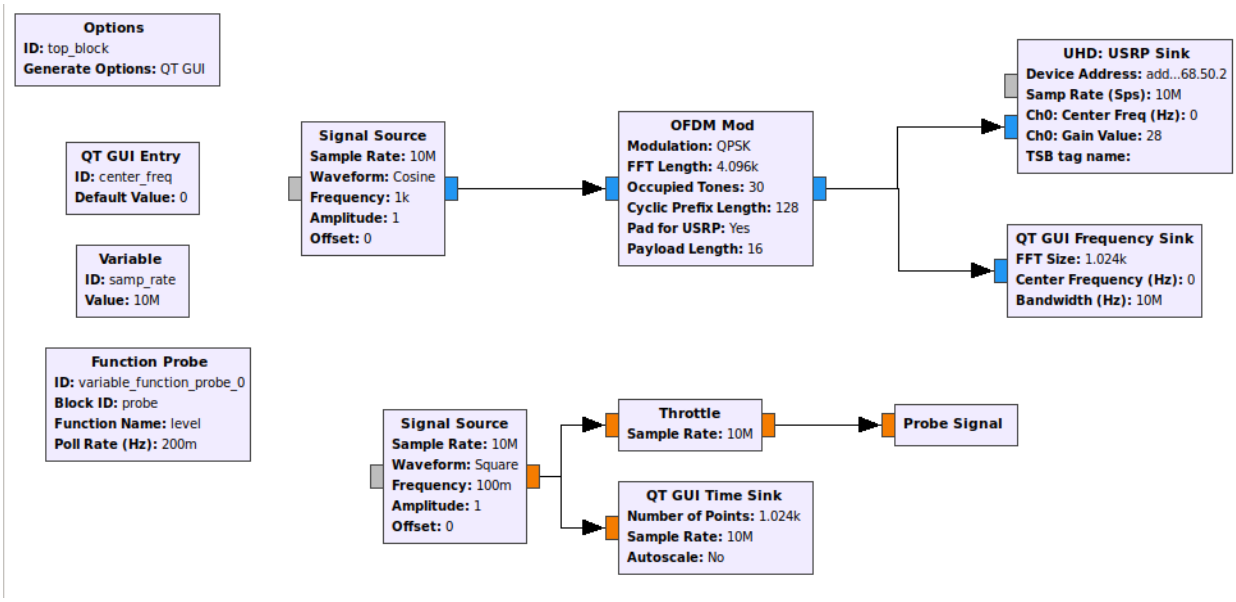


Figure 4. 5 The GNU Radio flowgraph for narrowband jamming via frequency hopping includes the elements of the partial band jammer (top) with the bandwidth set to 200 kHz. The addition of the clock blocks (bottom) served to change the center frequency of the original

After setting up the flowgraph in the GNU Radio Companion the top_block code, that was generated from the flowgraph, was modified to change the center frequency of the USRP sink to a random value relative to the clock rate. The justification for making these changes include the need to change the center frequency in order to "hop" to different frequencies. Without this block the frequency would stay at one value and never change. The function probe block function, as shown below in its original form, was modified.

```
def _variable_function_probe_0_probe():
          while True:
             val = self.probe.level()
             try:
                 self.set_variable_function_probe_0(val)
             except AttributeError:
                 pass
             time.sleep(1.0 / (10))
```

The modified function, that changes the center frequency randomly according to the clock is shown below with the changes highlighted. The addition included an if statement that depended on the

value of the amplitude measurement the probe signal obtained, hence why these measurements had to be exact values. If the value of the amplitude (`val`) was a 0 then the center frequency would change randomly, using the python random function, otherwise, it would remain unchanged from its previous value. As shown in the code, the range of the center frequency was from 2.677 GHz to 2.6825 GHz at a step size of 1 kHz. The module for random functions needed to be added to the top_block as well thus the call `import random` was also added at the beginning of the file.

```
def _variable_function_probe_0_probe():
        while True:
            val = self.probe.level()
            if val == 0:
                self.set_center_freq(random.randrange(2.6775e9,2.6825e9,1e3))
            try:
                self.set_variable_function_probe_0(val)
            except AttributeError:
                pass
            time.sleep(1.0 / (40))
```

The relationship of the signal clock rate, f_{clock}, and the probe polling rate, $f_{polling}$, is shown in Equations (1) and (2). The clock signal source is the f_{clock} parameter and the $f_{polling}$ parameter is found in the python code top_block in the function probe function line **time.sleep(1.0/(40))** where the value of 40, in this instance, is the $f_{polling}$ rate. These parameters were used to determine how often the center frequency would change. The value of $f_{polling}$ was designed to be twice the value of f_{clock} so that the center frequency would change only when the amplitude of the clock signal was 0.

$$f_{clock} = 1T \quad (1)$$

$$f_{polling} = 2 * f_{clock} \quad (2)$$

A waterfall plot was generated after the frequency hopping jammer code was executed to verify the signal jumped from different frequency levels as shown in Figure 4.6. This figure shows time across frequency and as expected the signal changed frequency as time increased. The frequency hopping jammer was changing the frequency of the signal at a rate of 50ms. The reason that the signals are a dark red color is being they are being transmitted at a high power, specifically 28 dB.

