4G LTE Simulation Project Parts 2-5

Group:

New Comms Platform 4

Teacher:

Dr. Mark Mahon

Course:

CMPEN 462

Abstract

LTE was first introduced in 2004 by NTT Docomo, and was finalized as a wireless communications standard in 2008. The goal of LTE was to increase the network throughput with a new architecture which relied on newer DSP techniques to boost the capacity and speed that the network was capable of delivering compared to previous generation 3G technologies.

According to the LTE specifications, the network is capable of delivering up to 300 Mbps downlink, and 75 Mbps uplink. The transfer latency is 5 ms, along with carrier bandwidths scaling from 1.4 MHz to 20 MHz. The standard utilizes frequency division duplexing and time division duplexing. [3]

Introduction

We were tasked with creating a simulation of the LTE architecture, from upstream to downstream. Our simulation was scripted in MATLAB.

We broke the LTE structure into a 7-stage pipeline for the upstream, and 6-stage for the downstream, as shown in Figure 1. The bitstream was given to us as input, then the input was encrypted, run through QPSK, Resource Element Mapping, IFFT, OFDM, and given a Cyclic Prefix. The data was then transmitted to the receiver, who would remove the cyclic prefix, FFT, reverse QPSK, and finally decrypt the data.

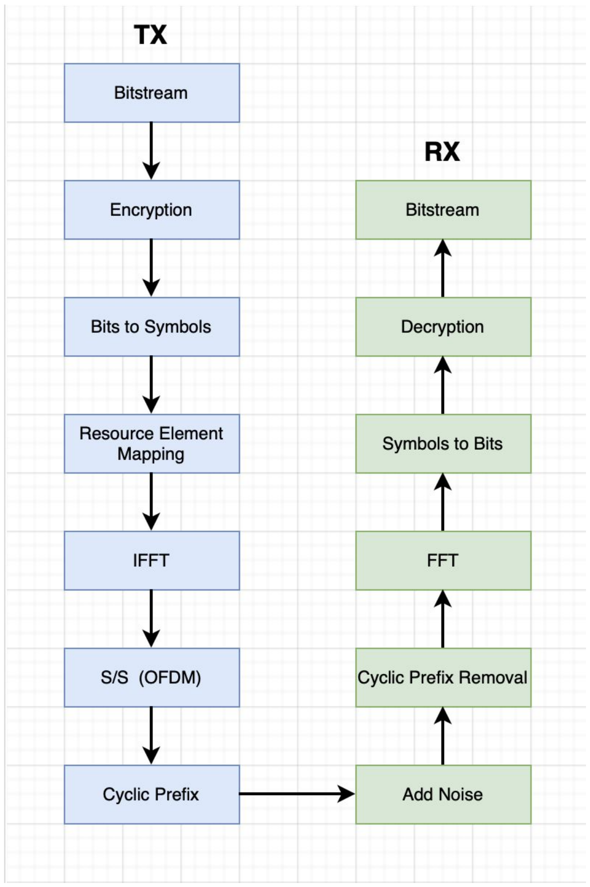

Figure 1. LTE Architecture

The following table (Table 1) includes the specifications we utilized in our implementation.

Our final Tx design is broken into four MATLAB function files. The four Tx function files in order are:

- CBCencrypt.m
- QPSK.m
- trans.m
- CP.m

Our final Rx design is broken into four MATLAB function files. The four Rx function files in order are:

- rmCP2.m
- ft.m
- sym2bit.m
- CBCdecrypt.m

The main5.m file first generates a random key and IV and calls all of the Tx functions and passes the output of one function as the input to the next one. It then calls noise.m which adds Gaussian White Noise to simulate the noise from wireless transmission. The output with noise is then passed into the Rx chain of functions in order and the final output is saved. After the Rx chain is done the bit error rate is calculated and printed in the command window.

