University of Stavanger

Faculty of Science and Technology

BACHELOR'S THESIS

Study program/ Specialization: Bachelor in Data	Spring semester, 2023 Open access
Writers: Nils Petter Håberg Sebastian Gustavsen	Håberg (Writer's signature) [illegible] (Writer's signature)
Faculty supervisor: Gianfranco Nencioni External supervisor(s):	
Thesis title: Pre-study on Wireless Communication lab: medium access control	
Credits (ECTS):20	
Key words: MAC schemes, wireless communication, tools, exercises, simulations, and learning	Pages: 70 + enclosure: Stavanger,14/05/2023 Date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Pre-study on Wireless Communication lab: medium access control

Bachelor's Thesis in Computer Science
by

Nils Petter Håberg and
Sebastian Gustavsen

Internal Supervisor

Gianfranco Nencioni

May 14, 2023

Abstract

This thesis aims to create a series of exercises to help students understand wireless communication MAC schemes better. We did this by finding the right tool to help us create good exercises. The right tools for the exercises were found by reading through survey papers and documentation of various tools and comparing them to each other through what we found from the survey papers. In addition, we created exercises to showcase different MAC schemes, each constructed differently to provide a unique learning experience. The exercises are about CSMA/CD, CSMA/CA, CDMA/FHSS, CDMA/DSSS, and OFDM. The CSMA/CD and CDMA/CA exercises with the tool NS-3 are great exercises to see the two MAC schemes through a simulation and how they work. CDMA/FHSS and CDMA/DSSS were two different exercises designed to be practical, that the student needed to solve the exercises by coding in MATLAB. A more practical exercise that gives the student a deeper understanding of the core functionalities of the MAC schemes. The last exercise we designed was a walk-through about the different phases of OFDM. The main goal was to give the student an understanding of how the OFDM MAC scheme works by giving them a visual representation with plots of each phase during data transmission.

Acknowledgements

We would like to thank our supervisor for his fantastic enthusiasm and help with writing this thesis.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Wireless communication has transformed the way we engage with each other. In the past, we used physical connections, such as cables to communicate over large distances, called the Physical layer in the OSI model. Now the communication is done wirelessly through various MAC schemes, part of the Datalink layer. The MAC scheme allows everyone to communicate with another person on the other side.

Throughout history, there have been several inventions of various MAC schemes. Logically the schemes have improved over time. Different schemes are designated for different situations. However, they all have one thing in common. They are a set of rules that dictate how to transmit and receive data through wireless communication. Furthermore, specific MAC schemes are more developed to optimize the efficiency and reliability of transmitted data.

The MAC schemes can be challenging to comprehend. Fortunately, practice is the best way to learn the various MAC schemes. Therefore, this thesis aims to help the student better understand some MAC schemes through exercises we will design. A good understanding of the older and newer MAC schemes is a necessary part of developing better MAC schemes in the future.

1.2 Objectives

The end goal is to give people an understanding of the MAC schemes through exercises, but first, we need to find the tools that help us with that. The tools we are looking for are software that can perform simulations and modifications on the MAC schemes. Thus we will find these by reading through survey papers on the tools. Then we will compare them and find the best tools for our exercises.

The exercises will show the characteristic and functionalities of the schemes. The main goal is to give the person that does the exercise a firm understanding of how a scheme functions. When designing the exercises, we aim to make them different from each other. For instance, one will focus on gaining understanding through coding. Another one will be more of a walk-through of the scheme's work, with visual representations such as graphs. In addition, another one is where the student can change some parameters and see how well the scheme will perform when simulating it. We also want the student to explain why what they see when doing the exercises. Therefore, the student can only do these exercises with some background knowledge of the scheme. Therefore we also aim to give the student comprehensive knowledge about the various schemes. That is a necessity to get the knowledge needed for the exercises. We will also find the best suitable tools to construct these exercises. The tools used in the exercises must be easy to understand for the student's learning.

Also, our focus is not to have a real simulation of the scheme, but the simulation will act just enough as a realistic simulation. That student can gain a good understanding of the characteristic. Additionally, the complications would not be too severe when implementing the exercise because the complications might prevent the student from extracting knowledge. Because if the code is complex, it might result in a lost opportunity for the code to be flexible. We want the student to be able to change things and see how the simulation will fare with the changes. Our conclusion will reflect how good the exercises were in characteristics and the level of depth.

1.3 Approach and Contributions

We will construct four different exercises. Each with different approaches and focuses since their characteristics are different, including that we have chosen them because of our prior experience in practical learning. We mentioned above how we will construct the approach for the various exercises. Before all that, we need to learn about the schemes and how they operate. Then we will be able to understand the characteristics and effectively design the exercises that highlight the said characteristics. Our next task will be to do this using tools. The tools, as mentioned, must be able to implement the exercises we design. Therefore we must research through documentation and survey papers which existing tools would fit the exercises the best.

1.4 Outline

In Chapter 2, the background will explain the schemes and give a base knowledge that will be used further in the thesis. The background is a student's minimal knowledge before approaching the exercises. We will use the knowledge we gained in Chapter 2 to find the best tool, which we will cover in Chapter 3.

Chapter 3 will be about the state of the art. First, we go research related work where we go through some survey papers that reviews tools that can simulate the various MAC schemes. After that, we go through each tool more straightforwardly by reading the documentation for the different tools. Then we compare them based on what we have found in the survey papers and the documentation. In the end, we will put the tools side by side for comparison to each other and find the best tool for the exercises. Thus, at the end of the comparison, the chosen tools will reflect on the exercises we design in Chapter 4.

In Chapter 4, we explain the whole approach to the exercises. We will begin by presenting which exercises we will implement using which tool. After that, we will detail the motivation, meaning why we structure the exercise a certain way and what we aim to explain to the student doing the exercises. The description, where we dive deeper into our vision of the exercises with more detail, as the parameters we intend to use and the assumptions that will be used for the various implementation. At last, we explain how we implemented the exercises, such as how the code works. We will also include the parameters used and how the student's interface will look. Consequently, people can then feel free to set up the exercises themselves. The results of these exercises will be presented and evaluated in Chapter 5.

The evaluation of the exercises will take place in Chapter 5. That will consist of how well the exercises performed in comparison to our primary goal of the exercises. Through the results of the exercises, we will get a good indicator of the main goals achieved. The results will also represent the proper implementation of a student's answer. Furthermore, our analysis involved anticipating the outcomes, identifying areas for improvement, and providing an explanation in case the exercises provided the student with a positive learning experience.

How the thesis went will be delved into Chapter 6, where we explain what went well and what did not. How the choice of tools affected our exercises with their functions and abilities. In the end, we will suggest future work that could potentially improve the exercises that will give a better learning experience for the student.

Chapter 2

Background

Before creating the exercises for the various MAC schemes, we first need to explain the schemes we will use for our exercises. Each scheme has unique characteristics and functionalities, which we will shine upon when we design and implement the exercises.

2.0.1 CSMA

The scheme, Carrier Sense Multiple Access (CSMA), aims to reduce the likelihood of collisions between devices when transmitting data. Collisions mean that signals interfere with each other and corrupt the data. CSMA works by having one device that will listen to the channel before transmitting, ensuring that no other device is transmitting data through the same channel. The listening process means the network device will check if other signals are present in the channel. Developers have created different variations of CSMA to optimize efficiency even more. We describe the variations in the next sections [1].

2.0.2 CSMA/CD

Carrier Sense Multiple Access/Collision Detection is a variation of the CSMA to reduce the chance of a collision even more. The device listens if the channel is idle and also monitors the channel while transmitting data to detect if any collisions occur. If a device detects a collision, it will stop the transmission and send a jam signal to inform the other devices that a collision has occurred. Then the device will wait a random time interval before transmitting the data again. The process called "Exponential Backoff" time, where the random time interval is chosen in the interval of 0 and $2^n - 1$. Where n is the number of collisions that have happened. The time can only be chosen by the equation $T = 2_t$ where the small t is the propagation delay of the network, meaning the time it takes for the data to reach the destination [1].

2.0.3 Hidden terminal problem

A characteristic problem with CMSA/CD is the hidden terminal problem. When we have three nodes in a way where node 1 and node 3 can send to node 2, and nodes 1 and 3 are out of range of each other, see Figure 2.1. The problem occurs if node 1 wants to transmit data to node 2 and node 3 also wants to transmit data to node 2. They both will listen to the channel to check if it is idle. However, since node 1 and node 3 are out of range of each other, they will perceive the channel as idle. Thus, this will result in collisions. The "Exponential Backoff" timer will help a little, but the transmission will be random and without sense. Collisions will again occur as a result [1]. Fortunately, the solution is another CSMA variant called CSMA/CA. CSMA/CA uses a handshake method, as explained in the next section.

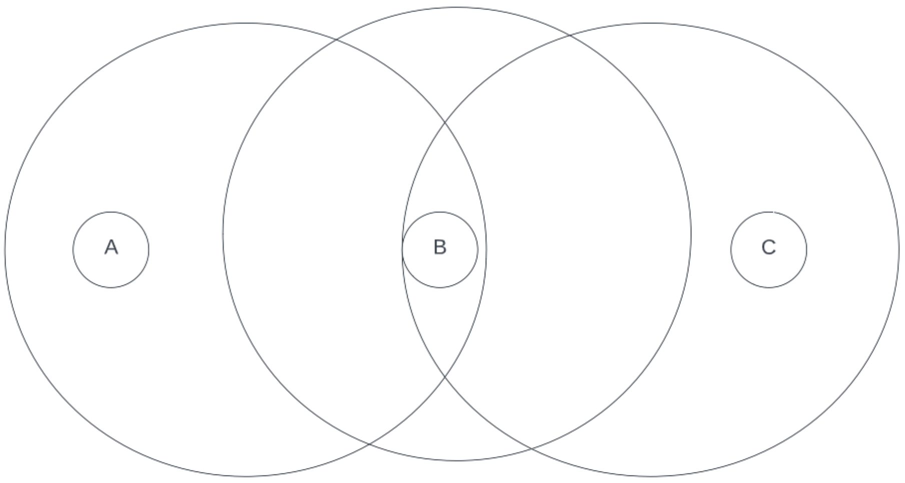


Figure 2.1: A figure that shows the hidden terminal problem. Notice that nodes A and c are out of range of each other.

CSMA/CA

Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) is another CSMA variation that reduces collisions when devices transmit data over the same channel. The device will also listen to the channel to see if it is idle. When the channel is idle, the sender and receiver use a "virtual carrier sensing" mechanism to avoid collisions [1].

Virtual carrier sensing works by the sender will send a Request to Send (RTS) message, and the receiver will respond to the sender with a Clear to Send (CTS) message, indicating that it is ready to receive the data. All this will preserve the channel and prevent other devices from transmitting data during the transmission [1].

When the sender gets the CTS message, it waits for a short time before sending the data, known as the Short Interframe Space (SIFS). The reason for having this time is to ensure that the devices have the time to receive and process the CTS message [2].

After the SIFS time, the transmission of the data will begin and continue until the transmission is complete. The receiver ensures the sender that the transmission was successful by sending back an acknowledgment (ACK) message to the sender [2].

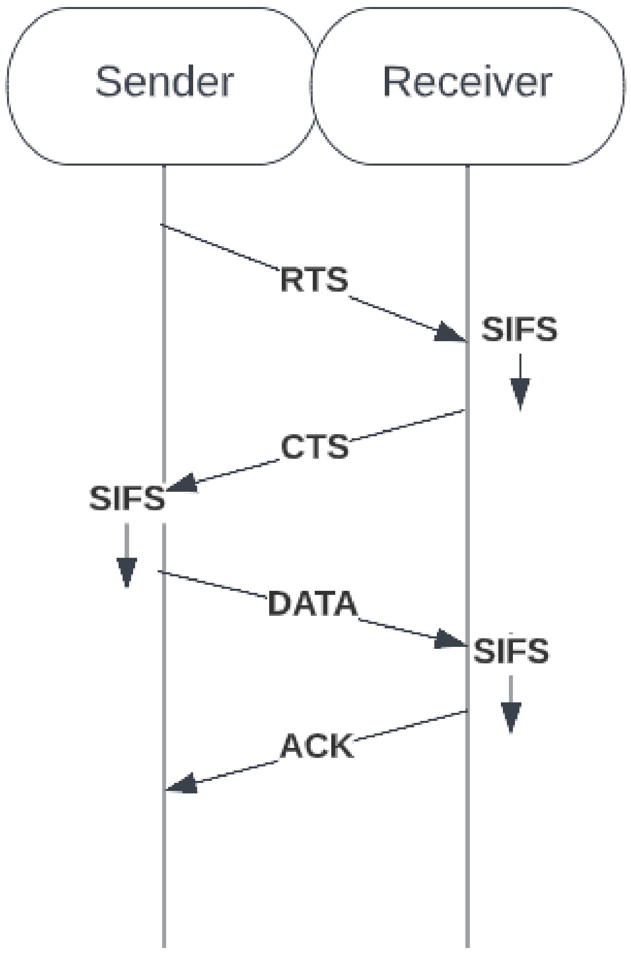


Figure 2.2: A figure that shows how the handshake method of CSMA/CA works.

2.0.4 CDMA

Code Multiple Division Access is a scheme that allows multiple users to send data through a single frequency band. It is possible by assigning each user a unique code and using that code to modulate their signal onto the frequency band. Thus the users can transmit data simultaneously. Then, to distinguish the users, the receivers selectively "despread" a particular signal, which separates it from the other signals. The receiver performs despreading by multiplying the received signal with the sender's spreading code to obtain the original data. The CDMA has two types of spreading that we will explain further below, which are FHSS and DSSS [1].

CDMA/FHSS

The Frequency Hopping Spread Spectrum (FHSS) is a variant of CDMA and uses the spread spectrum technique to send data over a channel. Sending data segments over different carrier frequencies within the same frequency band is the way of accomplishing this. A carrier frequency is a wave that carries the data through the channel [3].

When the sender transmits data, it will hop between the different frequencies. An algorithm performs the hop and is designed to appear random. Thus the sender will send one segment of data, then hop to another frequency, and then send another segment [3].