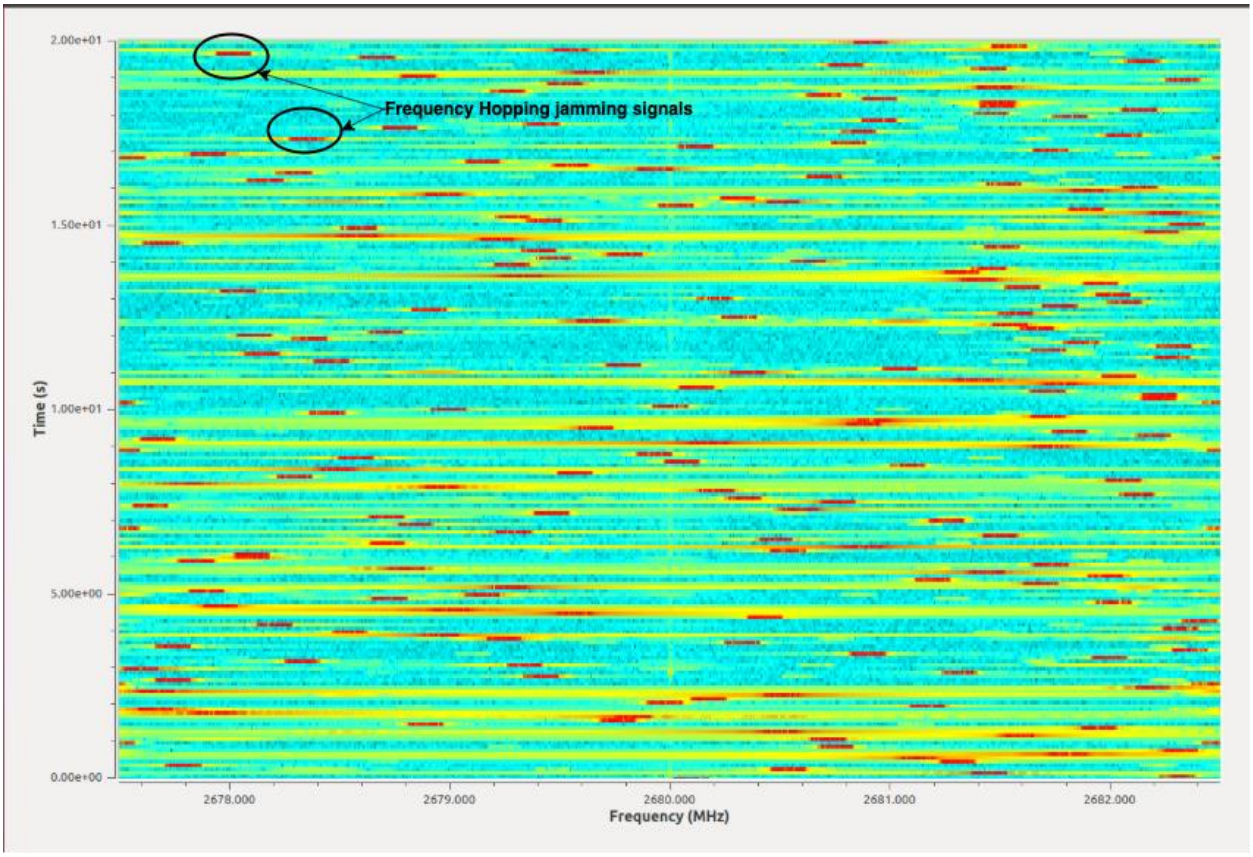


Figure 4. 6 Waterfall graph for frequency hopping jamming signal that is centered at 2.68 GHz. The narrowband jammer is shown in red due to its high power and changes every 50 ms.

4.4 Result Verification

The indication and evaluation of jamming attack effects were accomplished by measuring the packet receiving rate by the UE and analyzing packet loss. Packet loss occurs when one or more packets of data traveling across the LTE testbed network fails to reach the UE due to poor connection or inference from an outside source such as a jammer. When packet loss happens, the throughput from the eNodeB data is reduced because the packets were never received by the UE and the number of packets successfully received decreases. Packet loss causes retransmission of the packets, and a great amount of loss could result in delays of the network and even disconnection. The correlation between the jammer effects and LTE network packet loss was therefore utilized to prove whether the jammer could cause any damage to the network, and if so, to measure the damage the jammer caused.

In order to test whether the jammer was having an effect on the LTE system, several mobile applications on the UE that indicate the number of packets that were received from eNodeB were tested, such as Mobile Insight and iPerf. These applications all required rooting the phone and were not compatible with the Samsung s4. It was then decided to browse and analyze the transmission traffic from the eNodeB using Wireshark [28]. Wireshark is an open-source software, which can be used as a network protocol analyzer. It was compatible with Ubuntu and used to analyze the packets that the UE was attempting to receive and showed when a packet was dropped or when retransmission occurred. The packet information outputs messages as well as a graph that showed the number of packets per second across time. Both these methods were used to analyze the jamming effects. A YouTube video was streamed on the UE for each jamming test in order to keep the transmission consistent.

4.5 Chapter Summary

In this chapter, details about the testbed and the jamming methods used in this project were discussed. Once the LTE testbed was built, the eNodeB and UE were connected and the partial band jamming approach was implemented through GNU Radio. After successfully verifying that a partial band signal was produced by the testbed, the frequency hopping jamming was implemented through GNU Radio with different power levels and different bandwidths. All the results of jamming were measured by WireShark from the eNodeB by analyzing the rate of packets being received by the UE before and after the jammer was turned on.

Figure 4. 7 This image shows the Faraday cage that the experiments were completed.

5. Results & Discussion

In this chapter, the results and findings of the project are explained in detail. The chapter begins with the results of the partial band jammer and how much of an effect that had on the LTE system. It then moves to explain how well the frequency hopping jammer did in regards to packet corruption and the reasoning for this result. Lastly, the result and reasoning for the frequency hopping jammer to cause DoS is described in further detail.

5.1 Partial Band Jammer

The wideband jammer was tested first in order to verify jamming capabilities. The overall result was a Denial of Service (DoS) once the jammer was turned on. This was seen in WireShark by the retransmission messages and eventual loss of packets. The UE detached itself from the network due to the poor quality of the channel. Figure 5.1 shows the number of packets per second over time. The spikes in the graph represent a large number of packets following a time when no packets were being exchanged. This is due to the buffer used by YouTube. When a video is being streamed, bursts of packets are needed to be received instead of continuously since this would cause poor video quality if just one packet was lost or needed to be retransmitted. YouTube uses a buffer to combat this problem by getting a certain number of packets to fill the buffer and then when the video plays, packet transmission is stopped until the buffer has depleted to below a certain level [29]. This is shown in Figure 5.1 by the spikes, when the buffer is being filled, and then the time of no packets because the buffer is filled.

The jammer starts at the indicated dashed redline and very quickly the number of packets successfully received drops. It then eventually stops altogether indicating that the partial band was effective in causing a DoS scenario. Different bandwidths caused the signal to be completely lost or only degraded the channel but did not affect the UE enough for service to stop.

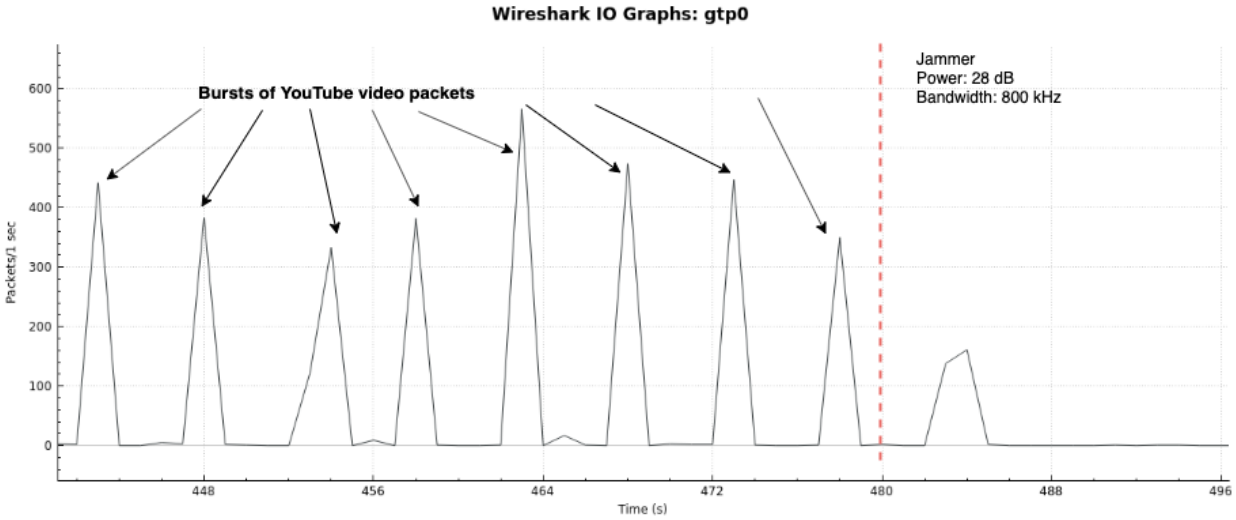


Figure 5. 1 Wideband jammer packet analysis results. After the jammer is on, the number of packets successfully transmitted drops to zero and the UE detached from the network.