System Bandwidth	10.24 MHz
Subcarrier Bandwidth	10 kHz
IF Frequency	100 MHz
Cyclic Prefix Length	6.84 μs
Max Transmit Power	100 μW
Input (binary data)	1 s
Number of Subcarriers	1024
Symbols per Second	10,000
Symbol Duration	0.1 ms
Noise (amplitude)	$\mu = 0$ $\sigma = 0.25$
Noise (phase)	$\mu = 0$ $\sigma = 22°$

Table 1. Specifications

Functional Blocks

Bitstream

The first stage of our pipeline was the input bitstream. This bitstream represents binary data to be sent over the 4G LTE communication standard. As 4G LTE is an IP-based service, the bitstream is actually composed of a stream of IP packets. For our project, we were given an MATLAB file titled Proj1InputData.mat to utilize as the input.

Encryption

Once the input stream is passed down from the upper layer of the protocol stack, the data must be encrypted. According to NIST LTE specification[2], there are 3 variations of cryptography algorithms utilized, with EPS Encryption Algorithms (EEA) and EPS Integrity Algorithms (EIA) being their basis. The three algorithms are titled EEA1 and EIA1 (based on SNOW 3G), EEA2 and EIA2 (based on AES), EEA3 and EIA3 (based on ZUC). Other algorithms such as PN sequences are also utilized for this stage. In our case we utilized CBC encryption. The encryption method does not provide end-to-end encryption, as the data is only encrypted when traveling through the radio waves, then once in the core-network, this encryption is removed. Therefore it is up to the user to implement a different form of application layer encryption such as SSL or SSH to ensure privacy of data.[3]

CBC (Figure 2) is a form of a block cipher, where a sequence of bits are encrypted as a block. CBC uses an initialization vector (IV) to XOR with the first block of plaintext to encrypt and hide any plaintext patterns. The chaining mechanism of CBC causes every stage of encryption to completely change the output ciphertext. CBC does have a disadvantage as one error bit can propagate error through all the chains in decryption, creating some error. [9]

The encryption is performed in the CBCencrypt.m function. This function takes the IV, key, and bitstream as inputs. The data is then grouped into blocks the same size as the key and IV. In our system that ended up being 1024 which was chosen because it is the number of subcarriers. The first group is then XORed with the IV and that value is then XORed with the key. The value of the IV is then replaced with the value of the XOR between the IV and plaintext group. The process is then repeated for all of the remaining blocks. After that all of the blocks are combined back into serial and you have your encrypted bitstream.