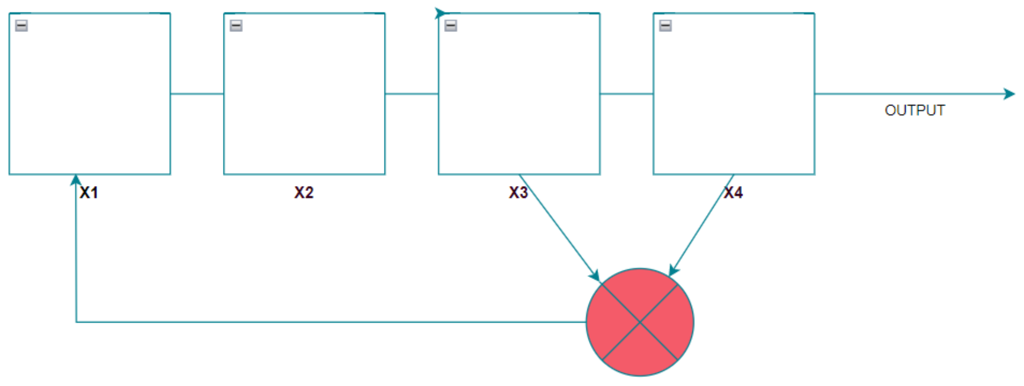
The receiver must synchronize with the sender. Therefore the receiver must use the same algorithm as the sender to know which frequency the sender uses. Thus the sender and receiver hop between different frequencies simultaneously through an algorithm [3].

2.0.5 CDMA/DSSS

Direct-Sequence Spread Spectrum is another way of implementing CDMA. Each user is assigned a PN code. The code is generated using a mathematical algorithm that seems random but is not. The length of the PN code determines how many users can be involved in the same frequency band and allows multiple users to share the same frequency band simultaneously. Before transmitting the data, the sender will multiply the PN code with the original data, called spreading. Thus each user's data is separated by the PN code. When the receiver gets the signal, it will "despread" it to restore the original data with the PN code. The PN code is the same for the sender and receiver [1].

The PN code is a sequence of binary digits generated by linear feedback shift registers (LFSR). Constructing these LFSRs involves using an elementary binary logic circuit. The LFSR structure depends on how many bits the seed is. For instance, if we have 4 bits, the polynomial will look like this $x^4 + x^3 + 1$. Then the flip flops in the LFSR will be structured like Figure 2.3 [1].

Figure 2.3: LFSR register



Notice that the two flip-flops, X3 and X4, go to an XOR port, represented as a red circle with an X in the image above. Also, the two flip-flops that change the first flip-flop X1 are the two X variables that have the power of their position in the polynomial. This XOR operation comes after a shift, which helps generate the sequence with its outputs. The output of the LFSR is a binary sequence called Maximum Length Sequence (MSL). MSL has to have a period of $2^n - 1$ where "n" of bits were in the shift register. The MSL has several properties that define it. Firstly, the difference between 1 and -1 (or 1 and 0) amounts is no bigger than one, and these are also called chips. Secondly, it has a run property where groups of -1 or 1 are in consecutive order until the other number appears. Exactly 50% of the chips must be groups of 1s and other - 1s. Lastly, MSL has an autocorrelation property. Auto-correlation is used on the receiver side when disguising the different signals. For instance, the incoming signal is compared with a shifted version of itself. If the number of chips that differ from the shifted does not exceed a threshold, then they are autocorrelated. That implies that the receiver has received the correct Data [1].

2.0.6 OFDM

Orthogonal Frequency Division Multiplexing (OFDM) is a scheme that transmits data by encoding data on multiple subcarriers. The sender will modify the digital sequence to a signal with a modulation technique. This method makes it possible to convert the data stream from serial to parallel and place them on different carrier frequencies. There are various modulation methods. However, the main idea is to encode digital data onto the carrier frequencies for the different subcarriers. The different subcarrier frequencies are orthogonal to each other, which is necessary because of multiple subcarriers on the same frequency carrier. Orthogonally is done by using Inverse Fast Fourier Transformation (IFFT) to map the modulated data into the different orthogonal subcarrier [4]. Where one subcarrier signal peaks in power, the other subcarriers will have a power equal to zero. That will cause very little interference between the adjacent subcarriers. To check whether the two carrier signals are orthogonal, we multiply their average bit time and get zero [1].

After the IFFT phase, a Cycle Prefix (CP) must be added before converting the parallel data to serial again and transmitting it. The reason for CP is to stop the interference between different signals, called Inter-Symbol Interference (ISI) [5]. ISI is when several echoes of the transmissions interfere with each other. In other words, the environment reflects our signal when we transmit it because of multiple delayed paths. Thus, if a sender's delayed signal is more significant than the receiver's data window, the transmitted data will overflow onto the next data window [6]. Thus, the overflow of the previous one will interfere with the next incoming signal. Therefore, we copy the last signal data signal and place it in front of the data signal to stop ISI. Thus if the signal is affected by a delayed signal, only the copies of the data will be corrupted and therefore the original signal will still be readable by the receiver. When the signal is received, the receiver converts the signal to parallel streams and removes the CP. Subsequently, recovering the original data involves using Fast Fourier Transformation (FFT) and demodulation. At last, it will convert the data from parallel to serial [4].

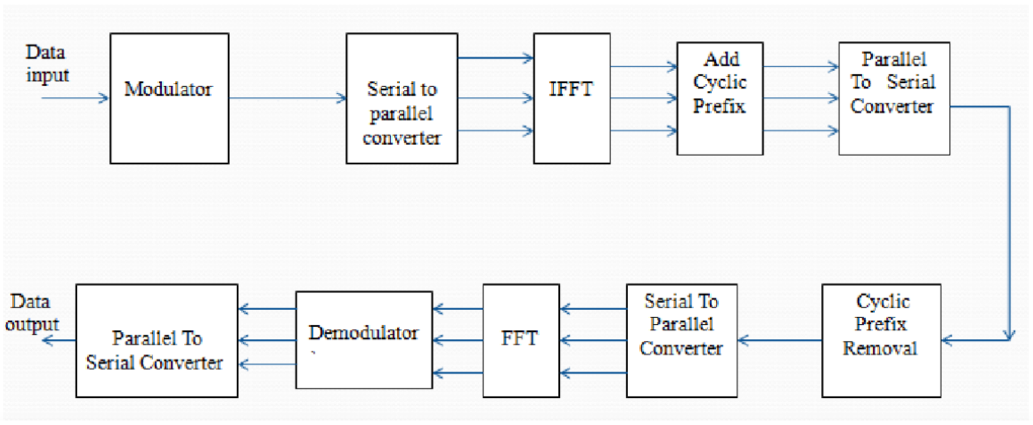

Data input
Modulator
Serial to parallel converter
IFFT
Add Cyclic Prefix
Parallel To Serial Converter
Cyclic Prefix Removal
Serial To Parallel Converter
FFT
Demodulator
Parallel To Serial Converter
Data output

Figure 2.4: A Figure that shows the OFDM scheme, taken from article [7]. It shows how the OFDM scheme works.

Chapter 3

State of the Art

3.1 Introduction

This chapter will first review some related works through survey papers. After that, the next section will be a detailed description of each tool we found through survey papers and our research, which will define how we design our exercises. At last, a side-by-side comparison of the tools to find the optimal tools for our exercises.

3.2 Related works

In [8] describes the performance of simulation tools for MAC schemes, like Network Simulator Version 2 (NS-2). According to the article, it has a great library of network and scheme objects. In addition, a C++ compiler is used to achieve great execution times and efficient simulations. In combination, it is easy to modify and customize the code, which will be beneficial for our design of the exercises. A similar survey paper, [9] mentions that NS-2 also includes a tracer file. That tracks the information about events during the simulation, making it possible to analyze the data and see the performance in detail after the simulation. This could be exploited for our exercises since the person solving them needs information on key values during the simulation. Moreover, it mentioned some MAC schemes integrated into the tool, such as CSMA/CA and CSMA/CD. That will make the design of the exercises easier since there is a foundation to start on.

The disadvantage of NS-2, according to the survey paper [10] is that it lacks customization for different MAC schemes. Further strengthened because [11] also states the same. Including that NS-2 has a long learning curve. Advanced programming skills are needed to perform more meaningful simulations.

On the other hand, the survey paper [9] mentioned NS-3 as a newer version of it. The main difference between NS-2 and NS-3 for MAC schemes is that NS-3 can handle multiple interfaces of nodes correctly. Hence, it makes it possible for us to design more complex network systems. In addition, NS-3 offers the possibility of making a more detailed MAC scheme.

In [8], they mention that MATLAB is a matrix laboratory used for simple programming. It is simple to program and debugs any errors that might occur, which can further be exploited in our exercises.

In [9] mentions a tool named OMNeT++ that focuses on the OSI model's lower level, like the Data Link Layer, which contains the MAC schemes. Article [12] writes similarly, which describes the OMNeT++ support for MAC schemes as an advantage compared to other tools.

3.3 Tools

3.3.1 NS-2

NS-2 is a discrete event-driven network simulation tool driven by the user's actions. It provides excellent support for simulating different schemes in the wireless network. It is a viral simulation tool for academic environments [13].

3.3.2 NS-3

NS-3 is an open-source discrete event network simulation intended for education and research. It can simulate network systems and schemes in Layers 1 and 2. The programming languages it is written in are C++ and Python. NS-3 is powerful, and the software infrastructure is designed to encourage the development of realistic models. Hence, NS-3 allows it to be a real-time simulation that reflects the real world. NS-3's modular architecture makes integrating other libraries and tools easier. Moreover, NS-3 generates trace files that enable the evaluation of the performance and behavior of the schemes [14].

3.3.3 GlomoSim

GlomoSim Network Simulator is an open-source program that can simulate various networks, including wireless networks. It has a library that supports CSMA. Additionally, it is also very extensible because users can add new schemes and modules to its library [15].

3.3.4 OPNET

OPNET is a network simulation tool that allows the modeling, evaluation, and visualization of communication networks, systems, and services. It includes a simulation tool that supports various wireless communication schemes. Through this simulation tool, it is possible to evaluate the performance of the different schemes in different situations. This includes a graphical user interface that can simulate various behaviors of the nodes and access points in a network and evaluate different performance metrics, such as latency, throughput, and jitter [16].

3.3.5 OMNeT++

OMNeT++ is a discrete event simulation framework and component-based C++ simulation library. It can model, simulate and analyze different networks and network schemes. The simulation will build components that will exchange messages. Each component has to be programmed in C++ using the simulator and class library. The simulator's result files are text-based but can be processed with R, MATLAB, or other tools [17].

3.3.6 Scilab

Scilab is an open-source numerical computing software platform. A feature they offer is a tool that can optimize matrix operations and make it easy to plot and visualize data. Through this feature, it is ideal for solving linear algebra and numerical optimization problems, which are critical problems concerning wireless communication. Another key feature is that Scilab offers interoperability with other programming languages. This makes integrating additional libraries and tools to simulate wireless communication easier. The tools provide ideal functionalities like signal generation, channel modeling, spreading code generation, and performance metrics. Furthermore, these tools support different schemes, such as CDMA and OFDM [18].

3.3.7 MATLAB

Matrix Laboratory (MATLAB) is also a numerical computing software platform. It offers tools and functions for different computing applications. A feature they provide is its matrix compilation capabilities. This makes it ideal for linear algebra and numerical optimization problems. Another key feature is that the high-level programming language allows users to construct custom scripts and functions. Additionally, MATLAB includes interoperability with other programming languages, such as C/C++ and Python. In addition, it provides optimal visualization through 2D and 3D plotting functions, heat maps, and histograms. Regarding wireless communication, MATLAB has various tools to simulate and analyze wireless communication systems [19].

3.3.8 QualNet

QualNet is a simulation software that evaluates wireless communication performance and can perform high-fidelity simulations of significant network typologies faster than real-time speed. That includes simulating communication networks with different configurations and conditions. All this makes it possible to design, test, and analyze the different communication schemes. Also, QualNet offers Human-in-the-Loop (HITL), which is an interface that supports dynamic interactions. That allows for deactivating or activating the different nodes and manipulating the traffic rate. Similarly to the other tools above, QualNet can compare different metrics to test the efficiency of the network topology, such as packet loss, throughput, and delays. Once simulated, the system can plot the metrics in a graph and store the results in a database for further use [20].

3.3.9 Java-DSP

Java-DSP is an open-source library in the Java programming language that offers digital signal processing (DSP) operations. Used to represent, analyze, and manipulate signals. Java-DSP provides high-level DSP operations, such as Fourier transmission, windowing, filtering, and frequency analyses. A key benefit of Java-DSP is that it provides easy-to-use APIs. Hence, it can construct low-level projects or complex, high-performance processes [21].

3.3.10 GSN3

GSN3 is free, open-source software. Which can configure, emulate, and test virtual and real networks. It can create a small topology with few devices or a more significant network with many devices. GNS3 consists of two components, the GNS3-all-in-one software (GUI) and the GNS3 virtual machine (VM). The GUI is the user interface that comes with everything. Users can drag components into an environment and create networks from them. The process can run on the local GNS3 server or a person's PC.

The developers recommend the local GNS3 virtual machine, achieved using virtualization software. The third and last option is the remote GNS3 virtual machine, which runs on a remote server or the cloud [22].

3.3.11 Comparison of tools

When designing exercises, we aim to showcase some key characteristics and functionalities of the different schemes. We will compare the tools to the exercises we want to implement and then compare the tools to each other. One of our approaches to the exercises is to make a student set up a simulation of the different schemes. Therefore we need a user-friendly tool that can convincingly simulate a scheme. In addition, some schemes are much more effective than others. Hence we need a tool that can show the measures after the simulation. For instance, how many collisions happened, delays, throughput, etc. Also, a way of changing the different parameters, such as bit rate. That allows the student to see the efficiency of the scheme in different scenarios. For more complex schemes like CDMA/DSSS that use spreading code, we can design exercises where they, for example, construct or analyze the spreading code. Therefore we need a tool that can manipulate the spreading code.

NS-3 makes it possible to study the network performance and the schemes in a controlled and sizeable manner. NS-3 provides built-in MAC schemes, which will make it easier to construct the exercise because we do not have to build them from scratch. Another advantage of NS-3 is that it includes trace files that can evaluate performance metrics when comparing different schemes. However, NS-3 has a steep learning curve for people unfamiliar with Linux and complicated terminal commands. Additionally, we noticed that there were not many tutorials and documentation about it online, and the manual from their developers was difficult to read.