5.2 Packet Corruption

The frequency hopping jammer had two different effects on the LTE system. Specifically, it achieved packet corruption and DoS. The first effect was packet corruption that caused the loss of a few packets and showed a decrease in the number of packets being successfully transmitted. This can be seen in Figure 5.2, where the jammer was started at the time indicated by the dashed red line. There is some delay when the jammer code was executed and when the device actually started transmitting, which is the cause for the delay in the effect. The time between packets increased once the jammer starts transmitting because a few packets are being dropped and retransmissions occur, which caused more time between packets.

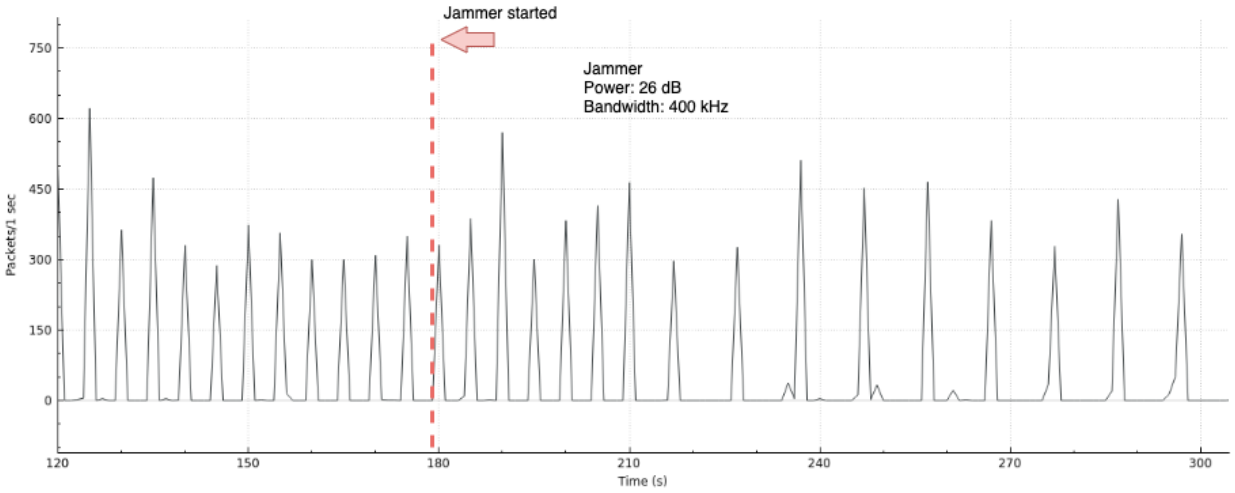


Figure 5. 2 The packet analysis result from frequency hopping jamming at 26dB, a bandwidth of 400 kHz, and at a clock rate of 50ms and a polling rate of 25ms.

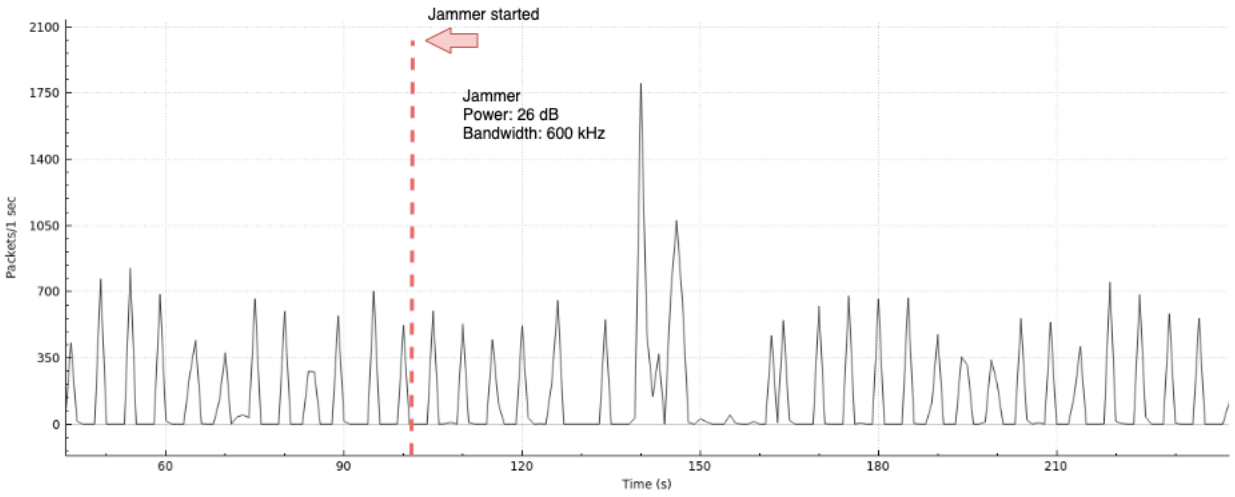


Figure 5. 3 The packet analysis result from frequency hopping jamming at 26dB, a bandwidth of 600 kHz, and at a clock rate of 50ms and a polling rate of 25ms.

Although packet corruption occurred the same could not be said about DoS. Throughout the time of testing, even though packets were being dropped, the video quality remained the same and the user was unable to notice a difference in service. This is due to the frequency hopping being randomized. There was no prior knowledge about which subcarriers the UE was using which means the frequency hopping was done by running through randomized subcarriers.

The randomness of the frequency hopping is illustrated in Figures 5.4 and 5.5. Figure 5.4 shows the waterfall graph of the LTE system with the darker orange parts being the PSS and SSS synchronization signals. The lighter yellow parts spread throughout the graph are the data signals. In Figure 5.5, the frequency hopping jammer had been turned on and the red spots are the jammer's signals. The signal is randomly scattered throughout the LTE system and the likelihood that it would drastically interfere with the phone signal, which is using one subcarrier of the LTE signal for communication, is small.

5.3 Denial of Service (DoS)

DoS was achieved by running the experiment multiple times until the phone disconnected in one case. This is shown in Figure 5.6 where the dashed red line indicates when the jammer was started and then there is one spike of delay. Then, the number of packets per second drops significantly until it drops to zero and the UE disconnected from the network. Figure 5.6 shows the output of the WireShark messages that shows an increasing amount of attempted retransmissions, which is consistent with the graph. This occurrence was due to chance because the frequency hopping was random and it just so happened to hit the subcarrier the UE was operating at and an important signal that caused the DoS.