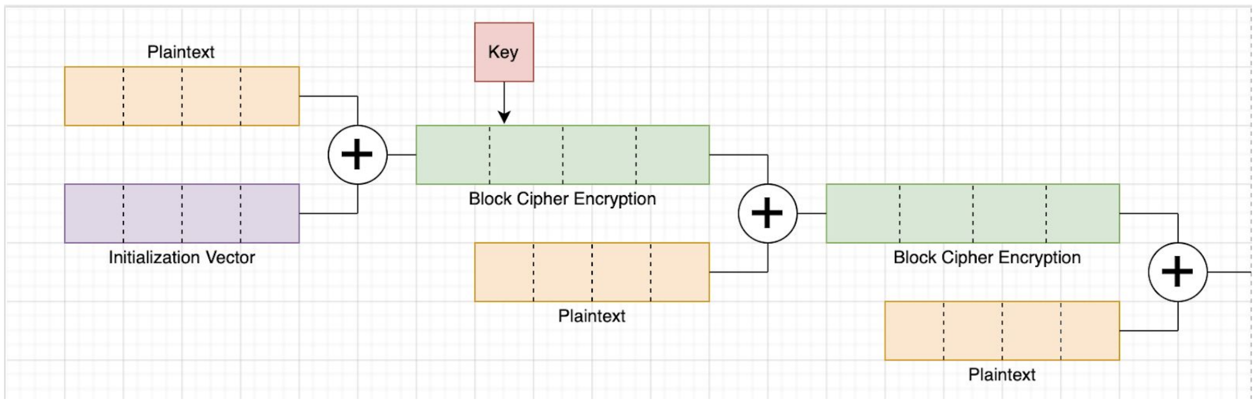

Figure 2. CBC Encryption

Bits to Symbols

When transmitting data over radio waves through a transmit antenna, we must think of the wave as a sinusoid (with time dependent phase, magnitude, and frequency components) as opposed to the literal bits we want to transmit. LTE specification[1] can utilize QPSK, QAM16, and QAM64 to make modifications to the sinusoid and map the bits to the wave. Quadrature Phase Shift Keying (QPSK) is a modulation scheme that involves shifting the sinusoid phase based on the binary input stream, taking in packets of 2 bits and mapping them to one symbol (the state of the carrier). We call the rate at which the carrier changes states the symbol rate. The LTE spec QPSK diagram is shown in Figure 3. Quadrature Amplitude Modulation (QAM) is another form of digital signal modulation that uses a combination of signal phase and magnitude to determine the symbol output. QAM offers higher data rates than QPSK, and can map 16, 64, or 256 symbols, however LTE spec only supports QAM16 and QAM64. In the case of QAM64, each symbol can encode 6 bits ($2^6 = 64$), and QAM16 can encode 4 bits per symbol ($2^4 = 16$).

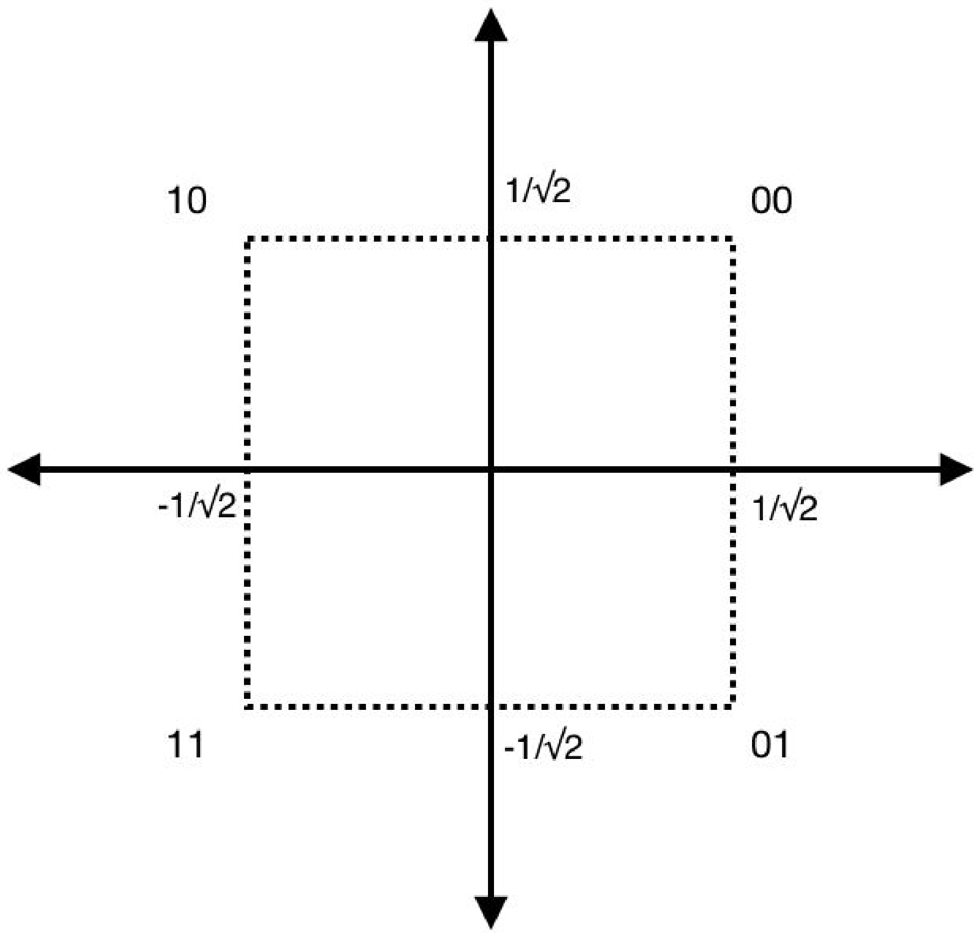

Figure 3. QPSK LTE Spec. Constellation Diagram

The bits to symbols conversion in our system was done in QPSK.m. To do this we first converted our data from an array to a 2 row matrix which split the data into symbols with each column representing a two byte symbol needed for QPSK. We then looped through this two row matrix column by column and assigned the real and imaginary parts of each complex element. The two parts of the complex element were put together and added as the next element of the output array.