MATLAB has a lot of tutorials online, which is needed to be able to do the different exercises. We have also noticed that it is not free and becomes quite expensive once you add all those sub-tools. Fortunately, the university provides a license. The disadvantage of MATLAB is that it does not provide virtual simulation. When simulating the different schemes, it is impossible to see the actual data packet go from node to node. MATLAB provides many logging techniques essential when evaluating the various performance measures and a graphical representation through plots. That creates an equilibrium for the lack of visualization.

We will not choose NS-2 at all since NS-3 is a newer version. It does not have as many customizations of MAC schemes as NS-3.

GNS3 provides a virtual interface that is very user-friendly. It will make the design of the exercises easier because we can create the exercises by dragging components into the environment. However, we do not think this would be a suitable tool since it does not go into detail for each MAC scheme. Instead, it simulates the network from a very general perspective. That makes it difficult to modify the parameters and manipulate the different aspects of the schemes. We want the tool to be easy to set up and capable of simulating without any problems. However, it provides a virtual machine that runs on a cloud server, not the local PC, which will be more complicated to utilize.

OMNeT++ can be used to create networks by programming each module. With this, we could configure the size of the network and the schemes used. The results of each simulation are text-based, but viewing the different measures will still be valuable. We can plot the results with other tools to get a prettier overview if needed. However, each component needs to be coded individually in C++. Resulting in the need for more work setting up the simulation. On the other hand, the need for coding the individual components provide more flexibility.

Java-DSP can perform complex digital signal processes like filtering and signal transformation, such as fast Fourier transformation and peak detection. In addition, java-DSP provides easy-to-use APIs for performing complex operations on the signal. As mentioned in the tool section, Java-Dsp is written in Java, one of the most used languages. Thus the probability of the student understanding the implementation of the simulation is higher. It is easier to do exercises where the student can change a piece of code and then evaluate what will happen.

It seems as if Opnet has some MAC schemes already integrated into the system, making implementing the exercises easier. Moreover, it can model and visualize the network and provide reasonable measures, which would be needed when we try to show the characteristics of MAC schemes. Unfortunately, it is not free, which is a factor that determines us not to use it.

Scilab looks very identical to MATLAB, but it is free. It can be used to create different MAC schemes. However, there is no good documentation of it.

Glomsim can be used for exercises that revolve around CSMA since it already has it Incorporated into its supportive library. However, it does not have the other MAC schemes in its library. For those, we would have to make ourselves.

QualNet can emulate different networks with different configurations, which would be good for doing exercises for the different MAC schemes. Additionally, it gives us a good evaluation of measures on network simulations, which is necessary for highlighting the characteristics of MAC schemes. The results can also be plotted in a graph for a better understanding. However, there is very little information about it online and therefore we will drop this tool.

In the sections above, we reflected on how the different tools can potentially be used in our exercise. In this next part, we will discuss what tools we will be using and why. There are three main features the tools must have to be able to show the characteristic of the MAC schemes most easily and effectively. The first one is a simulation tool that can perform a virtual representation of the exercise. The ones capable of this are OMNeT++, Glomosim, and GNS3. As for OMNeT++, it will be difficult and time-consuming when setting up the exercises because we need to code every node in C++. Also, C++ is considered one of the most difficult programming languages to learn. Furthermore, a good thing about Glomosim is that it offers integrated functions that specifically focus on MAC schemes. Selecting a node and choosing which MAC layer scheme the nodes should operate on is possible. Thus comparing this to OMNeT++, it is easier to set up the exercises to showcase the characteristics. However, Glomosim is an old tool, and it is difficult to find documentation. The GNS3 simulates the network with a general perspective, but it lacks the highlight of the characteristic of MAC schemes. Overall, these tools are either too old or do not have the capabilities to show the characteristics of MAC schemes, and that is why we will not choose any of them.

The second key feature the tool must have is the ability to show good performance measurement during and after simulation. NS-3 would be the best option for this. However, it needs a C++ compiler and is difficult to use. On the other hand, it is worth the struggle for the benefit of metric measures in a simulated environment. Therefore we will choose Ns-3 as a simulation tool that can capture the performance measurements.

Another key feature we are looking for when choosing the tools is flexibility when it comes to setting up the simulation. For instance, when we want to showcase the characteristics of CDMA/DSSS. We want to be able to change or make a different spreading code. Scilab, MATLAB, and Java-DSP all provide this. However, MATLAB is much more used. Therefore it is easier to use because it is much more documented, and the chance that someone is already familiar with MATLAB is bigger than the other tools. The only bad thing about MATLAB compared to Scilab and java-DSP is that it costs money, and the other two tools are free. Luckily, The university provides a license for MATLAB, which will be the tool we will choose for this section. In addition, MATLAB also includes a lot of functions for wireless communication, like FFT and modulation, and can provide plots as a representation of a simulation. Therefore we have chosen to use MATLAB for the simulation of a scheme.

Table 3.1: A table that shows the advantages and disadvantages sides of each tool

Tool	**Advantages**	**Disadvantages**
NS3	Open-source and has CSMA scheme implemented in the library, and the simulations are both good and highly customizable. Provides customization for each node to fit the circumstances. Can perform measures during simulations to see the effect of changes. Good tutorials on the main website.	It has a long learning curve, and it uses C++. The documentation of functions could be better in explanation.
NS2	Open-source and has CSMA scheme implemented in the library. Can perform measures during simulations to see the effect of changes.	It has a long learning curve, and it uses C++. It is also old, and NS3 is a newer version.
MATLAB	It has a lot of functions and tools that can simulate MAC schemes and capture measures of simulations. Many tutorials about it online. Easy to manipulate matrices.	It can be overwhelming with information, and it is not free.

Continued on next page

Table 3.1 Continued from previous page

Tool	**Advantages**	**Disadvantages**
Glomosim	Offers integrated functions that specifically focus on MAC schemes.	It is old and will take work to download and find tutorials.
Opnet	It can simulate many communication networks. It also provides a graphical user interface for easy configuration of a simulation. Can perform measures during simulations to see the effect of changes.	It is not free, and we do not have access to it.
OMNeT++	Open-source focuses on the OSI layer's lower layers, including MAC schemes.	It requires us to code each component in the simulation, and it is C++.
Java-DSP	It can perform signal processes and is coded in Java.	It is not free.
Scilab	An open-source version of MATLAB.	Few tutorials and fewer functions and tools than MATLAB.
GNS3	Open-source, user friendly.	Too general for our liking.
QualNet	Can emulate different dynamic changes during the simulation.	Minimal information about it, would be hard to learn.

Chapter 4

Exercises

4.1 Analysis and Requirements

In this chapter, before proceeding, we will present a visual representation that shows which exercise we will use for each tool, as seen in Figure 4.1. Under the figure of the visual representation that shows the exercises, we also will provide reasoning for our choices for the various tools for the exercises. After that, we will present the exercises highlighting the characteristics and essential functions of the schemes OFDM, CSMA with CA and CD, CDMA/FSHH, and CDMA/CDMA. Our approach for the exercises involves formulating problems that require a solution. The solution will demonstrate the scheme's characteristics and functionalities.

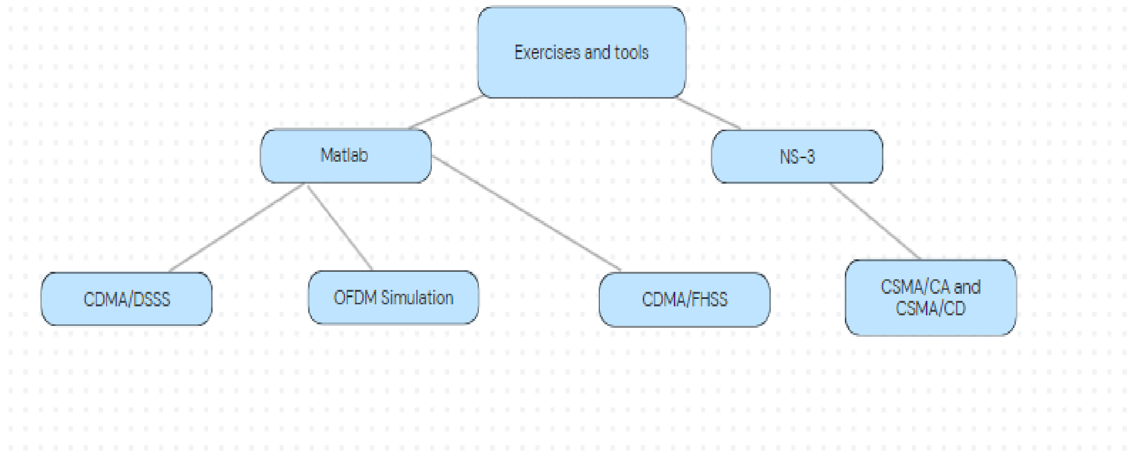
When designing the exercises, we want the exercises to be structured differently from each other. For instance, we want an exercise that simulates a whole MAC scheme. Additionally, we aim to highlight packet measurements by implementing an exercise that observes the transmission and reception of packets. The throughput, collisions, and packet loss, will be the measurements used in that observation. After the exercise, the student must individually analyze and evaluate the results. Another exercise will be a coding exercise. However, the coding part is simple. We want the student to implement the functionalities of the MAC scheme themselves, resulting in a better understanding because of practical learning. We also want the student to be able to play around with these exercises. In one exercise, we will allow the student to change the parameters so that the simulation will be changed and give a different result.

In Chapter 5, we will present and evaluate the results of the exercises. However, we will not present the results of the OFDM exercises, only the evaluation, because we will guide the student through the OFDM scheme step by step for the student's learning experience. Therefore, the student will not be required to complete any specific task related to the OFDM exercise. Instead, the student will have to follow our instructions and pay attention to how the signal changes through the different phases of the OFDM scheme.

4.2 Overview

Figure 4.1 describes what tools will be used for each exercise.

Figure 4.1: Overview figure



We will use MATLAB for three exercises since it is a matrix laboratory, making it easy to use. It specializes in matrix calculations for the OFDM, CDMA/FSHH, and CDMA/DSSS exercises. For the exercise with FHSS, matrix calculations are essential since we want the student to implement a part of the simulation. Additionally, we have seen that MATLAB provides functions that are useful for various schemes. The functions are modulation, IFFT, and FFT, which are essential for the OFDM scheme. For the DSSS exercise, we will use MATLAB because it is an easy tool to use, and it has some functions like XOR and vector multiplication, as stated before. That will help us in creating the exercises.

NS-3 will be used to perform the CSMA exercise. It includes libraries of the CSMA scheme, which is our main focus in this exercise.

4.3 CSMA/CD and CSMA/CA

4.3.1 Motivation

The end goal of this exercise is to make the student learn the hidden terminal problem. NS-3 has the CSMA/CD, and CSMA/CA functions in the library, making the exercise easier to construct. As explained in the background, the hidden terminal problem is a characteristic problem. Therefore, simulating the problem will give the student a better understanding of how it affects the throughput and number of packets received. These measurements help the student comprehend the hidden terminal problem and its solution. The student will understand how these schemes operate under the hidden terminal environment by looking at the simulation results. The hidden terminal is a flaw in CSMA/CD, and by comparing the flaw to the solution, the CSMA/CA. The student will understand the difference between these two.

4.3.2 Description

As mentioned, we will divide this exercise into two parts. The first one will be simulating the hidden terminal problem. For simulating the hidden terminal problem, the nodes must be placed in a specific way, as seen in Figure 4.2.

Figure 4.2: Node positioning

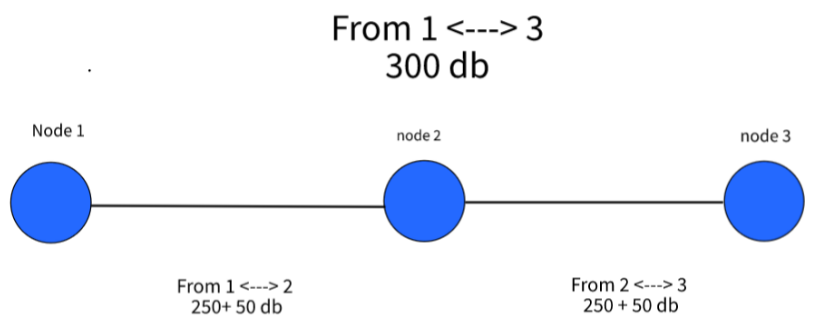


Figure 4.2 has a propagation loss model where data will wither after 250 dB. We will add 50 dB of propagation loss when node 1 transmits data to node 2. Therefore node 3 will be unable to sense node 1's packets, and node 1 will not sense node 3's packets, resulting in collisions. The propagation loss model is a class in NS-3 that enables the simulation of the hidden terminal problem.

To distinguish the CSMA/CD from CSMA/CA, we set a threshold to activate or deactivate the RTS/CTS. The threshold will initially be 2100 bits, and the size of the packets will be 1400 bits. Since the threshold is bigger than the packet size, the RTS/CTS will be disabled, and the student will see that the result are poor. The throughput and received packets will be bad.

In the second part, the student will set a threshold of 1000 bits and rerun the simulation. The 1000 bits threshold will now be under the packet size, activating the RTS/CTS. The student will then compare the two simulations and notice that the throughput and the number of received packets will be better.

Fortunately, NS-3 has an example-wireless file that simulates the hidden terminal, which we will use for this exercise. We will eliminate unnecessary functions and parameters for this exercise. The exercise takes inspiration from this script [23].

4.3.3 Implementation

Now we will explain the code used for implementation and the information needed so that someone can set up the same simulation. The script [23] gives the results on both with RTS/CTS and without it. Here is the code in Listing 4.1:

```
    std::cout << "Hidden station experiment with RTS/CTS disabled:\n"
 ↪  << std::flush;
  experiment (false);
  std::cout << "------------------------------------------------\n";
  std::cout << "Hidden station experiment with RTS/CTS enabled:\n";
   experiment (true);
```

Listing 4.1: NS3code

We removed the code above because we wanted the simulation of the two schemes consequently. First, simulating when the RTS/CTS was deactivated by the defined threshold. Then we simulate when the packet size exceeds the threshold, thus activating the RTS/CTS. The threshold enables us to distinguish the two simulations easier. The code had a function called *experiment* that simulates the scheme. The bool variables *true* and $false$ are to either enable or disable the RTS/CTS, as seen above in Listing 4.1.