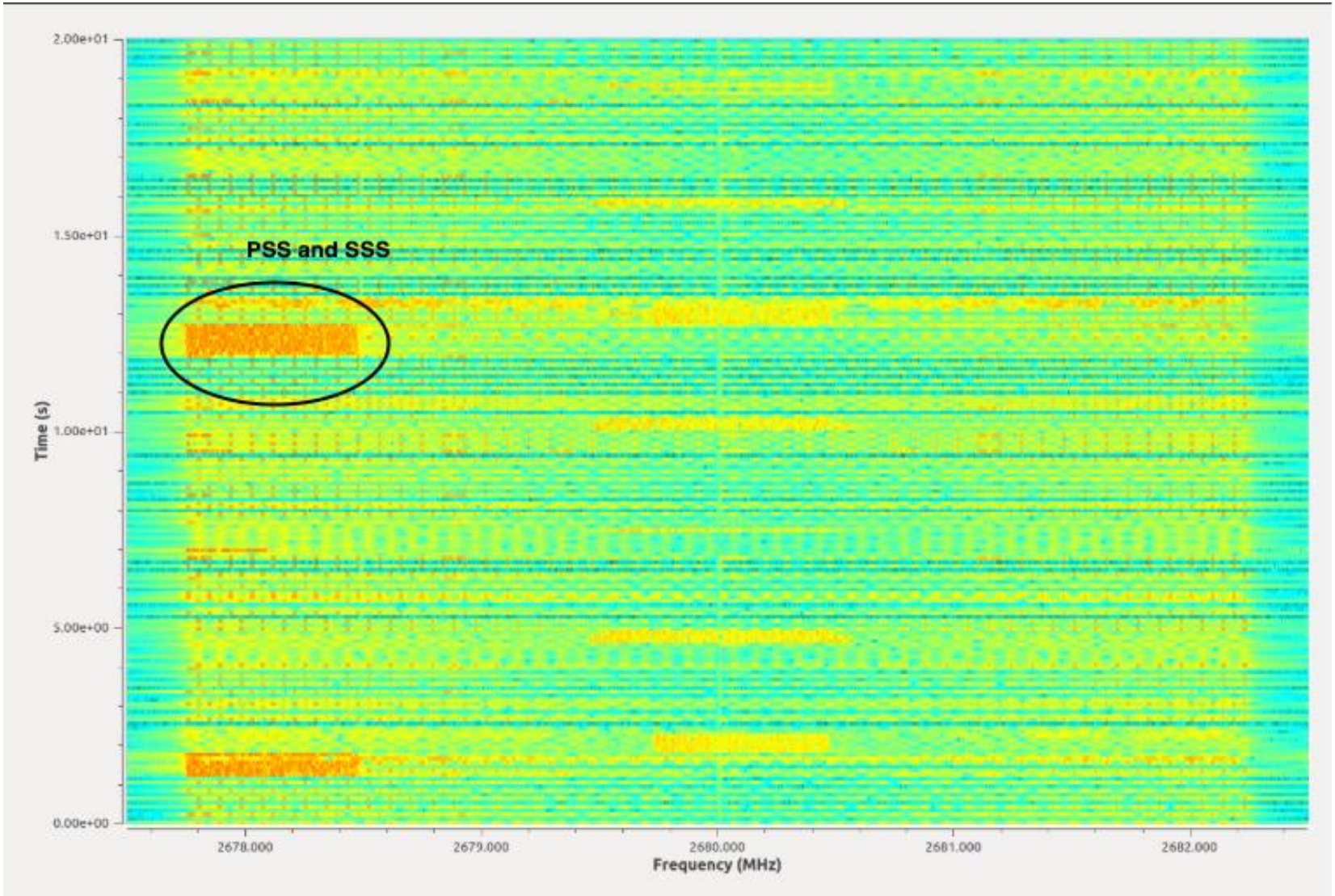


Figure 5. 4 The eNodeB LTE communication waterfall graph before any jamming has occurred.

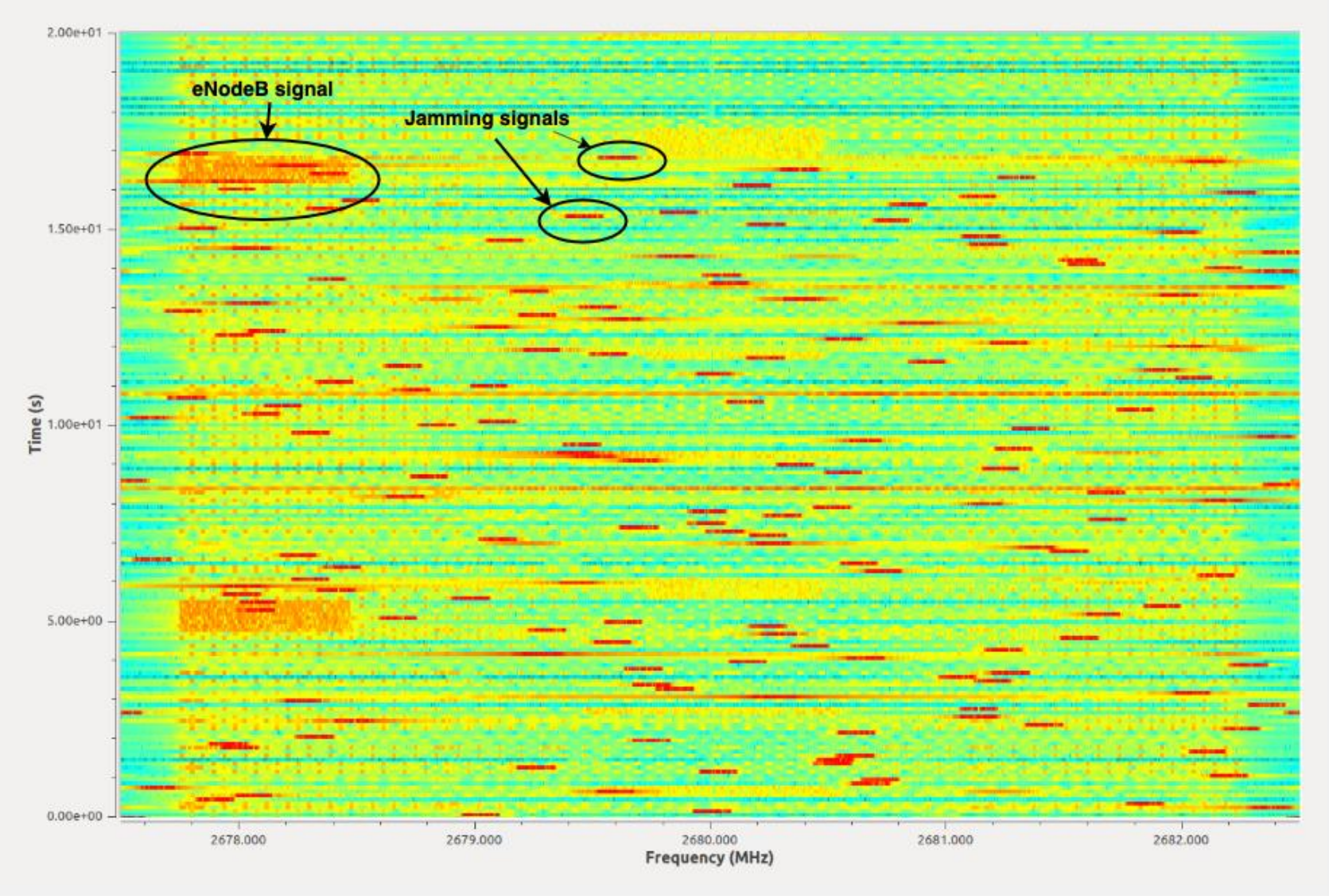


Figure 5. 5 The waterfall graph showing the eNodeB transmission after the frequency hopping jamming.

