Advanced Topics in Computer Networks

Capture & Decode LoRa Signals

Iliadis Ilias 2523 - ililiadis@uth.gr
Lamprinos Isidoros 2551 - ilamprinos@uth.gr

February 2022

1 Introduction

LoRa is a proprietary low-power wide-area network modulation technique with low-power consumption and it's used for long-range transmissions. By doing that LoRa devices have revolutionized the Internet of Things(IoT). LoRa is flexible for rural or indoor use cases in smart agriculture, smart cities, industrial IoT (IIoT), smart environment, smart homes and buildings, smart utilities and metering, and smart supply chain and logistics. Theoretically, we can expect the following ranges depending on different factors (indoor/outdoor gateways), as well as the environment:

- 2-3 Km wide coverage outdoor in urban areas;
- 5-7 Km in rural areas;
- and sometimes very long-range like 702 Km.

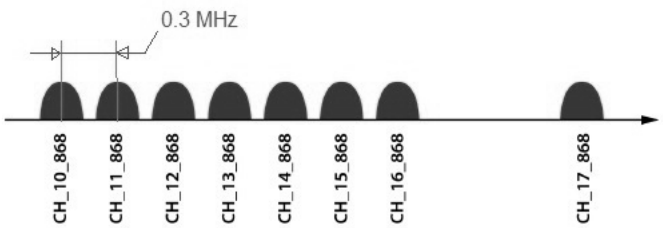
LoRa uses license-free sub-gigahertz radio frequency bands like:

- Europe: 433 MHz and 868 MHz;
- Australia and North America: 915 MHz;
- Asia: 923 MHz.

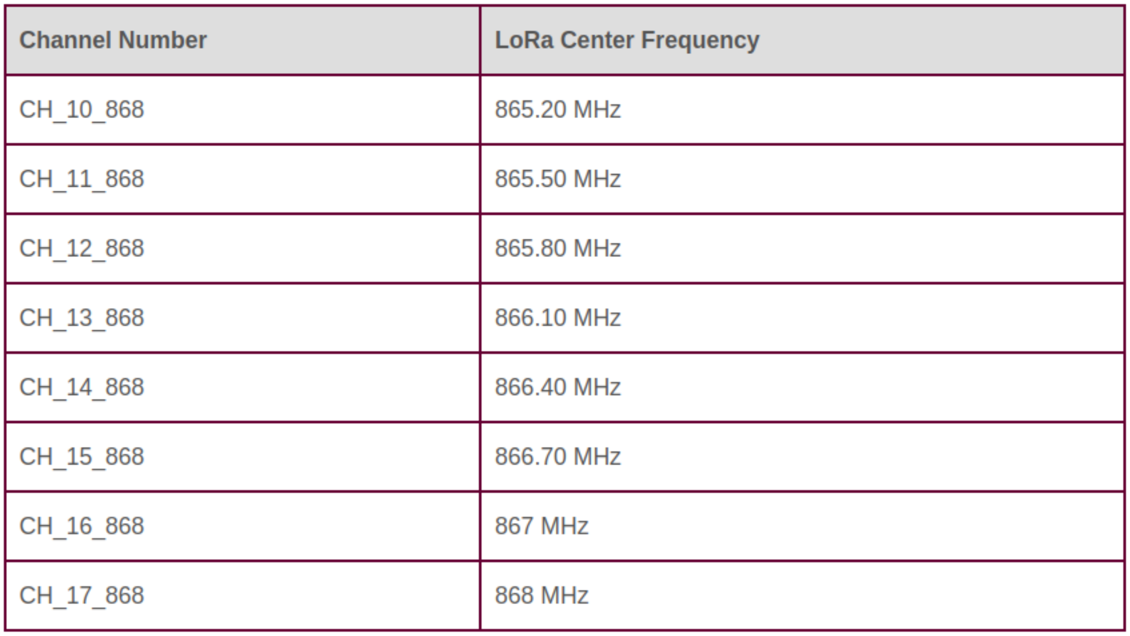
In order to capture the desired signals we need to know the frequencies utilized by our LoRa transmitter. The project we will describe below was implemented using an Adalm-Pluto SDR device.

2 Identify frequencies

The band defined in EU for LoRaWAN Network is form 863 to 870 MHz. The channel number and it's respective channel frequency are shown in the table below.All the 8 channels are separated by 0.3 MHz with respect to the adjacent channels.