We initialize the default value for the threshold parameter using the $WifiRemoteStationManager$ class. This class is used to manage the behavior of the WiFi stations and is necessary for the simulation. The student can see that the threshold is 2100 bits, as stated before. The $WifiRemoteStationManager$ will only activate the

RTS/CTS for each node when transmitting a packet with a size above the threshold. Here is the code in Listing 4.2:

```
Config::SetDefault("ns3::WifiRemoteStationManager::RtsCtsThreshold",
    ↪ UintegerValue(2100));
```

Listing 4.2: NS3code

Before this, there was a different code. This code took in the bool variable *false* or *true*, as seen in Listing 4.1, which defined the variable *ctsThr* as seen below in Listing 4.3, that could either activate or deactivate the CTS/RTS. By changing it to Listing 4.2, we only need to change the threshold number to activate or deactivate the RTS/CTS. Here is the previous code in Listing 4.2:

```
UintegerValue ctsThr = (enableCtsRts ? UintegerValue (100) :
    ↪ UintegerValue (2200));
Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
    ↪ ctsThr);
```

Listing 4.3: NS3code

Secondarily, We place the nodes in constant positions during the simulation to prevent them from moving, using the class *constantpositionmobilitymodel* and creating the nodes using the *NodeContainer* class. The code loops through the nodes and adds the nodes in a constant position. Notice that node 1 to will be at index 0 and node three at index 2. That is needed to understand the other code sections. Here is the code in Listing 4.4:

```

    NodeContainer nodes;
    nodes.Create(3);



    for (uint8_t i = 0; i < 3; ++i)
    {
        nodes.Get(i)->AggregateObject(CreateObject<
    ↪ ConstantPositionMobilityModel>());
    }
```

Listing 4.4: NS3code

Then the code creates the propagation loss model with the help of the class *propagationlossmodel*. The code sets the parameter of the additional loss by 50 dB so that nodes 1 or 3 will not sense a transmission. Therefore when either node 1 or 3 tries to send to node 2. The other node cannot sense the channel since the propagation loss is more than 250 dB. That will create a collision because the other node will think the channel is free and send its packet. Here is the code in Listing 4.5:

```
Ptr<MatrixPropagationLossModel> lossModel = CreateObject<
↪ MatrixPropagationLossModel>();
lossModel->SetDefaultLoss(250); // set default loss to 200 dB (no
↪ link)
lossModel->SetLoss(nodes.Get(0)->GetObject<MobilityModel>(),
                   nodes.Get(1)->GetObject<MobilityModel>(),
                   50); // set symmetric loss 0 <-> 1 to 50 dB
lossModel->SetLoss(nodes.Get(2)->GetObject<MobilityModel>(),
                   nodes.Get(1)->GetObject<MobilityModel>(),
                   50); // set symmetric loss 2 <-> 1 to 50 dB
```

Listing 4.5: NS3code

Then the code creates a WiFi channel with a class $yanswifichannel$". That is a way of creating a WiFi transmission channel that can be modified. The code adds the $lossModel$ to the channel and a *ConstantSpeedPropagationDelayModel* from the class *propagationdelaymodel* that was defined before. That will set the delay from the sender and receiver to constant, and the simulation cannot alter it. That means that the channel will behave as we design it. Here is the code in Listing 4.6:

```
 Ptr<YansWifiChannel> wifiChannel = CreateObject<YansWifiChannel>();
wifiChannel->SetPropagationLossModel(lossModel);
wifiChannel->SetPropagationDelayModel(CreateObject<
↪ ConstantSpeedPropagationDelayModel>());
```

Listing 4.6: NS3code

The devices must be wireless. As the student can see in the code, we use the $WIFI\_STANDARD\_80211b$ standard for the whole network with the help of the class $wifihelper$. This class makes it possible to create wireless network components. Additionally, we need to configure the physical layer and the MAC layer. We implement this using the built-in classes $YansWifiPhyHelper$ and $WifiMacHelper$. Also, since there is no need for a centralized access point to simulate the hidden terminal scenario, the code uses an ad-hoc wireless network. The $wifiManager$ variable was defined in the main and passed into the simulation function, the variable Arf, an algorithm of how $WifiManager$ operates.

Here is the code in Listing 4.7:

```
    WifiHelper wifi;
    wifi.SetStandard(WIFI_STANDARD_80211b);
    wifi.SetRemoteStationManager("ns3::" + wifiManager + "WifiManager");
    YansWifiPhyHelper wifiPhy;
    wifiPhy.SetChannel(wifiChannel);
    WifiMacHelper wifiMac;
    wifiMac.SetType("ns3::AdhocWifiMac");
    NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, nodes);
```

Listing 4.7: NS3code

Then we use the classes $internet - stack - helper$ and $ipv4 - address - helper$ to set up the communication between the nodes with a base IP address that will assign to the nodes. That is because then the nodes can communicate wirelessly with each other. Here is the code in Listing 4.8:

```
     InternetStackHelper internet
    internet.Install(nodes);
    Ipv4AddressHelper ipv4;
     ipv4.SetBase ("10.0.0.0", "255.0.0.0");
    ipv4.Assign(devices);
```

Listing 4.8: NS3code

The code can now create the data stream for the simulation. The code uses the $OnOffHelper$ from the class $on - off - helper$ to create a User Datagram Protocol (UDP). UDP is because of its simplicity, and it does not rely on acknowledgment messages or re-transmission to ensure packet delivery. The code also sets the target for the stream to node 2 with a port of name of our choosing which is 12345. The $ApplicationContainer$ is used to generate a data stream to node 2, which will be explained in more detail in Listing 4.10 and 4.11. It also sets the packet size to 1400 bits which imply that RTS/CTS will be disabled. Here is the code in Listing 4.9:

```
ApplicationContainer cbrApps;
    uint16_t cbrPort = 12345;
    OnOffHelper onOffHelper("ns3::UdpSocketFactory",
                            InetSocketAddress(Ipv4Address("10.0.0.2"),
   ↪ cbrPort));
    onOffHelper.SetAttribute("PacketSize", UintegerValue(1400));
```

Listing 4.9: NS3code

For here the code creates the stream of data from nodes 1 to 2:

```
onOffHelper.SetAttribute("DataRate", StringValue("4000000bps"));
onOffHelper.SetAttribute("StartTime", TimeValue(Seconds(1.000)));
cbrApps.Add(onOffHelper.Install(nodes.Get(0)));
```

Listing 4.10: NS3code

As well as for 3 to 2. We set the *DataRate* to 4000 Kbps which will be our initial data rate which the student will have the opportunity to change in the exercise. After that, we add them all in the *cbrApps*, defined by our *ApplicationContainer* class.

```
onOffHelper.SetAttribute("DataRate", StringValue("4000000bps"));
onOffHelper.SetAttribute("StartTime", TimeValue(Seconds(1.000)));
cbrApps.Add(onOffHelper.Install(nodes.Get(2)));
```

Listing 4.11: NS3code

In the original script, the *StartTime* for both streams differed by 0.001 seconds. We did not see any difference from what we currently have. Therefore we went for the same *StartTime* for both streams.

Unfortunately, there is a problem with NS-3, and it does not have a good Address Resolution Protocol (ARP). ARP is used to discover the unique MAC addresses of the nodes and map them to their IP address, which is needed for the communication between nodes and solved by using the *UdpEchoClientHelper* class. This code sets up the echo packets that nodes 1 and 3 will send to node 2. Here is the code in Listing 4.12:

```
 uint16_t echoPort = 9;
UdpEchoClientHelper echoClientHelper(Ipv4Address("10.0.0.2"),
↪ echoPort);
echoClientHelper.SetAttribute("MaxPackets", UintegerValue(1));
echoClientHelper.SetAttribute("Interval", TimeValue(Seconds(0.1)));
echoClientHelper.SetAttribute("PacketSize", UintegerValue(10));
ApplicationContainer pingApps;
```

Listing 4.12: NS3code

In Listing 4.12 the code sets the start time for each echo packet transmission and puts them in a *pingApps* class defined above.

In the code in Listing 4.12, we initiated the echo packet. Here we use it to configure the behavior of the echo packets, as the student can see in Listing 4.13.

```
    echoClientHelper.SetAttribute("StartTime", TimeValue(Seconds(0.001))
↪ );
    pingApps.Add(echoClientHelper.Install(nodes.Get(0)));
    echoClientHelper.SetAttribute("StartTime", TimeValue(Seconds(0.001))
↪ );
    pingApps.Add(echoClientHelper.Install(nodes.Get(2)));
```

Listing 4.13: NS3code

In the original script, the *StartTime* for both the streams was different by 0.005 seconds which we did not see any difference with what we currently have. Therefore we went for the same *StartTime* for both streams.

We need to install the *FlowMonitor* for all the nodes to capture the measurements during the simulation. We store this information in the *Monitor* variable. The *FlowMinitor* class provides measurements such as the total number of packets, throughput, and packets transmitted and received. Here is the code in Listing 4.14:

```
  FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.InstallAll()
```

Listing 4.14: NS3code

The code in Listing 4.15 is needed to run the simulation. As the student can see, the simulation will stop after 10 seconds.

```
    Simulator::Stop(Seconds(10));
    Simulator::Run();
```

Listing 4.15: NS3code

The last part is to display the results in the terminal window. The code in Listing 4.16 places the measurement in the *stats* variable and loops through that. Note that we skip the two first elements in the *stats* variable. That is because the two first *FlowIds* are for the ECHO apps. The *FlowIds* is the ID added to the for the different measurements. We do not need them displayed.

```
monitor->CheckForLostPackets();
 Ptr<Ipv4FlowClassifier> classifier =
 DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();
 for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
↪ stats.begin();
      i != stats.end();
      ++i)
 {

     if (i->first > 2)
     {
         Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(i->
↪ first);
         std::cout << "Flow " << i->first - 2 << " (" << t.
↪ sourceAddress << " -> "
                   << t.destinationAddress << ")\n";
         std::cout << "  Tx Packets: " << i->second.txPackets << "\n
↪ ";
         std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
         std::cout << "  TxOffered:  " << i->second.txBytes * 8.0 /
↪ 9.0 / 1000 / 1000
                   << " Mbps\n";
         std::cout << "  Rx Packets: " << i->second.rxPackets << "\n
↪ ";
         std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
         std::cout << "  Throughput: " << i->second.rxBytes * 8.0 /
↪ 9.0 / 1000 / 1000
                   << " Mbps\n";
     }
 }
```

Listing 4.16: NS3code

Now for what the student will do in this exercise. The student will first keep the threshold parameter of 2100 bits and run the simulation. Chapter 5 will explain the results produced. After that, the student will change the threshold to 1000 bits, enabling the RTS/CTS, and rerun the simulation to see the effects.

4.4 CDMA/DSSS

4.4.1 Motivation

This exercise aims to demonstrate how PN sequence and spreading work in DSSS. An MLS sequence is a crucial characteristic of the DSSS, and the student will learn about it by creating one themselves. The student must verify the three properties through the LFSR. The student is responsible for the verification and it creates a better understanding of how an MLS sequence resembles. There will also be a second part of this exercise. The purpose of the second part of this exercise is to emphasize to the student that the sender and receiver must have identical MLS sequences to distinguish the data. The modulation will not involve a standard modulation technique since this is not our primary focus for this exercise. This modulation will turn the binaries to -1's from 0's and 1's from 1's. That will make it easier to spread and despread the data and show the student how the scheme generally operates.

4.4.2 Description

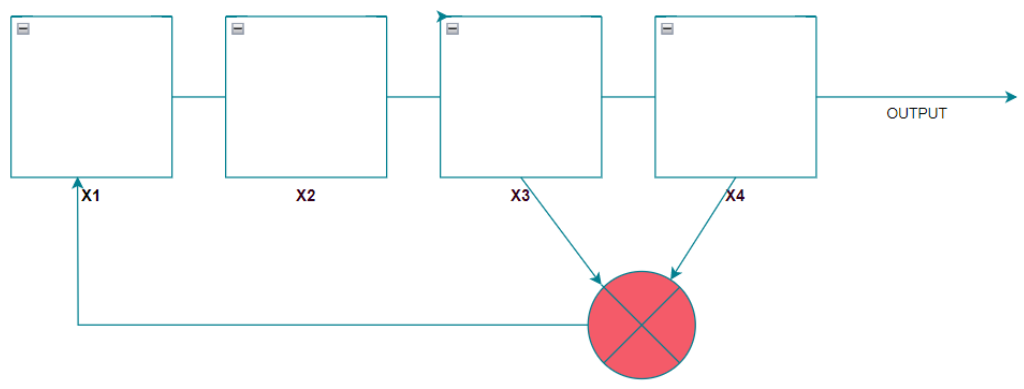
We will provide the student with a script that includes three different seeds for the LFSR. They must put the consequent seeds in the LFSR to generate an output. For each output, the student's assignment is to determine if the run and balance properties are satisfied. Studying the generated MLSes will be done to accomplish this. Then the auto-correlation must be verified, which is challenging by just studying the sequence. Therefore there will be a code for verifying auto-correlation in the LFSR script. We will compare the sequences to their shifted versions to perform the auto-correlation.

If these three properties are satisfied, then this MLS will be used in the next part of the exercise. A MATLAB script will implement primary senders and receivers in the next part. The student must then implement the MLS generated in the LFSR script. The following script will send 60 bits to 3 different receivers, where each receiver will randomly get allocated 15 bits. If the MLS for the sender and receiver does not match, the bits intended for the receiver will not be delivered.

Because the data will spread and despreaded differently, resulting in the received data not being equal to the original data. As a result, the receiver may mistakenly assume the receiver data is not intended for it. The student will observe a message in the command prompt if this is the case and, conversely, if the MLS sequence for the receiver is equal to that of the corresponding sender. Resulting in correct spreading and despreading, the print will then indicate delivery.

To generate the MLSes, we will model the code as Figure 4.3 a LFSR.

Figure 4.3: LFSR register



4.4.3 Implementation

To set up the exercise, the teacher must first set up the LFSR. That is for the creation of the MLS sequence. The student will put the seed in the *Seed* variable. Then the *flipflops* array will define the index in the seed, which is then subjugated to the XOR function. There is a line in Listing 4.17 that performs the shift:

```
seed(length_of_seed+1 - helperforshift)=seed(length_of_seed-
   ↪ helperforshift);
```

The initial length of the seed is 4. The line will perform a circular shift operation where the third element is defined as the fourth, and this operation repeats until one element is left. Notice if the length of the seed and the number of shifts change, then the third element will not be defined as the fourth. New elements will be chosen for the new parameters. But in this case, the third and fourth elements will perform an XOR operation. The XOR operation's output will be the new seed's first element.