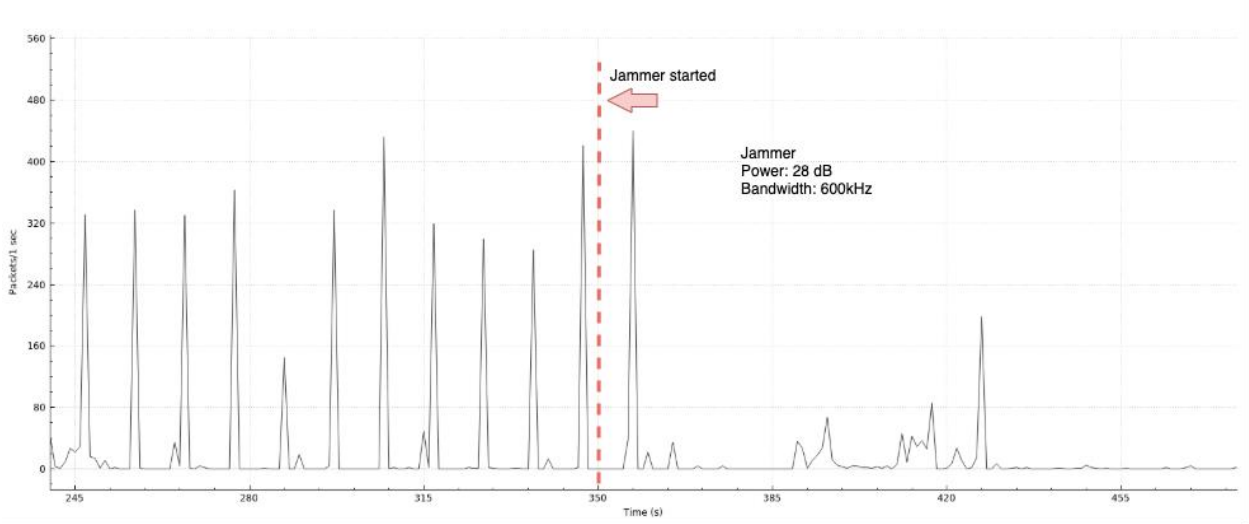
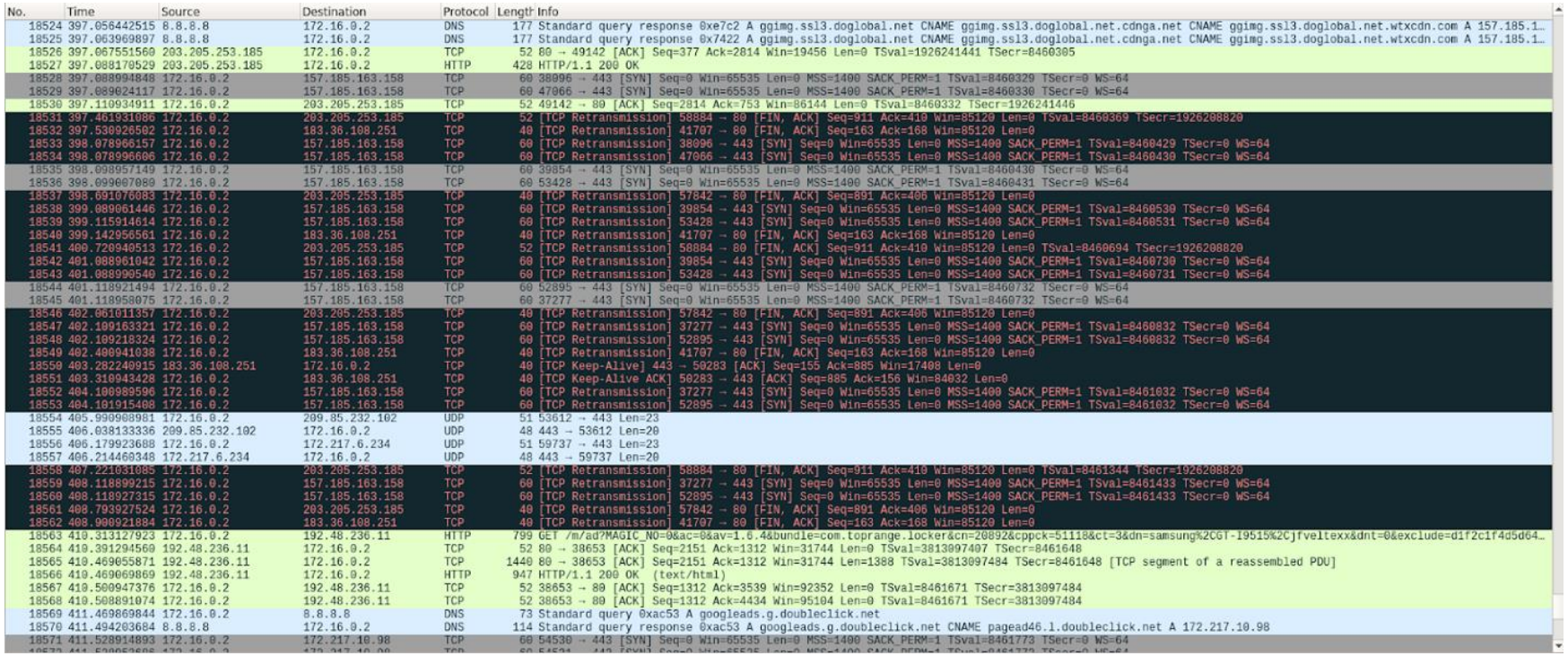


Figure 5. 6 Graph showing successful DoS caused by jammer at 28dB, a bandwidth of 600 kHz, and the clock at 50ms and a polling rate of 25ms.

5.4 Chapter Summary

In this chapter, the results obtained from this project were discussed. Through experiments, it was discovered that the frequency hopping jamming approach could disturb the testbed by causing packet retransmission or packet disruption at a lower power of 26 dB. When jamming signal power was 28 dB, frequency hopping jamming attack was able to create a DoS for the network. Overall, it had been proven that the frequency hopping jamming attack can be destructive to the LTE network when the power is 26 dB or higher.