Resource Element Mapping

According to the LTE spec[4], the number of subcarriers ranges from 128 to 2048, depending on the desired channel bandwidth. The subcarrier spacing is 15kHz. Resource element mapping allows us to have multiple users utilizing the network simultaneously.

The transmission of data can be scheduled by resource blocks (RBs), which are composed of 12 subcarriers with 180kHz of bandwidth per time slot. A resource element (RE) is an atomic unit within an RB, and is composed of one OFDM subcarrier. Figure 4 shows how an RB and RE exists in context to each other, within LTE spec.

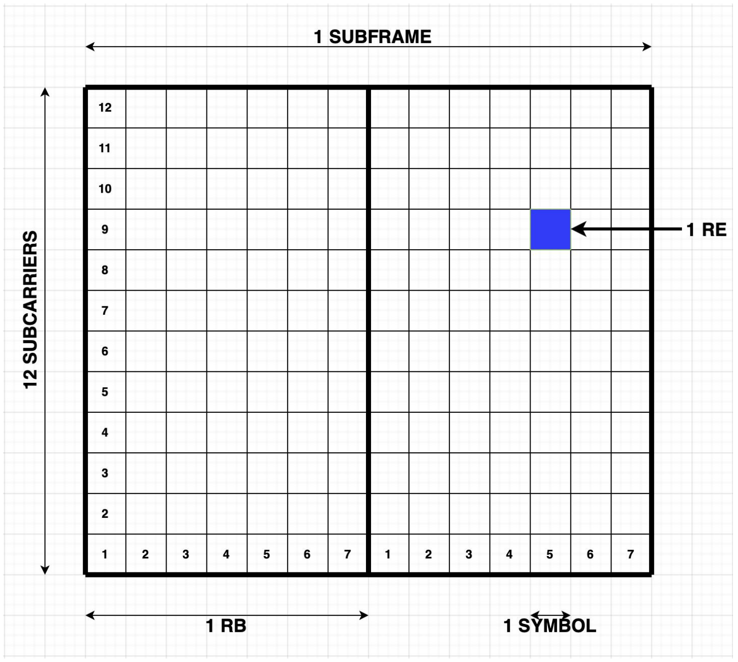

Figure 4. RE and RB Diagram

In our implementation this was done in the same function as the IFFT which will be described in more detail in the next section.

IFFT

The Inverse Fast Fourier Transform (IFFT) is used to convert signals from the frequency domain to the time domain, as the signal must be represented in the time domain for transmission.

In our system the IFFT along with the Resource Element Mapping was performed in the trans.m file. For resource element mapping all that's needed in our system is to break the data vector into groups of 1024 for the IFFT. This was achieved by reshaping the data into a matrix with each row being one group of 1024 symbols. We then looped through those rows and performed an IFFT on each row.

OFDM

Orthogonal Frequency Division Multiplexing (OFDM) is a multi-carrier modulation method utilizing frequency division multiplexing (FDM). It allows multiple users to transmit and receive data without having to worry about wasting bandwidth, or intercarrier interference (ICI) between subcarriers. OFDM is used for the downlink, over carriers of size 180 kHz. Each user is given a certain number of RBs for their data transmission, and the number of RBs assigned to a particular user depends on LTE scheduling mechanisms to provide ideal performance in varied radio environments.[5]

This did not need to be implemented into our project because we only had one "user", the one second binary stream.