Uncomment the commented *fprintf* function to see the state of the shifts.

```
seed=[];
flipflops=[3 4];

length_of_seed=length(seed);
state_of_register(1,:)=seed;
lenght_of_flipflops=length(flipflops);
for number_ofshifts=1:2^length_of_seed-2;
xored=xor(seed(flipflops(1)), seed(flipflops(2)));

helperforshift=1:length_of_seed-1;
seed(length_of_seed+1-helperforshift)=seed(length_of_seed-helperforshift
    ↪ );
seed(1)=xored;
state_of_register(number_ofshifts+1,:)=seed;
%fprintf('State of the register is: [%s]\n', join(string(
    ↪ state_of_register)))
end
output_sequence=state_of_register(:,length_of_seed)';
fprintf('PN-sequence: [%s]\n', join(string(output_sequence)));
```

Listing 4.17: MATLABcode

In the upcoming code Listing 4.18, the autocorrelation check will be presented. In this code, we circularly shift the whole sequence several times. For each time, we compare the shifted version with the original version. These are the *if_like* and *if_notlike*, and then we divide them on *lenght_of_sequence*. After that, we put them in the *correlation*(k) array. Ultimately, we see if every comparison is the same number when we divide the 1 on the *lenght_of_sequence*.

```
for number_ofshifts=1:lenght_of_sequence;
    circl_shifted=circshift(output_sequence, [1 number_ofshifts-1])
    if_like=0;
    if_notlike=0;
    for i=1:lenght_of_sequence;
        if output_sequence(i)==circl_shifted(i);
            if_like=if_like+1;
        else
            if_notlike=if_notlike+1;
        end
    end
    correlation(number_ofshifts)=(if_like-if_notlike)/lenght_of_sequence
   ↪ ;
end

they_are_correlated=0;
if correlation(2:lenght_of_sequence)== -(1/lenght_of_sequence);
    they_are_correlated=1;
end
if  they_are_correlated==1;
    fprintf('The Code satisfies Autocorrelation Property')
else
    fprintf('The Code does NOT satisfy the Autocorrelation Property')
end
    fprintf(' See the Figure 1')
```

Listing 4.18: MATLABcode

For the setup of the second part of the exercise, the senders and receivers, we first will set up the 60 bits that will be converted from 0's and 1's to -1's and 1's. The code is shown in Listing 4.19.

```
data=round(rand(1,60));
data=2*data-1;
```

Listing 4.19: MATLABcode

Next, the student will input the MLS sequences into the code, and the sequences will be converted into -1's and 1's to match the spreading code. see Listing 4.20

```
Spreading_for_station_A = [];
Spreading_for_station_A  = 2*Spreading_for_station_A  - 1;
Spreading_for_station_B = [];
Spreading_for_station_B = 2*Spreading_for_station_B - 1;
Spreading_for_station_C = [];
Spreading_for_station_C= 2*Spreading_for_station_C - 1;

Received_for_station_A = [];
Received_for_station_A = 2*Received_for_station_A - 1;
Received_for_station_B = [];
Received_for_station_B = 2*Received_for_station_B - 1;
Received_for_station_C = [];
Received_for_station_C = 2*Received_for_station_C - 1;
```

Listing 4.20: MATLABcode

Then we randomly allocate 15 bits to the stations by using switch cases. The *temp_data*, which is the data, will be multiplied by the respective station's spreading code. As seen in the code in Listing 4.21:

```
    spreaded_data=[];
for dividerofbits=1:15:60
    temp_data = data(1, dividerofbits : dividerofbits + 14);
    random = randi([1,3]);
    switch(random)
        case(1)
                spreaded_data = [spreaded_data Spreading_for_station_A
    ↪ .* temp_data];
        case(2)
               spreaded_data = [spreaded_data Spreading_for_station_B .*
    ↪  temp_data];
        case(3)
            spreaded_data = [spreaded_data Spreading_for_station_C .*
    ↪ temp_data];
    end
end
```

Listing 4.21: MATLABcode

After that, the receiver's code is shown in Listing 4.22. Here we check for each of the 15 bits in the $temp2 - data$ by multiplying them with each station spreading codes to despread it. Then we compare it with the 15 bits of the original data.

```
    for counter=1:15:60
    temp_data =  data(1, counter : counter + 14);
    temp2_data=spreaded_data(1,counter:counter+14);
    if temp2_data.*Received_for_station_A==temp_data
        fprintf('\n bit number %d to %d was meant for station A.',
    ↪ counter,counter+6);
    elseif temp2_data.*Received_for_station_B==temp_data
          fprintf('\n bit number %d to %d was meant for station B.',
    ↪ counter,counter+6);
    elseif temp2_data.*Received_for_station_C==temp_data
        fprintf('\n bit number %d to %d was meant for station C.',
    ↪ counter,counter+6);
    else
        fprintf('\n bit number %d to %d could not be received.',counter,
    ↪ counter+6);
    end
end
```

Listing 4.22: MATLABcode

Now we will explain the task the student must do. The student will receive the seeds for this exercise, which they will input into the LFSR script. Here are the seeds: $[1, 0, 0, 0], [1, 1, 1, 1], [0, 1, 0, 1]$. After generating the MLS sequences, the student must verify them and then load the other script. In the new script, the student must put the MLS sequences in the senders and receivers spreading code array and then run the script.

Here is the code again:

```
Spreading_for_station_A = [];
Spreading_for_station_A  = 2*Spreading_for_station_A  - 1;
Spreading_for_station_B = [];
Spreading_for_station_B = 2*Spreading_for_station_B - 1;
Spreading_for_station_C = [];
Spreading_for_station_C= 2*Spreading_for_station_C - 1;

Received_for_station_A = [];
Received_for_station_A = 2*Received_for_station_A - 1;
Received_for_station_B = [];
Received_for_station_B = 2*Received_for_station_B - 1;
Received_for_station_C = [];
Received_for_station_C = 2*Received_for_station_C - 1;
```

Listing 4.23: MATLABcode

All stations will receive their designated bits if the student has implemented them correctly.

4.5 CDMA/FHSS

4.5.1 Motivation

The FHSS's key characteristics involve hopping through different frequencies during data transmission. This unique feature is the main topic of this exercise. It involves assigning the frequencies to the frequency carriers, implementing the random hopping mechanism, and setting up the receiver. We designed this exercise such that the receiver will act as a verification that the implementation is correct. This exercise will help the student to get a comprehensive understanding of how the FHSS scheme works.

4.5.2 Description

The student will get the script of a basic FHSS scheme. However, the frequency allocation, hopping, and receiver are up to the student to implement. We will provide pointers to help the student. However, they can implement their creative solution if the receiver's data corresponds with the transmitted data. The structure of the exercise is to send 20 bits over to a receiver. Each bit is converted to 60 chips, either 1 or -1, depending if the bit is a 1 or a 0. Each bit with 60 chips will also be put into its own carrier during the hopping mechanism, resulting in 20 randomly chosen frequency carriers from the initial 5 carriers. The modulation is a simple implementation of BPSK. That is a basic implementation since we focus on how the FHSS algorithm works.

The first part is the implementation of the frequency carrier. We will provide the specifics for the frequency carriers to the student. Such as the length of the main frequency spectrum and the five frequency carriers. The spectrum of these frequencies is from 0 to 2*π, where each frequency carrier is calculated by $2 * \pi/(Afixednumber)$. These numbers obtain by using the try-and-fail method. The frequency spectrum numbers were the most optimal for an excellent visual representation of the plot. We will provide the student with fixed numbers and the frequency spectrum for the modulated signals. The number we have chosen is just one that makes showcasing the different frequencies easier. The significance of the fixed number just makes the plot easier to understand.

To implement hopping, the student must create a code randomly selecting the carrier to modulate the data. When coding the hopp mechanism, the student will be asked to change the carrier for each bit. Thus, the code must be implemented in a for-loop with switch cases. At last, the student will be asked to verify that the implementation is correct by creating the receiver for the sender. For this, the despreading, demodulating, and plotting of the signal must be implemented. We will give the student a pointer that the receiver can be done by doing the inverse of the sender. This will show that the original data stream will equal the despreaded and demodulated signal.

4.5.3 Implementation

To implement this exercise, we first create 20 random binary bits. Then we create the $main_carrier$ with 60 chips resulting from the calculation shown above. We take the primary frequency's whole length from 0 to 2π and then increment it by $2 * pi/59$ until it reaches 2*π. Then we transform the frequency from a time domain to a frequency domain by using the cos operation on the $main_carriersample$ Furthermore, we go through each bit, and if the bit is a 0, we multiply it with -60 chips. If the bit is 1, then we multiply it by 60 chips. After this operation, we are left with 1200 chips. Since the data is not equal in length to the carrier, we need to extend the $main_carrer$ to length 1200.

In Listing 4.24 the code:

```
bits=round(rand(1,20));
signal=[];
main_carrier=[];
main_carrier_sample=[0:2*pi/59:2*pi];
samples_ofcarriers=cos(main_carrier_sample);
for index_ofbits=1:20
    if bits(1,index_ofbits)==0
        chips=-ones(1,60);
    else
        chips=ones(1,60);
    end
    main_carrier=[main_carrier samples_ofcarriers];
    signal=[signal chips];
end
```

Listing 4.24: MATLABcode

Now that we have the same length, we can modulate the data stream to the main carrier. Which is done in the line of code in Listing 4.25:

```
signal_modualted=signal.*main_carrier;
```

Listing 4.25: MATLABcode

All the codes above are the only thing the student will receive for this exercise. The rest the student must implement. How to implement the code is written in the *TODO* section of the script, as seen in Listing 2.26. We provide some tips the student can follow and examples of variables they can use. A possible answer for this implementation will be provided later in chapter 5.

```
% TODO: Implement the carrier frequencies here. Where the frequency
    ↪ spectrum
should be from 0 to 2*pi. The interval should be 2*pi/ "fixed number".
Where the fixed number is 9, 19, 29, 39, and 59 for each frequency
    ↪ carrier.
Then Add a cos to the samples. Note that carriers will differ in length
    ↪ since
we divide 2*pi with different numbers. So you must take that into
    ↪ consideration.
The length of each frequency carrier should be 60. For example, the
    ↪ first carrier
with a fixed number 9 will generate 10 samples. Therefore you must put
    ↪ the
samples 6 times into an array to make the length of the array 60.
% Implement Here:
```

Listing 4.26: MATLABcode

The student must also implement the code for choosing which carrier frequency to select for each bit. Chapter 5 will provide a possible implementation. see Listing 4.27

```
% TODO: Implement the hopp mechanism here. Tip: Each bit should be
placed randomly. For each n, we need to choose a carrier, then create
a random function that picks a random carrier and puts it into
the carrier_for_spreading array.
The carrier array will be used to multiply with the data array,
creating the FHSS signal. %
carrier_for_spreading=[];
for n=1:20
    %implement here


end
```

Listing 4.27: MATLABcode

That will result in that $carrier_for_spreading$ array will no longer be empty. It will therefore be multiplied with the data, creating the FHSS signal in the code. see code in Listing 2.28

```
fhss_signal=signal_modualted.*carrier_for_spreading;
```

Listing 4.28: MATLABcode

For the last part which is shown in Listing 4.29, the student must despread the signal by dividing the modulated data on the received data stream. That will be a basic receiver with the same spreading algorithm as the sender and therefore be able to extract the data.

```
% TODO: Create the receiver here. A tip is to reverse the different
phases of the sender. Then set
the answer as the variable received_bits.
%received_bits=...
```

Listing 4.29: MATLABcode

If the $receivied_bits$ match the original signal, then several plots will appear. We will interpret these figures in chapter 5. The goal of these figures is to explain the different processes of FHSS.

4.6 OFDM simulation

4.6.1 Motivation

For the last exercise, the student will learn about the different steps of how OFDM schemes work. We will initially present the code in the comments. Thus the student must uncomment each section consecutively. We will explain what the student sees. The sections will show the student how the signal gets separated into different carriers, how it modulates with PSK, IFFT's modification on the signal, the parallel to serial conversion and the reverse, and how the FFT works on the signal. In the end, the student will know about the different processes in OFDM. Through each step, provide a plot.

4.6.2 Description

The exercise will separate the OFDM into different learning parts:

- Serial to parallel
- Modulation

- IFFT
- adding cyclic prefix
- Parallel to serial
- Serial to parallel
- FFT
- removal of cyclic prefix
- De-modulation
- parallel to serial

For each part which sometimes will be divided into several sections for ease of explanation, the student will take away the comment on a specific code section. The result of each part will show the student how that OFDM section functions. However, the subcarriers will not be orthogonal as in true OFDM, and they will just be vectors. This vector will be just the data divided into different vectors, and these vectors will be the subcarriers. Additionally, when we create the serial signal, the vectors will be added together into one big vector.

We have structured the code to first send 256 bits, separated into four subcarriers carrying 64 bits each. The bits generated will be random for each simulation but will not affect the student's learning opportunities.

Then the PSK modulation will be performed. That is important because we will divide the data into subcarriers, and each subcarrier will carry a modulated signal. Then the CP length will be initially 3. The student can change this and experience that the signal will be longer or shorter because of the varying length of the cyclic prefix.

After all that, the parallel to the serial will commence when the receiver receives the signal. It turns the signal back into parallel subcarriers, removes the cyclic prefix, and then performs the FFT to reverse the IFFT. The demodulation will be activated, resulting in the original signal as sub-carriers. We convert the sub-carriers back into a serial signal. Then the original signal will be compared to the received signal, and they will match.

The code will have parameters for the simulation, which the student can change if wanted. However, if they change the subcarriers parameter, they must add additional subcarriers. That is not a problem since they copy and paste the existing subcarriers. The implementation will be what the student will go through, which is the whole code. Therefore, this exercise's results will not be in chapter 5 but only in the implementation of the exercise.