No.	Time	Source	Destination	Protocol	Length	Info
18524	397.056442515	8.8.8.8	172.16.0.2	DNS	177	Standard query response 0xe7c2 A ggimg.ssl3.doglobal.net CNAME ggimg.ssl3.doglobal.net.cdnga.net CNAME ggimg.ssl3.doglobal.net.wtxcdn.com A 157.185.1…
18525	397.063969897	8.8.8.8	172.16.0.2	DNS	177	Standard query response 0x7422 A ggimg.ssl3.doglobal.net CNAME ggimg.ssl3.doglobal.net.cdnga.net CNAME ggimg.ssl3.doglobal.net.wtxcdn.com A 157.185.1…
18526	397.067551560	203.205.253.185	172.16.0.2	TCP	52	80 → 49142 [ACK] Seq=377 Ack=2814 Win=19456 Len=0 TSval=1926241441 TSecr=8460305
18527	397.088170529	203.205.253.185	172.16.0.2	HTTP	428	HTTP/1.1 200 OK
18528	397.088994848	172.16.0.2	157.185.163.158	TCP	60	38096 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460329 TSecr=0 WS=64
18529	397.089024117	172.16.0.2	157.185.163.158	TCP	60	47066 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460330 TSecr=0 WS=64
18530	397.110934911	172.16.0.2	203.205.253.185	TCP	52	49142 → 80 [ACK] Seq=2814 Ack=753 Win=86144 Len=0 TSval=8460332 TSecr=1926241446
18531	397.461931086	172.16.0.2	203.205.253.185	TCP	52	[TCP Retransmission] 58884 → 80 [FIN, ACK] Seq=911 Ack=410 Win=85120 Len=0 TSval=8460369 TSecr=1926208820
18532	397.530926502	172.16.0.2	183.36.108.251	TCP	40	[TCP Retransmission] 41707 → 80 [FIN, ACK] Seq=163 Ack=168 Win=85120 Len=0
18533	398.078966157	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 38096 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460429 TSecr=0 WS=64
18534	398.078996606	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 47066 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460430 TSecr=0 WS=64
18535	398.098957149	172.16.0.2	157.185.163.158	TCP	60	39854 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460430 TSecr=0 WS=64
18536	398.099007080	172.16.0.2	157.185.163.158	TCP	60	53428 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460431 TSecr=0 WS=64
18537	398.691076083	172.16.0.2	203.205.253.185	TCP	40	[TCP Retransmission] 57842 → 80 [FIN, ACK] Seq=891 Ack=406 Win=85120 Len=0
18538	399.089061446	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 39854 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460530 TSecr=0 WS=64
18539	399.115914614	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 53428 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460531 TSecr=0 WS=64
18540	399.142956561	172.16.0.2	183.36.108.251	TCP	40	[TCP Retransmission] 41707 → 80 [FIN, ACK] Seq=163 Ack=168 Win=85120 Len=0
18541	400.720940513	172.16.0.2	203.205.253.185	TCP	52	[TCP Retransmission] 58884 → 80 [FIN, ACK] Seq=911 Ack=410 Win=85120 Len=0 TSval=8460694 TSecr=1926208820
18542	401.088961042	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 39854 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460730 TSecr=0 WS=64
18543	401.088990540	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 53428 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460731 TSecr=0 WS=64
18544	401.118921494	172.16.0.2	157.185.163.158	TCP	60	52895 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460732 TSecr=0 WS=64
18545	401.118958075	172.16.0.2	157.185.163.158	TCP	60	37277 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460732 TSecr=0 WS=64
18546	402.061011357	172.16.0.2	203.205.253.185	TCP	40	[TCP Retransmission] 57842 → 80 [FIN, ACK] Seq=891 Ack=406 Win=85120 Len=0
18547	402.109163321	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 37277 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460832 TSecr=0 WS=64
18548	402.109218324	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 52895 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8460832 TSecr=0 WS=64
18549	402.400941038	172.16.0.2	183.36.108.251	TCP	40	[TCP Retransmission] 41707 → 80 [FIN, ACK] Seq=163 Ack=168 Win=85120 Len=0
18550	403.282240915	183.36.108.251	172.16.0.2	TCP	40	[TCP Keep-Alive] 443 → 50283 [ACK] Seq=155 Ack=885 Win=17408 Len=0
18551	403.310943428	172.16.0.2	183.36.108.251	TCP	40	[TCP Keep-Alive ACK] 50283 → 443 [ACK] Seq=885 Ack=156 Win=84032 Len=0
18552	404.100989596	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 37277 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8461032 TSecr=0 WS=64
18553	404.101915408	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 52895 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8461032 TSecr=0 WS=64
18554	405.990968981	172.16.0.2	209.85.232.102	UDP	51	53612 → 443 Len=23
18555	406.038133336	209.85.232.102	172.16.0.2	UDP	48	443 → 53612 Len=20
18556	406.179923688	172.16.0.2	172.217.6.234	UDP	51	59737 → 443 Len=23
18557	406.214468348	172.217.6.234	172.16.0.2	UDP	48	443 → 59737 Len=20
18558	407.221031085	172.16.0.2	203.205.253.185	TCP	52	[TCP Retransmission] 58884 → 80 [FIN, ACK] Seq=911 Ack=410 Win=85120 Len=0 TSval=8461344 TSecr=1926208820
18559	408.118899215	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 37277 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8461433 TSecr=0 WS=64
18560	408.118927315	172.16.0.2	157.185.163.158	TCP	60	[TCP Retransmission] 52895 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8461433 TSecr=0 WS=64
18561	408.793927524	172.16.0.2	203.205.253.185	TCP	40	[TCP Retransmission] 57842 → 80 [FIN, ACK] Seq=891 Ack=406 Win=85120 Len=0
18562	408.900921884	172.16.0.2	183.36.108.251	TCP	40	[TCP Retransmission] 41707 → 80 [FIN, ACK] Seq=163 Ack=168 Win=85120 Len=0
18563	410.313127923	172.16.0.2	192.48.236.11	HTTP	799	GET /m/ad?MAGIC_NO=0&ac=0&av=1.6.4&bundle=com.toprange.locker&cn=20892&cppck=51118&ct=3&dn=samsung%2CGT-19515%2Cjfveltexx&dnt=0&exclude=d1f2c1f4d5d64…
18564	410.391294560	192.48.236.11	172.16.0.2	TCP	52	80 → 38653 [ACK] Seq=2151 Ack=1312 Win=31744 Len=0 TSval=3813097407 TSecr=8461648
18565	410.469055871	192.48.236.11	172.16.0.2	TCP	1440	80 → 38653 [ACK] Seq=2151 Ack=1312 Win=31744 Len=1388 TSval=3813097484 TSecr=8461648 [TCP segment of a reassembled PDU]
18566	410.469069869	192.48.236.11	172.16.0.2	HTTP	947	HTTP/1.1 200 OK (text/html)
18567	410.500947376	172.16.0.2	192.48.236.11	TCP	52	38653 → 80 [ACK] Seq=1312 Ack=3539 Win=92352 Len=0 TSval=8461671 TSecr=3813097484
18568	410.508891074	172.16.0.2	192.48.236.11	TCP	52	38653 → 80 [ACK] Seq=1312 Ack=4434 Win=95104 Len=0 TSval=8461671 TSecr=3813097484
18569	411.469869844	172.16.0.2	8.8.8.8	DNS	73	Standard query 0xac53 A googleads.g.doubleclick.net
18570	411.494203684	8.8.8.8	172.16.0.2	DNS	114	Standard query response 0xac53 A googleads.g.doubleclick.net CNAME pagead46.1.doubleclick.net A 172.217.10.98
18571	411.528914893	172.16.0.2	172.217.10.98	TCP	60	54530 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1400 SACK_PERM=1 TSval=8461773 TSecr=0 WS=64

Figure 5. 7 Wireshark message that show the number of retransmission messages in black which indicates a DoS situation.