Cyclic Prefix

The Cyclic Prefix (CP) is used in LTE communication as a guard between symbols, while preserving the orthogonality of the subcarriers. The CP is a cyclic set of symbols from the end of each symbol set, and added to the front of the transmitted symbol to eliminate intersymbol interference (ISI). The receiver can then identify where each symbol ends, and determine what symbol sets correlate to what data. The normal CP has a duration of 5.2 μs for the first prefix, and then 4.7 μs for the remaining symbols.[6]

The cyclic prefix addition was performed in the CP.m function. We then looped through these groups and added a copy of the last 70 symbols (our calculated prefix length) to the beginning of each group. This group was then added to a new vector for the output.

Noise

Once data is transmitted from the Tx stage, our simulation needed to account for noise that would have interfered with data transmission. This would test the robustness of our simulation, and if it could handle real world interference. We generated a Gaussian White Noise distribution to add on to the output at the Tx stage. Gaussian White Noise was generated for amplitude as N(0, 0.25), and amplitude N(0, 22°). The Gaussian White Noise distribution has a flat power spectral density, and is commonly used to model random processes.[8]

The addition of noise was performed in the noise.m file. We first created two arrays (one for amplitude and one for phase) containing random values with each the same size of the input data. The two arrays were then combined into one array of complex numbers. This array was then run through an ifft of size 1024 so they were in the same domain as the data. After the ifft the noise was added to the data.

Cyclic Prefix Removal

The receiver uses the cyclic prefix to identify where each symbol ends and what symbol sets correlate to what data, then strips the prefix and passes it onto the FFT stage.

The Cyclic Prefix Removal was performed in the rmCP2.m function. This function does exactly the opposite of CP.m. It loops through the data removing each of the 70 symbols of the cyclic prefix and puts the data into the serial output array.

FFT

The Fast Fourier Transform (FFT) is used to convert signals from the time domain to the frequency domain, as we must now convert the signal into a binary bitstream.

The FFT was performed in the ft.m function. This function groups the data into chunks of size 1024 and performs an FFT on each chunk and puts them back together in serial at the end.

Symbols to Bits

At this stage, we have the QPSK symbols in the frequency domain. Since our signal had added noise from transmission, it was necessary to check for symbol proximity in the real-imaginary plane quadrants. It seemed that the noise interfered with the QPSK symbols, however the majority of the symbols were still within the same quadrant as their constellation diagram mapping.

The Symbol to Bit translation was performed in the sym2bit.m function. This function first splits the input symbols into real and imaginary parts. It then checks which quadrant the point falls under using the mapping shown above in Figure 3. The corresponding two bit value is then added to the output based on the quadrant it is in.

Decryption

CBC decryption is achieved by grouping data into blocks and then XORed through the key, and repeating for as many iterations the original CBC encryption chains were run. The issue with CBC is that a single bit error can propagate through the system and cause a lot of data to be decrypted incorrectly. In our simulation, we were able to keep the error rate low, however if the error rate was too high, one could implement ECC. [9]

The decryption is performed in the CBCdecrypt.m function. This function takes the key, IV, and encrypted data as inputs. The data is then grouped into blocks the same size as the key and IV. In our system that ended up being 1024 which was chosen because it is the number of subcarriers. The first group is then XORed with the key and that value is then XORed with the IV. The value of the IV is then replaced with the value of the XOR between the key and group. The process is then repeated for all of the remaining blocks. After that all of the blocks are combined back into serial and you are back to your initial bitstream.

Testing

The testing of parts 2-5 was done in a seperate test.m file. For parts 2-4 the test data from part 1 could be used to test the functionality because these parts were just doing the Rx side of the Tx performed in part 1. We tested by taking the .mat file from Canvas that corresponded to the step in the Rx pipeline we were working on and compared the output to the .mat file that corresponded to what the output should equal. To test part 5 we first created a small sample with a 12 bit input and a 3 but key and IV. We worked out what the values should be for the encryption and decryption and compared those against the output of our functions. After we got that working we created a random array of bits the same length as the actual input would be and then ran just the encryption and decryption functions back to back to verify that it worked for a larger input. We then added the rest of the stages and tested to see if our error rate was in an acceptable range when the noise was added. To find our error rate we looped through the input and output data and counted the number of bit errors. This number was divided by the total number of bits to get our error rate.