4.6.3 Implementation

We begin with the parameters for the OFDM simulation. Here we will set the parameters we will use in the rest of the simulation. Including the number of bits per carrier as seen in Listing 4.30:

```
number_of_subcarriers = 4;
subcarrier_size = 16; % size of each subcarrier
CP_len = 3; %  the length of cycle prefix
number_of_bitstotal= 256; % number of bits the sender transmit
number_of_bitspercarrier=number_of_bitstotal/number_of_subcarriers;
```

Listing 4.30: MATLABcode

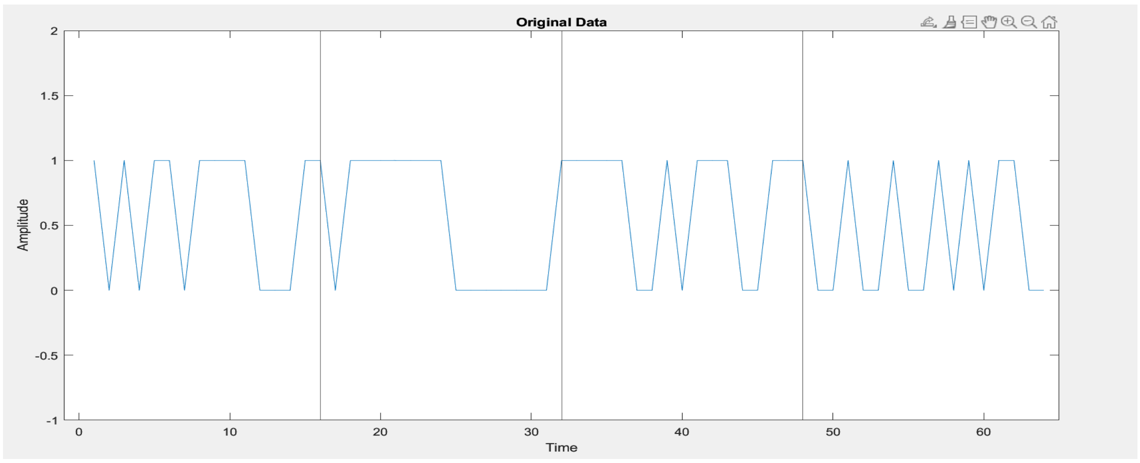
Here is the separation into sub-carriers. They need to uncomment that code section in Listing 4.31:

```
data = randi([0 1], number_of_bitspercarrier, 1);
```

Listing 4.31: MATLABcode

Suppose the student plotted the data with an interval of 16,32 and 48 with the MATLAB function *xline* for each subcarrier. That will generate a plot that will look like Figure 4.4.

Figure 4.4: Original Data



That will show the original data before being separated into four different subcarriers. The *xlines* enables us to see each subcarrier in the original data.

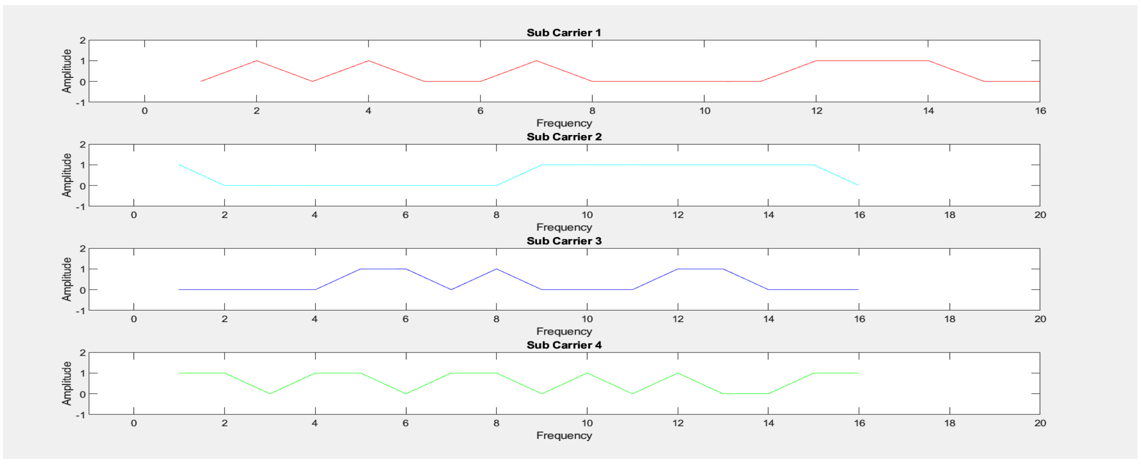
After that, the student will uncomment the block section in Listing 4.32. This section separates the matrix into four different vectors while performing PSK modulation on each. That will result in the variable $Serial2Parallel$. However, we can separate the matrix into different vectors. Therefore we can more easily plot them.

```
Serial2Parallel = reshape(pskmod(data, number_of_subcarriers),
number_of_bitspercarrier/number_of_subcarriers, number_of_subcarriers);
subcarrier1 = Serial2Parallel(:,1);
subcarrier2 = Serial2Parallel(:,2);
subcarrier3 = Serial2Parallel(:,3);
subcarrier4 = Serial2Parallel(:,4);
```

Listing 4.32: MATLABcode

Suppose the student decides to plot the different subcarriers. They will see how subcarriers look, as seen in Figure 4.5.

Figure 4.5: Subcarriers



The student can also use the subcarrier variables from above, to perform IFFT on them, and plot them to see how they differ from Figure 4.5.

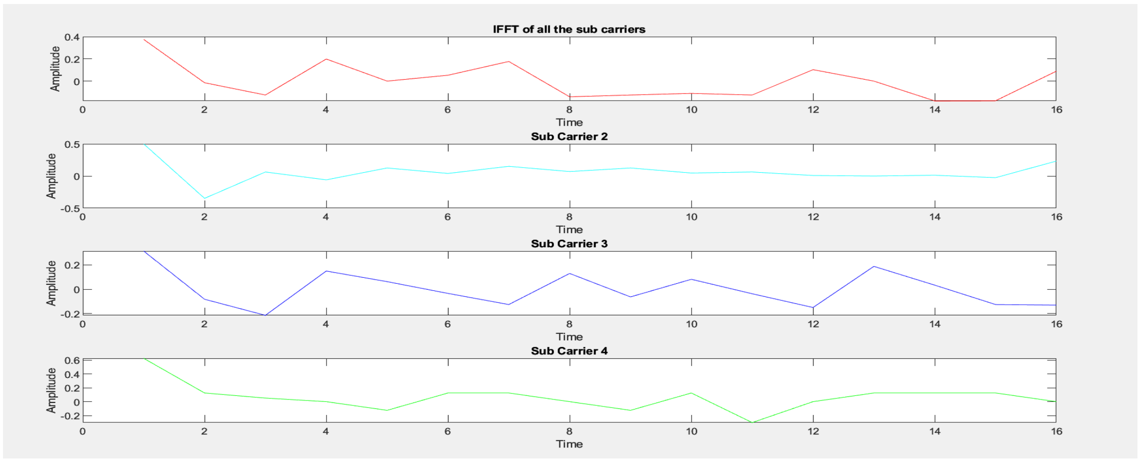
```
CP_start = block_size- CP_len;
ifft_subcarrier1 = ifft(subcarrier1);
ifft_subcarrier2 = ifft(subcarrier2);
ifft_subcarrier3 = ifft(subcarrier3);
ifft_subcarrier4 = ifft(subcarrier4);
```

Listing 4.33: MATLABcode

If the student plots the carriers now, they will know they are different from the initial carriers as seen in Figure 4.6.

Now the student will un-comment the next code segment shown in Listing 4.34. First, we define the CP_start that we will use as a start for the copying of the last elements.

Figure 4.6: Subcarriers with IFFT



In the first for loop, the IFFT will be performed on each subcarrier. Note that the IFFT performed on the carries above are not real carriers, but just to show the student how they look. In the next for loop, the cyclic prefix will be converted into an array and put in front of every subcarrier. This will generate a matrix *appned_prefix*.

```
CP_start = subcarrier_size- CP_len;
for subcarrier_index = 1:number_of_subcarriers
    ifft_subcarriers(:,subcarrier_index)= ifft(Serial2Parallel
    (:,subcarrier_index), 16);
for CP_index = 1: CP_len,
    cycle_prefix(CP_index,subcarrier_index) = ifft_subcarriers(CP_index+
    ↪ CP_start, subcarrier_index);
end
appended_prefix(:,subcarrier_index) = vertcat(cycle_prefix(:,
    ↪ subcarrier_index), ifft_subcarriers(:,subcarrier_index));
end
```

Listing 4.34: MATLABcode

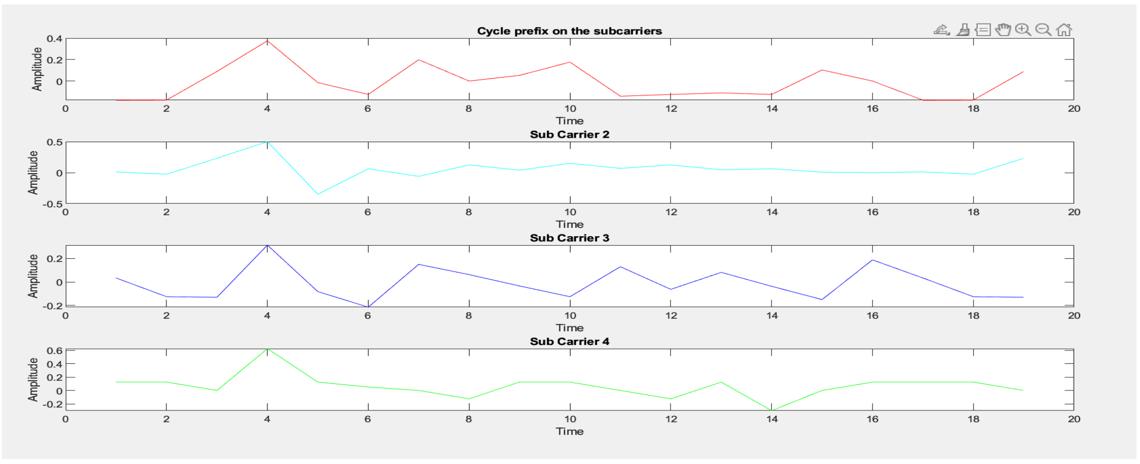
Supposing the student wants to see how the different subcarriers look like, they will just have to uncomment this code in Listing 4.35. This will divide *append_prefix* into 4 different which the student can plot to see each subcarrier with the prefix.

```
subcarrier1_prefix = appended_prefix(:,1);
subcarrier2_prefix = appended_prefix(:,2);
subcarrier3_prefix = appended_prefix(:,3);
subcarrier4_prefix = appended_prefix(:,4);
```

Listing 4.35: MATLABcode

Providing that the student now plots these carriers. We can see that the length of each carrier is extended because of the cyclic prefix, as seen in Figure 4.7.

Figure 4.7: Subcarriers with cycle prefix



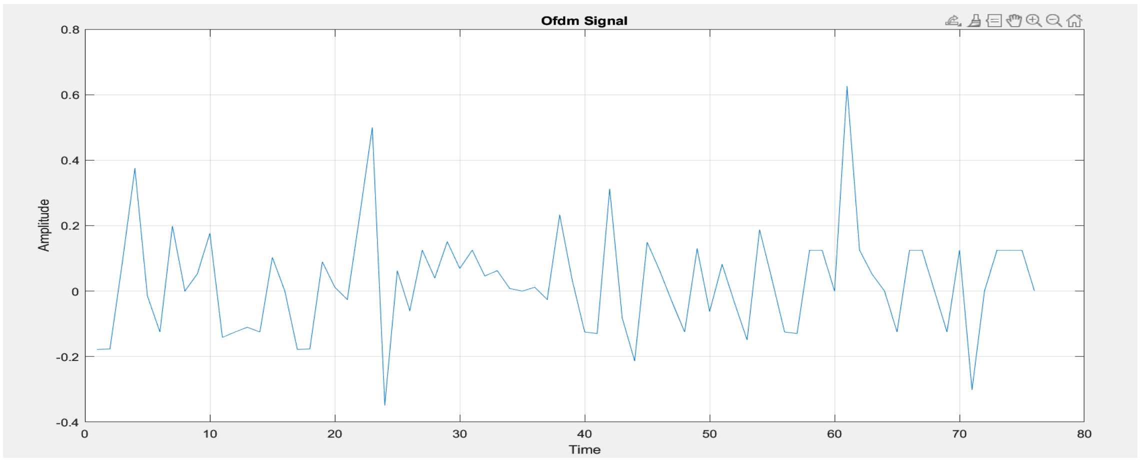
In the following code segment in Listing 4.36, the student will uncomment the parallel to the serial conversation. We do this by creating two vectors from *appended_prefix* and multiplying them to get the length of the OFDM data. After that, we reshape the data into one serial signal called *ofdm_signal*. As stated, the serial is just the subcarriers added together for simplicity.

```
[rows_append_prefix, col_append_prefix] = size(appended_prefix);
len_ofdm_data = rows_append_prefix*col_append_prefix;
ofdm_singal = reshape(appended_prefix,1,len_ofdm_data);
```

Listing 4.36: MATLABcode

The student may decide to plot the signal. Resulting in the student witnessing the OFDM signal and seeing that sections of the signal represent each subcarrier. Figure 4.8 shows the OFDM signal.

Figure 4.8: OFDM signal



The second to last part the student will uncomment is where we reshape the serial into a matrix with the subcarriers. Then we remove the cyclic prefix on each carrier by iterating through the matrix of *received_data_matrix* as seen in Listing 4.37. After that, we will perform the FFT on each subcarrier in the matrix.

```
received_data_matrix = reshape(ofdm_singal, rows_append_prefix,
    ↪ col_append_prefix);
received_data_matrix = received_data_matrix(CP_len+1:end,:); % remove
    ↪ cycle prefix
% perform FFT on each subcarrier
for subcarrier_index = 1:number_of_subcarriers
    received_data_matrix(:,subcarrier_index) = fft(received_data_matrix
    ↪ (:,subcarrier_index));
end
```

Listing 4.37: MATLABcode

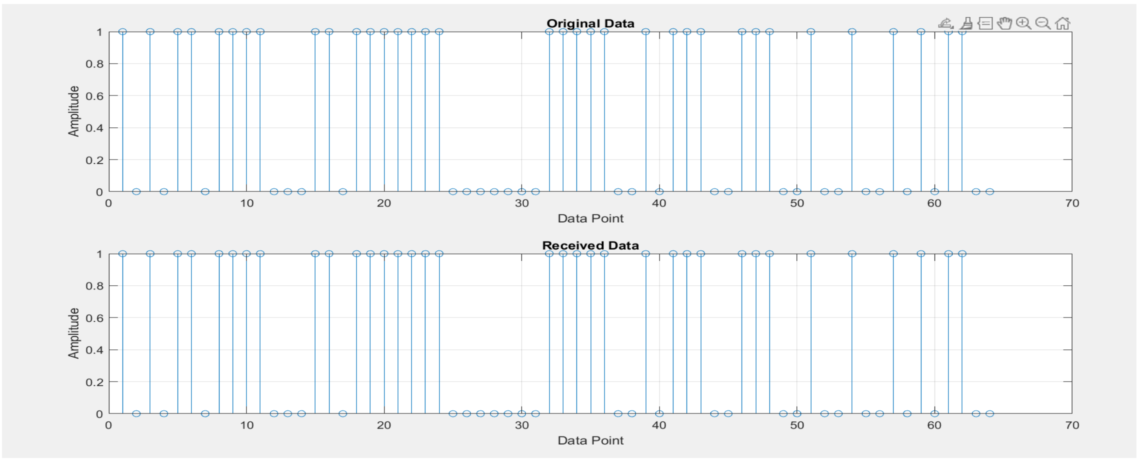
In Listing 4.38 the student can see that we perform the PSK demodulation on each subcarrier in the matrix and reshape it into a serial signal by adding them up.

```
demodulated_data = pskdemod(reshape(received_data_matrix,
number_of_bitspercarrier,1),number_of_subcarriers);
```

Listing 4.38: MATLABcode

The student can then plot both the *demodulated_data* and the original data and see that they are equal. Figure 4.9 shows this.