6. Conclusions/ Future work

This project analyzed the LTE physical layer vulnerability to jamming regarding various channels and signals in the uplink and downlink. In order to perform jamming, the LTE testbed was built on the OpenAirInterface platform, using a USRP B210 as the eNodeB transceiver. After comparing possible jamming approaches and testing with a partial band jammer, the project implemented the frequency hopping jamming approach with a USRP N210. The result showed that LTE communication is vulnerable to partial band jamming, which could directly create the disconnection of the user equipment (UE). Furthermore, a frequency hopping jammer could affect the LTE system and result in packet retransmission, packet disruption, and even Denial of Service (DoS) when the power of the jamming signals was no less than 28 dB. An LTE cellular network is susceptible to jamming from relatively inexpensive equipment. The partial band jammer was effective in creating a DoS situation, however, it would be easily detected. The frequency hopping jammer would not be as easy to detect since it utilizes narrowband jamming.

6.1 Future Work

The focus of this project was to create an LTE testbed and effectively disrupt the communication link through a jamming attack. However, additional approaches can be conducted to analyze those scenarios. One possibility is to create a reactive jammer that is synchronized with the LTE system and only transmits when the target wireless system is active, or even more specifically, when crucial signals and channels in LTE transmission are active, making it more difficult to detect and defend. There is a theoretical synchronization algorithm for LTE systems that allows the jammer to locate the Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS) within the LTE downlink frame and decode the information contained in them [30]. In short, the algorithm utilizes the outputs

corresponding to the received signal vectors that do not correspond to PSS transmission to detect the location of the PSS signal within the received LTE downlink signal [30]. Once obtaining the correct cell identity, duplex mode, and CP mode from PSS and SSS, the jammer could decode the information for the whole LTE testbed downlink and thus perform more signal-targeted jamming for possibly better results.

Another possible direction for future work is to explore techniques for mitigating jamming attacks, especially, the frequency hopping jamming attack. Reference [31] introduces a method called *RF Frequency*, which uses frequency hopping strategy on the combined channels in the downlink, uplink or both, to avoid jamming attacks. This mechanism will be simply performed by RF switches or reconfiguration commands, which does not result in any system performance degradation [31]. In such an approach, if jamming occurs at constant frequencies with a period smaller than 10 ms, communication will not be disrupted due to the Hybrid Automatic Repeat Request (HARQ) message retransmission process [31].

References

[1] J. Donner, “Blurring Livelihoods and Lives: The Social Uses of Mobile Phones and Socioeconomic Development,” *Innov. Technol. Governance, Glob.*, vol. 4, pp. 91–101, 2009.

[2] C. Cox, *An introduction to LTE: LTE, LTE-advanced, SAE, VoLTE and 4G Mobile Communications*. John Wiley & Sons Ltd., 2014.

[3] “2019 CTIA Annual Survey Highlights,” *CTIA*. [Online]. Available: https://www.ctia.org/news/2019-annual-survey-highlights.

[4] T. Collins, R. Getz, D. Pu, and A. M. Wyglinski, *Software-Defined Radio for Engineers*. Artech House, 2018.

[5] R. Ghannam, F. Sharevski, and A. Chung, “User-targeted Denial-of-Service Attacks in LTE Mobile Networks,” *Int. Conf. Wirel. Mob. Comput. Netw. Commun.*, vol. 2018-Octob, pp. 1–8, 2018.

[6] S. Hilton, “Dyn Analysis Summary Of Friday October 21 Attack: Dyn Blog.” [Online]. Available: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[7] T. C. Clancy, M. Norton, and M. Lichtman, “Security challenges with LTE-advanced systems and military spectrum,” *Proc. - IEEE Mil. Commun. Conf. MILCOM*, pp. 375–381, 2013.

[8] “Home,” *OpenAirInterface*. Available: https://www.openairinterface.org/.

[9] “Software Radio Systems,” *SRS*. Available: https://www.softwareradiosystems.com/#.

[10] T. Pushpalata and S. Y. Chaudhari, “Need of Physical Layer Security in LTE : Analysis of Vulnerabilities in LTE Physical Layer,” pp. 1722–1727, 2017.

[11] L. Wang and A. M. Wyglinski, “A combined approach for distinguishing different types of jamming attacks against wireless networks,” *IEEE Pacific RIM Conf. Commun. Comput. Signal Process. - Proc.*, pp. 809–814, 2011.

[12] R. M. Rao, S. Ha, V. Marojevic, and J. H. Reed, “LTE PHY layer vulnerability analysis and testing using open-source SDR tools,” *Proc. - IEEE Mil. Commun. Conf. MILCOM*, vol. 2017-Octob, pp. 744–749, 2017.

[13] 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. Mag.*, vol. 54, no. 4, pp. 54–61, 2016.

[14] R. Krenz, “Jamming LTE signals,” *2015 IEEE Int. Black Sea Conf. Commun. Netw.*, pp. 72–76, 2015.

[15] K. Pelechrinis, M. Iliofotou, and S. V. Krishnamurthy, “Denial of service attacks in wireless networks: The case of jammers,” *IEEE Commun. Surv. Tutorials*, vol. 13, no. 2, pp. 245–257, 2011.

[16] A. M. Wyglinski, “Fundamentals of LTE-A: Upper Layer Functionality and Evolved Packet Core,” *TechOnline*, 2016. Available: https://www.youtube.com/watch?v=2RULSah9fXY.

[17] “About GNU Radio · GNU Radio,” *About GNU Radio*. Available: https://www.gnuradio.org/about/ .

[18] E. Yaacoub and Z. Dawy, “Joint uplink scheduling and interference mitigation in multicell LTE networks,” *IEEE Int. Conf. Commun.*, pp. 1–5, 2011.

[19] M. Lichtman, J. H. Reed, T. C. Clancy, and M. Norton, “Vulnerability of LTE to hostile interference,” *2013 IEEE Glob. Conf. Signal Inf. Process. Glob. 2013 - Proc.*, pp. 285–288, 2013.

[20] R. P. Jover, J. Lackey, and A. Raghavan, “Enhancing the security of LTE networks against jamming attacks,” *Eurasip J. Inf. Secur.*, vol. 2014, pp. 1–14, 2014.

[21] A. Hafsaoui, N. Nikaein, and L. Wang, “OpenAirInterface Traffic Generator (OTG): A realistic traffic generation tool for emerging application scenarios,” *Proc. 2012 IEEE 20th Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. MASCOTS 2012*, pp. 492–494, 2012.

[22] W. Stallings, *Dara and Computer Communications*, 10th ed. Pearson, 2013.