Problems

We only faced one major issue during parts 2-5. At first we were not running the noise we generated through an IFFT so the noise values were in the frequency domain and were being added to the data which was in the time domain. This meant that the noise was very large compared to the data and ended up giving us an error rate of about 50%. After we realized the noise had to go through an IFFT the error rate dropped to about 0.06% for out encryption and 0.035% for the PN encryption.

Results and Functionality

The second part of the project was Cyclic Prefix removal. When compared to the Part 1 .mat files we had no margin of error. The third part of the project was performing an FFT. When this section was tested using part 1 test data we had a margin of error of about 10^18 which we found acceptable. The fourth part of the project was converting symbols to bits including noise. We found our margin of error to be 0.0307% which is less than the 0.035% benchmark given. The last part was adding a new form of encryption which we chose to be CBC. When we tested this function we found our margin of error to be about 0.0609%. This was higher than part 4 but we think this is due to the errors propagating through the CBC decryption.

Project Part	Functionality	Margin of Error
Part 2	Remove Cyclic Prefix	none
Part 3	Perform FFT	~10^{-18}
Part 4	Symbols to Bits	0.0307%
Part 5	CBC	0.0609%

Table 2. Results

Work Performed and Roles

Task	Completed by
MATLAB Simulations	Nikolas Collina, Praharsh Verma
Document Writeup (MATLAB Parts)	Nikolas Collina
Document Writeup (Theory Parts)	Praharsh Verma

Table 3. Worked Performed & Roles

For the MATLAB scripts, Nikolas and Praharsh worked together over FaceTime and the screen sharing capability on MacOS to allow us to collaborate on the code in real-time, debugging and ensuring quality of output.

For the document writeup, Nikolas and Praharsh worked together using Google Docs and FaceTime to collaborate on the document in real-time. Nikolas wrote up the sections pertaining to how the MATLAB scripts were implemented in each section, as well as the Testing, Problems, and Results & Functionality sections. Praharsh wrote up the sections pertaining to the theory on how LTE architecture functions, as well as the Introduction and Abstract.

Bibliography

[1] StackPath. [Online]. Available:
https://www.electronicdesign.com/technologies/4g/article/21796272/an-introduction-to-lteadvanced-the-real-4g#âModulationâ.

[2] J. Cichonski, J. M. Franklin, and M. Bartock, "Guide to LTE security," 2017.

[3] "The MobileBroadband Standard," LTE. [Online]. Available:
https://www.3gpp.org/technologies/keywords-acronyms/98-lte.

[4] "LTE in a Nutshell: The Physical Layer." [Online]. Available:
https://home.zhaw.ch/kunr/NTM1/literatur/LTE in a Nutshell - Physical Layer.pdf.

[5] "LTE OFDM Technology," Tutorialspoint. [Online]. Available:
https://www.tutorialspoint.com/lte/lte_ofdm_technology.htm. [Accessed: 01-Apr-2020].

[6] L. Z. Pedrini, "What is CP (Cyclic Prefix) in LTE?," telecomHall Forum, 10-Nov-2019. [Online]. Available: https://www.telecomhall.net/t/what-is-cp-cyclic-prefix-in-lte/6369. [Accessed: 01-Apr-2020].

[7] "The Benefits of an Intermediate Frequency in RF Systems: Selected Topics: Electronics Textbook," All About Circuits. [Online]. Available:
https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/selected-topics/the-benefits-of-an-intermediate-frequency-in-rf-systems/. [Accessed: 01-Apr-2020].

[8] Appendix II: Gaussian White Noise. (2012, May 29). Retrieved May 01, 2020, from
https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780471679370.app2

[9] Rouse, M. (2007, June 04). What is cipher block chaining (CBC)? - Definition from WhatIs.com. Retrieved May 01, 2020, from
https://searchsecurity.techtarget.com/definition/cipher-block-chaining