Figure 4.9: Comparison of signals



Overall, this exercise has gone through the key characteristics of OFDM. Its various processes in transmitting and receiving the data. However, as stated before, there is a lot of simplification, which we will dive deeper into in the evaluation in chapter 5.

Chapter 5

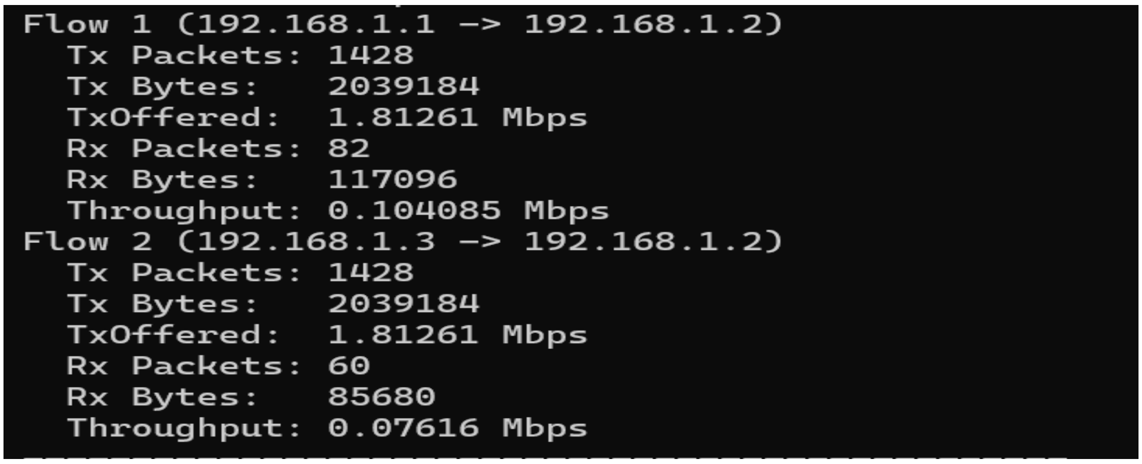
Results and Evaluations

In this chapter, we will present the results through images and evaluate them. When evaluating, we will focus on explaining what the result means and the pros and cons of the exercises for learning. Could we have done things better concerning the result? Moreover, was the result as we expected? and what did we expect the student to notice and answer? The results and evaluations are presented in the same order as Chapter 4.

5.1 CSMA/CD and CSMA/CA Results and Evaluation

When the student first simulates with the same RTS/CTS threshold as described in the implementation, the result will look like Figure 5.1.

Figure 5.1: CSMA without RTS/CTS

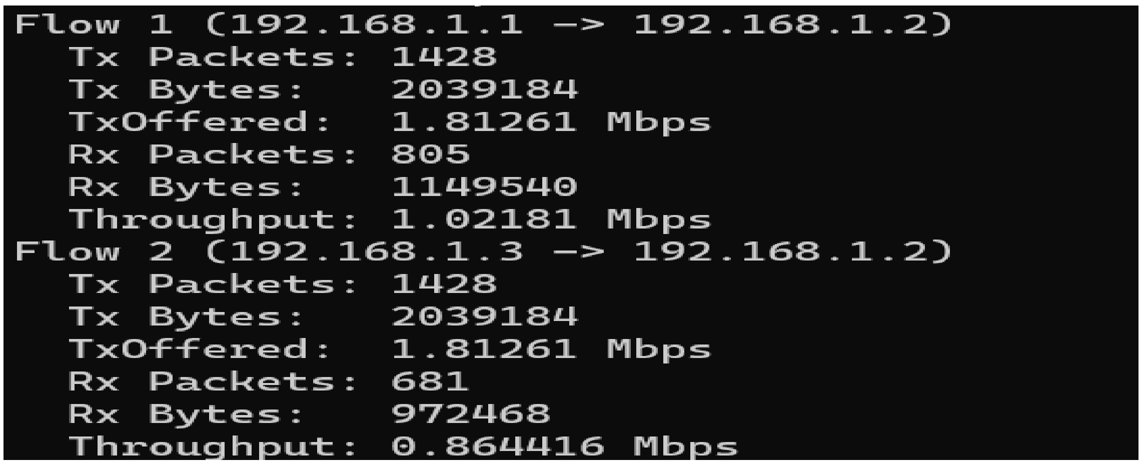


```
Flow 1 (192.168.1.1 -> 192.168.1.2)
  Tx Packets: 1428
  Tx Bytes:   2039184
  TxOffered:  1.81261 Mbps
  Rx Packets: 82
  Rx Bytes:   117096
  Throughput: 0.104085 Mbps
Flow 2 (192.168.1.3 -> 192.168.1.2)
  Tx Packets: 1428
  Tx Bytes:   2039184
  TxOffered:  1.81261 Mbps
  Rx Packets: 60
  Rx Bytes:   85680
  Throughput: 0.07616 Mbps
```

Here the student can see that the throughput of node 1 to node 2 is 104.085 Kbps, and received packets are only 82 compared to the transmitted 1428 packets. The bytes received are as well much less than the sender sent. The throughput for nodes 3 to 2 is 76.16 Kbps, and the received packet is 60 compared to 1428. The student will conclude that this is not a good result, a consequence of the simulation. In addition, the student can also conclude that the hidden terminal problem is the reason for the bad results.

When the student change the threshold of RTS/CTS to 1000 bits, enabling the RTS/CTS. The result will be like Figure 5.2.

Figure 5.2: CMSA with RTS/CTS



```
Flow 1 (192.168.1.1 -> 192.168.1.2)
  Tx Packets: 1428
  Tx Bytes:   2039184
  TxOffered:  1.81261 Mbps
  Rx Packets: 805
  Rx Bytes:   1149540
  Throughput: 1.02181 Mbps
Flow 2 (192.168.1.3 -> 192.168.1.2)
  Tx Packets: 1428
  Tx Bytes:   2039184
  TxOffered:  1.81261 Mbps
  Rx Packets: 681
  Rx Bytes:   972468
  Throughput: 0.864416 Mbps
```

Here the student will see that the throughput for node 1 to node 2 is 1021.81 Kbps, and there are 805 received packets compared to the transmitted 1428 packets. Similarly, node 3 to node two will have a throughput of 864.416 Kbps and received packets will be 681 compared to the sender 1428.

With these results, the student can distinguish the two schemes by their very different performances under the simulation. Additionally, the student must now understand why the CTS/RTS are suitable schemes to increase performance. The exercise is good because it accurately simulates the schemes. We expected to see these results, highlighting the scheme's characteristics. Overall, this high-end exercise demonstrated a big problem with CSMA/CD and CSMA/CA's CTS/RTS solution.

5.2 CDMA/DSSS Results and Evaluation

By setting the seeds variables *seed* as provided in the LFSR script. The student will get the MLS sequences which is shown in Listing 5.1:

```
    PN-sequence: [0 0 0 1 0 0 1 1 0 1 0 1 1 1 1]
    PN-sequence: [1 1 1 1 0 0 0 1 0 0 1 1 0 1 0]
    PN-sequence: [1 0 1 0 1 1 1 1 0 0 0 1 0 0 1]
```

Listing 5.1: Results

The student will see that the balance property has been satisfied because there are 8 1s and 7 0s in each sequence. The run property has also been satisfied. There are 8 runs, where 50% are 0 runs and the other half 1. Also, the auto-correlation property is satisfied because the shifted version of the output sequence correlates with the output sequence.

The receiver and sender spreading codes should look like Listing 5.2 after the student have been generated the the MLS sequences:

```

Spreading_for_station_A = [0 0 0 1 0 0 1 1 0 1 0 1 1 1 1];
Spreading_for_station_A  = 2*Spreading_for_station_A  - 1;
Spreading_for_station_B = [1 1 1 1 0 0 0 1 0 0 1 1 0 1 0];
Spreading_for_station_B = 2*Spreading_for_station_B - 1;
Spreading_for_station_C = [1 0 1 0 1 1 1 1 0 0 0 1 0 0 1];
Spreading_for_station_C= 2*Spreading_for_station_C - 1;

Received_for_station_A = [0 0 0 1 0 0 1 1 0 1 0 1 1 1 1];
Received_for_station_A = 2*Received_for_station_A - 1;
Received_for_station_B = [1 1 1 1 0 0 0 1 0 0 1 1 0 1 0];
Received_for_station_B = 2*Received_for_station_B - 1;
Received_for_station_C = [1 0 1 0 1 1 1 1 0 0 0 1 0 0 1];
Received_for_station_C = 2*Received_for_station_C - 1;
```

Listing 5.2: Results

If the student had implemented it correctly, the presented result will be correct, although random. All the bits will be received by their stations, as seen in the result in Listing 5.3:

```
bit numbers 1 to 7 were meant for station C.
bit numbers 16 to 22 were meant for station C.
bit numbers 31 to 37 were meant for station C.
bit numbers 46 to 52 were meant for station A.
```

Listing 5.3: Results

However, if the student implemented it incorrectly, here is an example where C stations do not have the same MLSes, then Listing 5.4 will show:

```
bit numbers 1 to 7 were meant for station B.
bit numbers 16 to 22 could not be received.
bit numbers 31 to 37 were meant for station A.
bit numbers 46 to 52 could not be received.
```

Listing 5.4: Results

When finished with this exercise, the student should understand the different properties of MLS that must be satisfied. Including how CDMA/DSSS uses the spreading codes to distinguish the different bits from the sender to the receiver.

This exercise did not show what an accurate simulation could have done

The exercise did not have an accurate modulation. The modulation is simplified, but it shows the modulation part of the scheme in a good way. Additionally, the exercise lacks CDMA/DSSS's resistance to interference and its bit rate in general and during the impairments from interference. On the other hand, this was not our primary goal, but it would have been nice to have this implemented. It could result in a better learning experience overall.

5.3 CDMA/FHSS Results and Evaluation

A potential solution for dividing the carrier is adding a different fixed number to each increment. As mentioned in the implementation, a fixed number was given to the student. However, this number can be modifiable. The only important thing is that the carrier has the same length. This can be done by copying the exact values until they have the same length. In this case, each carrier has a length of 60. The code transforms the samples into the frequency domain. As shown in Listing 5.5:

```
sample1=[0:2*pi/9:2*pi];
sample2=[0:2*pi/19:2*pi];
sample3=[0:2*pi/29:2*pi];
sample4=[0:2*pi/39:2*pi];
sample5=[0:2*pi/59:2*pi];
carrier1=cos(sample1);
carrier1=[carrier1 carrier1 carrier1 carrier1 carrier1 carrier1];
carrier2=cos(sample2);
carrier2=[carrier2 carrier2 carrier2];
carrier3=cos(sample3);
carrier3=[carrier3 carrier3];
carrier4=cos(sample4);
carrier4=[carrier4 carrier4(1:20)];
carrier5=cos(sample5);
```

Listing 5.5: Results

One of many solutions for the hopping mechanism is doing a random switch for each bit. The student must know that the choice of carrier is random. Here we implemented a for loop that goes through each bit and randomly places them on a carrier. Each bit will be multiplied by 60 each. Thus, the $carrier\_for\_spreading$ will have a length of 1200 after the for-loop. Listing 5.6 is the solution:

```
carrier_for_spreading=[];
for index_ofbits=1:20
     different_carriers=randi([1, 5]);
    switch(different_carriers)
        case(1)
            carrier_for_spreading=[carrier_for_spreading carrier1];
        case(2)
            carrier_for_spreading=[carrier_for_spreading carrier2];
        case(3)
            carrier_for_spreading=[carrier_for_spreading carrier3];
        case(4)
            carrier_for_spreading=[carrier_for_spreading carrier4];
        case(5)
            carrier_for_spreading=[carrier_for_spreading carrier5];

    end
end
```

Listing 5.6: Results

The receiver solution will perform despreading and demodulating of the signal. If we follow the tip by doing the opposite of the sender, we take the FHSS signal and divide it by the *carrier_for_spreading* variable that we used to multiply into the FHSS signal. In Listing 5.7 is the code:

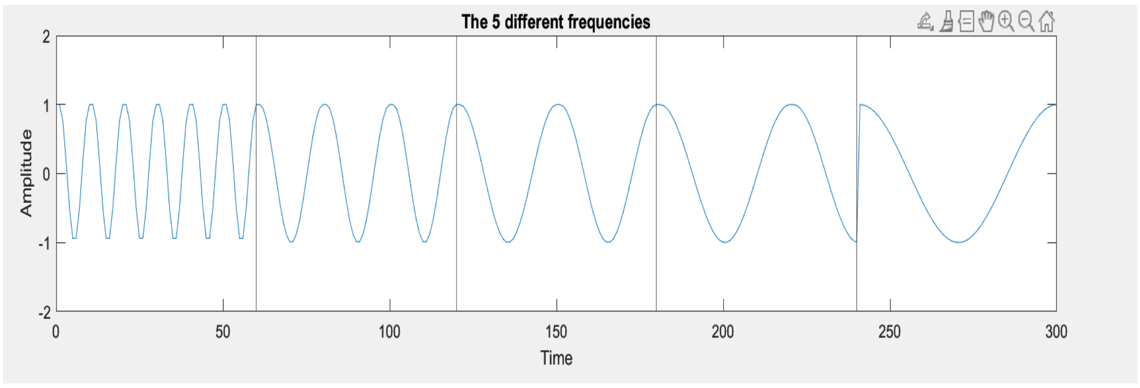
```
received_signal=fhss_signal./carrier_for_spreading;
received_bits=originalsignal./main_carrier;
```

Listing 5.7: Results

As mentioned, there is more than one solution. There are many ways to implement a code that will do the same thing. Thus, the student can play around with different implementations. The only this that they need to keep in mind. When multiplying or dividing the signal, the two variables, signal, and carrier, must be the same length. The must be a random hopping mechanism.