[23] R. Wang, Y. Peng, H. Qu, W. Li, H. Zhao, and B. Wu, “OpenAirInterface-An effective emulation platform for LTE and LTE-Advanced,” *Int. Conf. Ubiquitous Futur. Networks, ICUFN*, pp. 127–132, 2014.
[24] H. Anouar, C. Bonnet, D. Câmara, F. Fillali, and R. Knopp, “OpenAirInterface Simulation Platform,” *Time*, pp. 1–2.
[25] C. Tarver, “OAI Tutorial,” 2018. Available: https://chancetarver.com/oai/.
[26] A. Laurent, “All in one OpenAirInterfact,” *4G and 5G reference software*, 2019. [Online]. Available: https://open-cells.com/index.php/2019/09/22/all-in-one-openairinterface/.
[27] “SIM cards,” *Open Cells Project*. [Online]. Available: https://open-cells.com/index.php/sim-cards/.
[28] “WireShark.” Available: https://www.wireshark.org/.
[29] M. Haddad *et al.*, “A Survey on YouTube Streaming Service,” in *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, 2011.
[30] A. El-Keyi, O. Ureten, H. Yanikomeroglu, and T. Yensen, “LTE for Public Safety Networks: Synchronization in the Presence of Jamming,” *IEEE Access*, vol. 5, pp. 20800–20813, 2017.
[31] S. Barros, J. Bazzo, R. Takaki, D. Carrillo, and J. Seki, “LTE jamming mitigation based on frequency hopping strategies,” *2016 8th IEEE Latin-American Conf. Commun. LATINCOM 2016*, pp. 1–6, 2016.

Appendix A: Python Code for Frequency Hopping Jammer

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Top Block
# Generated: Tue Nov 19 13:53:59 2019
##################################################

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 analog
from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from grc_gnuradio import blks2 as grc_blks2
from optparse import OptionParser
import sys
import threading
import time
import random
from gnuradio import qtgui


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

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

self.restoreGeometry(self.settings.value("geometry").toByteArray())


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

        ##################################################
        # Blocks
        ##################################################
        self.probe = blocks.probe_signal_f()
        self._center_freq_tool_bar = Qt.QToolBar(self)
        self._center_freq_tool_bar.addWidget(Qt.QLabel("center_freq"+":
"))
        self._center_freq_line_edit = Qt.QLineEdit(str(self.center_freq))
        self._center_freq_tool_bar.addWidget(self._center_freq_line_edit)
        self._center_freq_line_edit.returnPressed.connect(
              lambda:
self.set_center_freq(int(str(self._center_freq_line_edit.text().toAscii())
)))
        self.top_grid_layout.addWidget(self._center_freq_tool_bar)


        self.uhd_usrp_sink_0 = uhd.usrp_sink(
              ",".join(("addr=192.168.50.2", "")),
              uhd.stream_args(
                    cpu_format="fc32",
                    channels=range(1),
              ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_center_freq(center_freq, 0)
        self.uhd_usrp_sink_0.set_gain(28, 0) #gain of Jammer
        self.digital_ofdm_mod_0 = grc_blks2.packet_mod_c(digital.ofdm_mod(
                    options=grc_blks2.options(
                          modulation="qpsk",
                          fft_length=4096,
                          occupied_tones=60,#Frequency BW
                          cp_length=128,
```

```
                                pad_for_usrp=True,
                                log=None,
                                verbose=None,
                            ),
                        ),
                        payload_length=16,
                    )
                    self.blocks_throttle_0 = blocks.throttle(gr.sizeof_float*1,
samp_rate,True)
                    self.analog_sig_source_x_0 = analog.sig_source_c(samp_rate,
analog.GR_COS_WAVE, 1000, 1, 0)
                    self.Clock = analog.sig_source_f(samp_rate, analog.GR_SQR_WAVE,
20, 1, 0)

                def _variable_function_probe_0_probe():
                    while True:

                        val = self.probe.level()
                        if val ==0:

                def self.set_center_freq(random.randrange(2.6775e9,2.6825e9,1e3))
                        try:
                            self.set_variable_function_probe_0(val)
                        except AttributeError:
                            pass
                        time.sleep(1.0 / (40))
                    _variable_function_probe_0_thread =
threading.Thread(target=_variable_function_probe_0_probe)
                    _variable_function_probe_0_thread.daemon = True
                    _variable_function_probe_0_thread.start()


                    ##################################################
                    # Connections
                    ##################################################
                    self.connect((self.Clock, 0), (self.blocks_throttle_0, 0))
                    self.connect((self.analog_sig_source_x_0, 0),
(self.digital_ofdm_mod_0, 0))
                    self.connect((self.blocks_throttle_0, 0), (self.probe, 0))
                    self.connect((self.digital_ofdm_mod_0, 0), (self.uhd_usrp_sink_0,
0))

                def closeEvent(self, event):
                    self.settings = Qt.QSettings("GNU Radio", "top_block")
                    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_sink_0.set_samp_rate(self.samp_rate)
        self.blocks_throttle_0.set_sample_rate(self.samp_rate)
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
        self.Clock.set_sampling_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
        Qt.QMetaObject.invokeMethod(self._center_freq_line_edit,
"setText", Qt.Q_ARG("QString", str(self.center_freq)))
        self.uhd_usrp_sink_0.set_center_freq(self.center_freq, 0)


def main(top_block_cls=top_block, 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()
```

Appendix B: Python Code for Wideband Jammer

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Wd Top Block
# Generated: Wed Dec  4 15:37:30 2019
##################################################

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 analog
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import qtgui
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from grc_gnuradio import blks2 as grc_blks2
from optparse import OptionParser
import sip
import sys
import time
from gnuradio import qtgui


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

    def __init__(self):
        gr.top_block.__init__(self, "Wd Top Block")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Wd Top Block")
        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", "wd_top_block")

self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.samp_rate = samp_rate = 20e6

        ##################################################
        # Blocks
        ##################################################
        self.uhd_usrp_sink_0 = uhd.usrp_sink(
              ",".join(("addr=192.168.50.2", "")),
              uhd.stream_args(
                     cpu_format="fc32",
                     channels=range(1),
              ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_center_freq(2.56e9, 0)
        self.uhd_usrp_sink_0.set_gain(28, 0)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
              1024, #size
              firdes.WIN_RECTANGULAR, #wintype
              0, #fc
              samp_rate, #bw
              "Partial Band Jammer", #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(0.05)
        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.digital_ofdm_mod_0 = grc_blks2.packet_mod_c(digital.ofdm_mod(
                        options=grc_blks2.options(
                                modulation="qpsk",
                                fft_length=4096,
                                occupied_tones=90,
                                cp_length=128,
                                pad_for_usrp=True,
                                log=None,
                                verbose=None,
                        ),
                ),
                payload_length=16,
        )
        self.analog_sig_source_x_0 = analog.sig_source_c(samp_rate,
analog.GR_COS_WAVE, 1000, 1, 0)



        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_sig_source_x_0, 0),
(self.digital_ofdm_mod_0, 0))
        self.connect((self.digital_ofdm_mod_0, 0),
(self.qtgui_freq_sink_x_0, 0))
        self.connect((self.digital_ofdm_mod_0, 0), (self.uhd_usrp_sink_0,
0))

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

```
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.samp_rate)
        self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)


def main(top_block_cls=wd_top_block, 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()
```