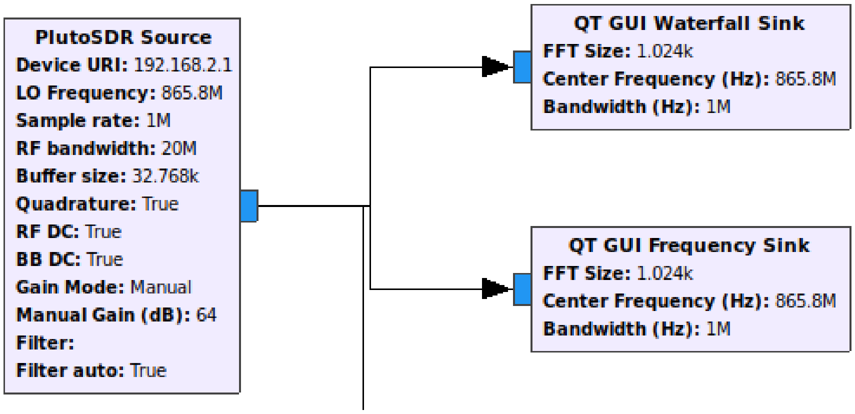


863 to 870 MHz Channels



Channel Number	LoRa Center Frequency
CH_10_868	865.20 MHz
CH_11_868	865.50 MHz
CH_12_868	865.80 MHz
CH_13_868	866.10 MHz
CH_14_868	866.40 MHz
CH_15_868	866.70 MHz
CH_16_868	867 MHz
CH_17_868	868 MHz

To detect the LoRa center frequency used by our transmitter, we used a waterfall and frequency GUI to have a representation of the signal across the frequency range, but also an FFT block to find the different frequency components of a signal holding it's max value when the signal is triggered. Here is the gnuradio-companion schema that implements the above.



PlutoSDR Source
Device URI: 192.168.2.1
LO Frequency: 865.8M
Sample rate: 1M
RF bandwidth: 20M
Buffer size: 32.768k
Quadrature: True
RF DC: True
BB DC: True
Gain Mode: Manual
Manual Gain (dB): 64
Filter:
Filter auto: True

QT GUI Waterfall Sink
FFT Size: 1.024k
Center Frequency (Hz): 865.8M
Bandwidth (Hz): 1M