When running the script after the implementation is done as previously stated, plots will appear when they compare the original data vs. the received data if the original data is equal to the received data. Figures 5.3, 5.4, and 5.5 will appear.

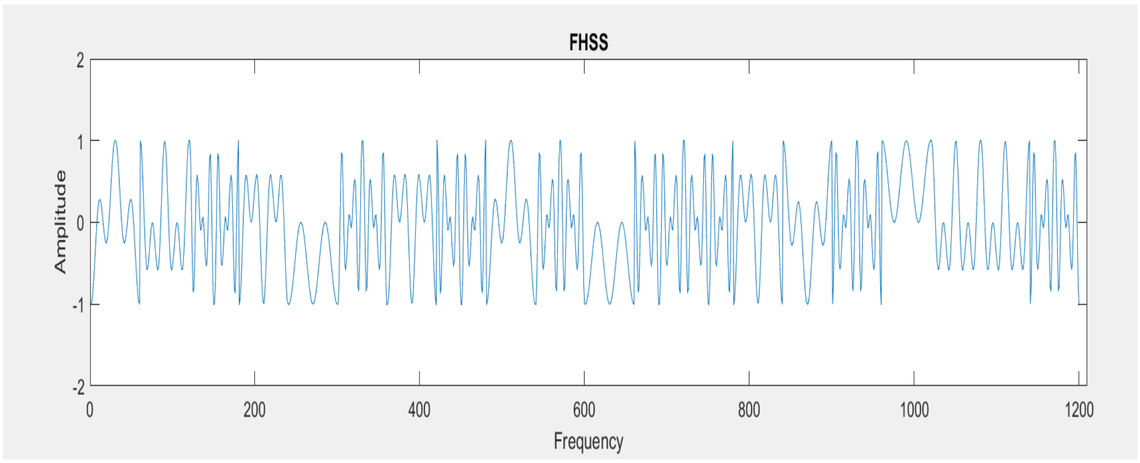
Figure 5.3: The 5 carrier frequencies



The vertical lines in Figure 5.3 represent each frequency carrier. The student will notice that each frequency carrier has a different frequency, as we implemented in the exercises. This plot represents what we have done, and the hopping mechanism will randomly choose these carriers.

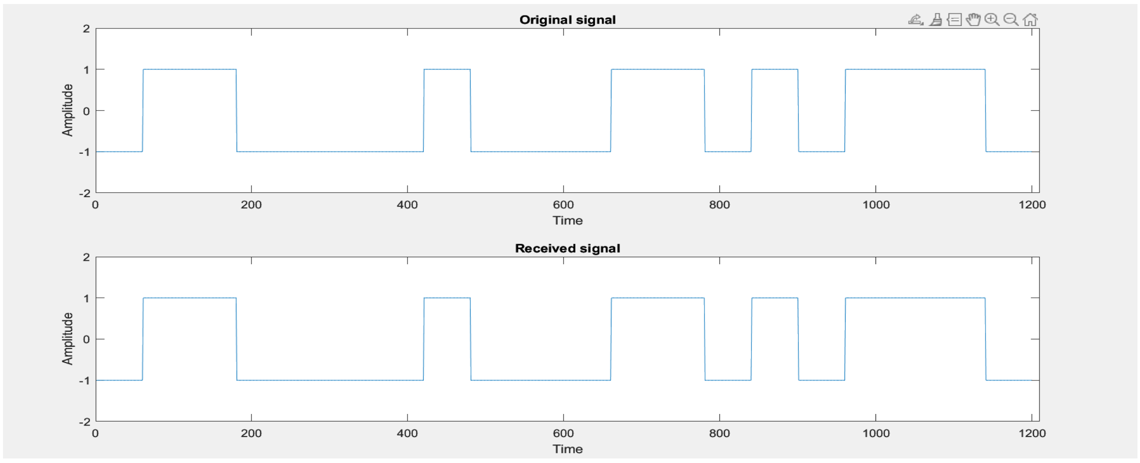
The student can see the transmitted FHSS signal in Figure 5.4. Overall, this plot represents the characteristic of FHSS because there are different frequencies.

Figure 5.4: FHSS signal



If the student accomplished the exercise, the original signal would be the same as the received signal, as shown in Figure 5.5.

Figure 5.5: Comparison of signals



The exercises were designed to give the student a good understanding of how the FHSS scheme works, which involves creating the frequency allocation and random hopping mechanism and setting up the receiver in MATLAB.

As stated in the description, this exercise uses a random Hopp. However, in reality, it is seen as random but determined through an algorithm that both the receiver and transceiver use through synchronization. We used a random switch case instead, which provided the different frequency carrier hops variable multiplied with the data to create the frequency allocation of the data. Then the data was divided into the same variable. Furthermore, this exercise also lacks the same as CDMA/DSSS, which is the modulation and interference that cannot result in accurate simulation. However, this exercise gives a good understanding of the functionalities of the FHSS scheme.

5.4 OFDM Evaluation

The subcarriers setup in the OFDM exercise could be more realistic to increase its informativeness. As mentioned in the exercise description, the subcarriers are not orthogonal and combine a long signal during parallel to serial conversion. Additionally, there was no interference of the signal implemented in the exercise, although a more realistic modulation was used compared to the other exercises. Despite these simplifications, the exercise demonstrates how the OFDM scheme works. The student should have gained a basic understanding of the OFDM scheme by observing the different parts of the exercise. While the exercise could have been more realistic, it still provided a valuable learning opportunity for the student.

Chapter 6

Conclusions

6.1 Conclusions

In conclusion, through this thesis, we have created exercises for the student to further their knowledge about wireless communication schemes. The schemes were first presented with a general description to achieve this. This knowledge gave us a good direction for constructing the exercises. Then by reading through related work and survey papers, we found the most optimal tools for the exercises. These tools were selected based on their ease of implementation of the schemes and their ability to efficiently showcase the characteristics and give the student an optimal learning experience.

NS-3 was the ideal tool for the simulation, as it allowed for the modification of different parameters to observe the corresponding outcome of this change. It included parameters such as enabling/disabling RTS/CTS for CSMA. We simulated the CSMA/CA and CSMA/CD schemes to showcase the hidden terminal problem and its solution. Through the comprehensive impact of changing parameters, the student will have gained a hands-on learning experience and can see how well the two schemes perform. Thus our main goal of this exercise was achieved, and this is a great exercise to see how two variations of a MAC scheme work.

We focused more on coding for the CDMA/DSSS exercises because it promotes a better practical learning experience when implementing and designing the scheme. Through this, the student could gain a more technical understanding of CDMA/DSS. We chose MATLAB because it is excellent in vector manipulation and ease of coding. The creation of this exercise was made straightforward by this. This exercise gave the student excellent knowledge about MLS and LFSR. By a generated MLS through LFSR and verifying the properties, the student will have gained practical knowledge of the theory. Furthermore, in the second part of the exercises, the student implemented the MLS sequence from the LFSR into a CDMA/DSSS MATLAB script. The student witnessed how important it is for the sender and receiver to have an equal MLS sequence. This exercise did not only teach the student the theory behind MLS and LFSR but also how it functions in the DSSS communication system, which was our main target.

Like the CDMA/DSSS exercise, the CDMA/FHSS exercise used MATLAB for the same reason. In addition, we allowed the student to be more creative with their solutions as long as they followed the main principles. The main focus of the exercise was to demonstrate how the FHSS scheme works by letting the student implement it. Since the student could be more creative, we encouraged a solution that ensured the received data corresponded with the transmitted data, which included the hopping mechanism and allocation of the frequencies to the different carriers and the receiver. By doing the exercise the student should now have a good knowledge of the main characteristics of FHSS.

Our last goal of this thesis was to implement a walk-through exercise. We selected the OFDM scheme for this as it is one of the more complex schemes with various procedures. Through our walk-through, we showed how the procedures work by showing plots of the signal and how it changes throughout the walk-through. Unfortunately, the exercise was more simplistic than we would have liked and did not truly achieve what we expected. Making a realistic real-time simulation was much more complex than anticipated. Therefore, we did not achieve our target with this exercise. However, it still manages to demonstrate the various procedures educationally but needs to be improved. Thus the student will still learn from this exercise, but our walk-through exercises were less realistic than we would have liked.

When starting this thesis, our goal was to give the student an understanding of wireless MAC schemes and their characteristics through a series of exercises. As stated, we did this by first presenting the various MAC schemes and finding the most optimal tool. The survey papers we went through for our related works gave us a sense of direction. However, these survey papers were scarce, and the majority did not contain enough information for our thesis.

Doing these exercises, we have promoted a comprehensive view of how efficiently the schemes performed under various scenarios, the key characteristics, and how some key function works in the various schemes. This information is valuable for the student that will work in this field since this can lead to the development of future schemes or improvements on existing ones.

6.2 Future Directions

After evaluating the results, we found that having a bit rate, like in the hidden terminal problem exercise, is valuable. For this reason, it demonstrates how efficiently a scheme performs under various situations. Not only can bit rate be implemented for other exercises, but also other impairments such as noise interference. However, as stated before, we wanted unique exercises that differ from one another. Which was the main focus, but being able to modify parameters will only increase the educational gain for the student.

As mentioned, the OFDM exercise did not achieve what we expected, leaving room for future work. By incorporating the fact that the subcarriers are orthogonal to each other, the OFDM exercise can be implemented more realistically, as well as including a channel model with noise interference. Additionally, it would have been beneficial to implement a bit rate to analyze the performance of OFDM in a noisy involvement.

Bibliography

[1] Gianfranco Nencioni. Data link layer media access control, 2022. source:Dat610-Wireless Communication pfd file.

[2] Moumita. Virtual channel sensing using csma/ca. URL https://www.tutorialspoint.com/virtual-channel-sensing-using-csma-ca. Accessed:06.05.2023.

[3] Margaret Rouse. Frequency hopping–code division multiple access. URL https://www.techopedia.com/definition/7169/frequency-hopping-code-division-multiple-access-fh-cdma. Accessed:06.02.2023.

[4] Inc. Keysight Technologies. Concepts of orthogonal frequency division multiplexing (ofdm) and 802.11 wlan, . URL https://rfmw.em.keysight.com/wireless/helpfiles/89600B/WebHelp/Subsystems/wlan-ofdm/content/ofdm_basicprinciplesoverview.htm. Accessed: 27.01.2023.

[5] Electronics Notes. Ofdm cyclic prefix, cp-ofdm. URL https://www.electronics-notes.com/articles/radio/multicarrier-modulation/ofdm-cyclic-prefix-cp.php. Accessed: 12.04.2023.

[6] Tri T. Ha. *Intersymbol interference and equalization*, page 473–521. Cambridge University Press, 2010. doi: 10.1017/CBO9780511778681.010.

[7] E. VarshaP. and P JagadeeshChandraA. Real time implementation of ofdm system on tms320c6713. 2015.

[8] Shreya Khisa and Sangman Moh. Performance simulation tools for mac protocols in the internet of things. 09 2020. doi: 10.1145/3426020.3426026.

[9] J. Sánchez-García, J.M. García-Campos, M. Arzamendia, D.G. Reina, S.L. Toral, and D. Gregor. A survey on unmanned aerial and aquatic vehicle multi-hop networks: Wireless communications, evaluation tools and applications. *Computer Communications*, 119:43–65, 2018. ISSN 0140-3664. doi: https://doi.org/10.1016/j.

comcom.2018.02.002. URL https://www.sciencedirect.com/science/article/pii/S0140366416304315.

[10] Chandresh Pratap Singh, O. P. Vyas, and Manoj Ku Tiwari. A survey of simulation in sensor networks. In *2008 International Conference on Computational Intelligence for Modelling Control & Automation*, pages 867–872, 2008. doi: 10.1109/CIMCA.2008.170.

[11] Harsh Sundani, Haoyue Li, Vijay Devabhaktuni, Mansoor Alam, and Prabir Bhattacharya. Wireless sensor network simulators a survey and comparisons. *International Journal of Computer Networks*, 2(5):249–265, 2011.

[12] Ronit Patel. Survey on network simulators. 10 2018.

[13] Network simulator 2 Projects. Ns2 manual. URL https://networksimulator2.com/ns2-manual/. Accessed 07.02.2023.

[14] nsnam. About. URL https://www.nsnam.org/about/. Accessed 07.02.2023.

[15] NS-2 Project. Glomosim network simulator. URL https://www.ns2project.com/glomosim-network-simulator/. Accessed:10.02.2023.

[16] OPNET PROJECTS TEAM. Opnet network simulator. URL https://opnetprojects.com/opnet-network-simulator/. Accessed:07.05.2023.

[17] OpenSim Ltd. Simulation with omnet++. URL https://omnetpp.org/documentation/simulation-with-omnet. Accessed:10.02.2023.

[18] Wikipedia. Scilab. URL https://en.wikipedia.org/wiki/Scilab. Accessed:07.05.2023.

[19] Inc. The MathWorks. Matlab. URL https://www.mathworks.com/products/matlab.html. Accessed:07.05.2023.

[20] Inc. Keysight Technologies. Qualnet network simulator, . URL https://www.keysight.com/us/en/assets/3122-1395/technical-overviews/QualNet-Network-Simulator.pdf. Accessed:07.05.2023.

[21] Jdsp. Jdsp. URL https://jdsp.dev/. Accessed:07.05.2023.

[22] Galaxy Technologies LLC. Getting started with gns3. URL https://docs.gns3.com/docs/. Accessed:09.02.2023.

[23] Pavel Boyko <boyko@iitp.ru>. wifi-hidden-terminal.cc. URL https://www.nsnam.org/docs/release/3.19/doxygen/wifi-hidden-terminal_8cc_source.html. Accessed:28.03.2023.