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 865.8M
Bandwidth (Hz): 1M

Figure 1: GNU Radio schema to find a targeted signal

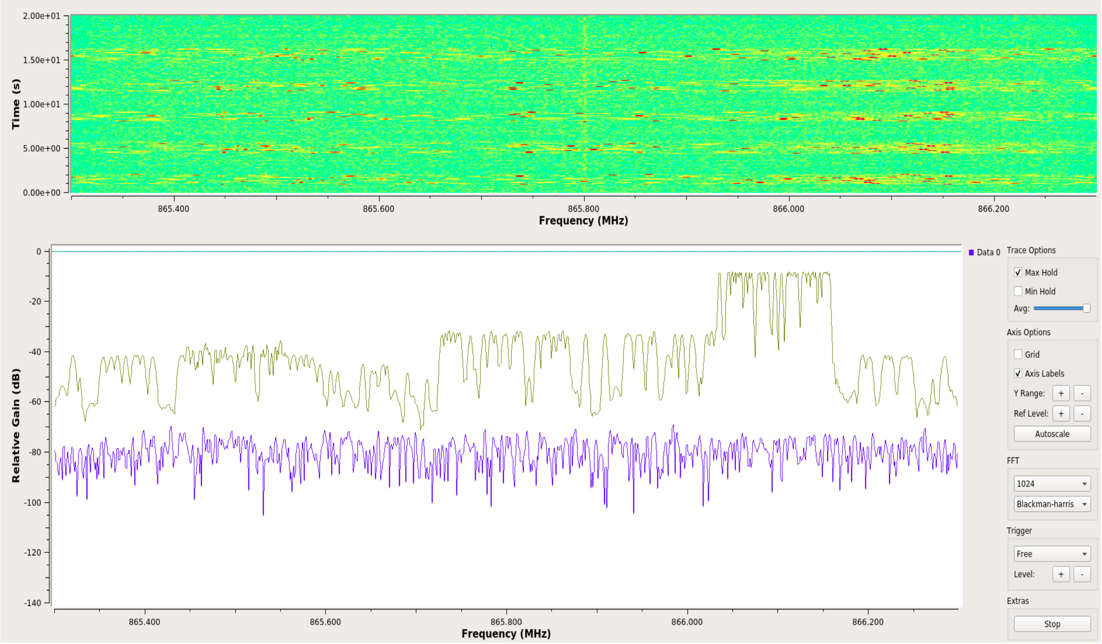


Figure 2: LoRa signal triggered in GNU Radio FFT and waterfall display

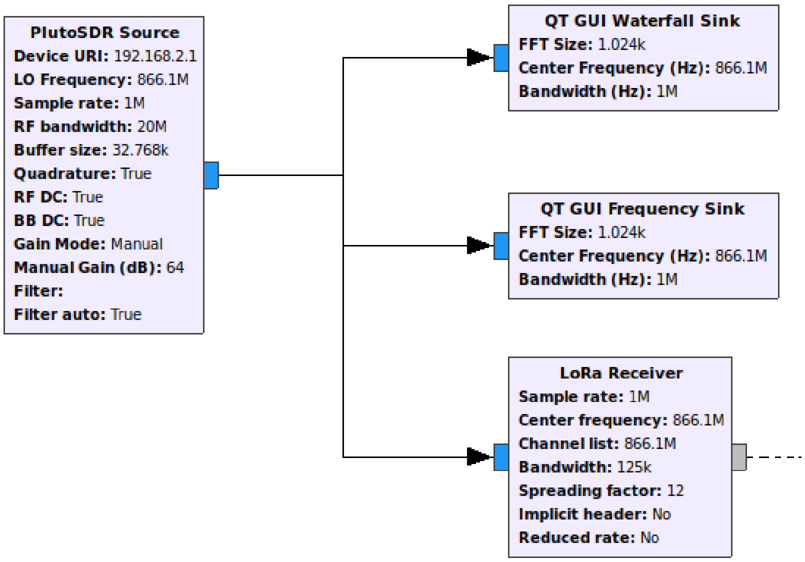
As shown in Figure 2 we can clearly see that the transmission is set in channel number CH_13_868 with center frequency: 866.1 MHz. We've set center frequency equal to the neighbor channel (CH_12_868) in order to detect a transmission frequency and the transmission we detected was the one mentioned above.

3 Demodulating and decoding captured signals

At this stage, we know the channel frequency used by the transmitted signal, which is 866.1 MHz. In order to proceed to the demodulation and decoding of the incoming LoRa signals, we took advantage of the repository [1], which provides a collection of GNU Radio blocks for receiving LoRa modulated radio messages using a Software Defined Radio.

3.1 Demodulation

In order to demodulate real-time incoming LoRa signals, we added a LoRa Receiver provided by the gr-lora repository to our schema:



PlutoSDR Source
Device URI: 192.168.2.1
LO Frequency: 866.1M
Sample rate: 1M
RF bandwidth: 20M
Buffer size: 32.768k
Quadrature: True
RF DC: True
BB DC: True
Gain Mode: Manual
Manual Gain (dB): 64
Filter:
Filter auto: True

QT GUI Waterfall Sink
FFT Size: 1.024k
Center Frequency (Hz): 866.1M
Bandwidth (Hz): 1M

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 866.1M
Bandwidth (Hz): 1M

LoRa Receiver
Sample rate: 1M
Center frequency: 866.1M
Channel list: 866.1M
Bandwidth: 125k
Spreading factor: 12
Implicit header: No
Reduced rate: No

Figure 3: Adding gr-lora to the schema

Before we start analyzing our implementation for signal demodulation, the three modulation parameters must be introduced:

3.1.1 Spreading Factor (SF)

LoRa processing gain is introduced in the RF channel by multiplying the data signal with a spreading code or chip sequence (carrier). By increasing the chip rate, we increase the frequency components of the total signal spectrum. In other words, the energy of the total signal is now spread over a wider range of frequencies, allowing the receiver to discern a signal with a lower signal-to-noise ratio (SNR). Also, this code sequence multiplication buys you a higher RF link budget, so you can transmit over a longer range. When the transmitted signal arrives at the RF receiver, it is multiplied with an identical copy of the spreading code, resulting in a replica of the original signal. The amount of spreading code applied to the original data signal is called **spreading factor (SF)**. LoRa modulation has a total of six spreading factors (SF7 to SF12). The larger the spreading factor used, the farther the signal will be able to travel. For example, an end device located close to a gateway should transmit data at a low spreading factor, since very little link budget is needed. However, an end device located several miles from a gateway will need to transmit with a much higher spreading factor. Importantly, the LoRa modulation spreading factors are inherently orthogonal. This means that signals modulated with different spreading factors and transmitted on the same frequency channel at the same time do not interfere with each other. Instead, signals at different spreading factors simply appear to be noise to each other.

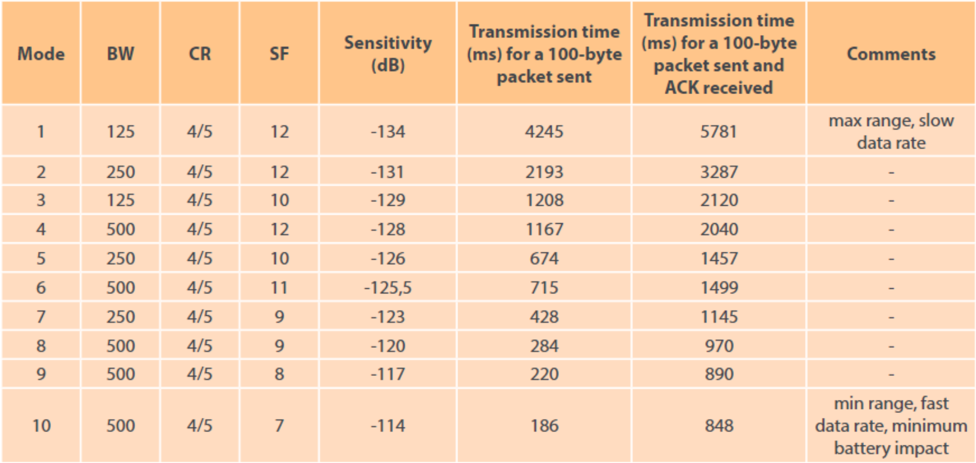
3.1.2 Coding Rate (CR)

Forward error correction (FEC) techniques are used in LoRa to further increase the receiver sensitivity. Code rate defines the amount of FEC. LoRa offers CR values between 0 to 4, where CR = 0 means no FEC. LoRa uses code rates of 4/5, 2/3, 4/7 and 1/2. Which means, if the code rate is denoted as k/n, where k represent useful information, and encoder generates n number of output bits, then n-k will be the redundant bits. The redundancy allows the receiver to detect and often to correct errors in the message, but it also decreases the effective data rate. In our case, CR is equal to 0.

3.1.3 Bandwidth (BW)

BW is the width of frequencies in the transmission band. Higher BW gives a higher data rate (thus shorter time on air), but a lower sensitivity (because of integration of additional noise). A lower BW gives a higher sensitivity, but a lower data rate. LoRa provides three scalable BW settings of 125kHz, 250kHz and 500kHz. Transmitter sends the spreaded data at a chip rate equal to the system bandwidth in chips per-second-per-Hertz. So a LoRa bandwidth of 125 kHz corresponds to a chip rate of 125 kcps.

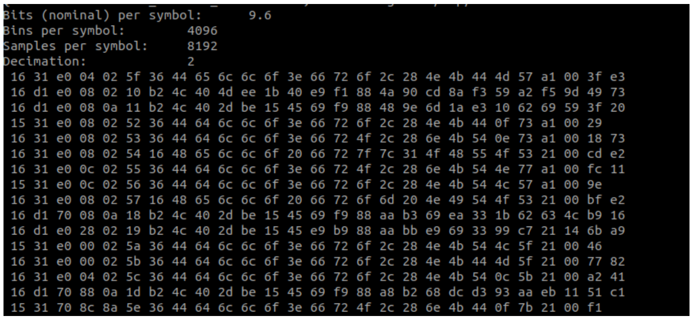
Because our LoRa transmitter changes the SF and BW values periodically, in order to find what kid of values it could take, we changed the corresponding values from the LoRa receiver block based on the values featured by the table below:



Mode	BW	CR	SF	Sensitivity (dB)	Transmission time (ms) for a 100-byte packet sent	Transmission time (ms) for a 100-byte packet sent and ACK received	Comments
1	125	4/5	12	-134	4245	5781	max range, slow data rate
2	250	4/5	12	-131	2193	3287	-
3	125	4/5	10	-129	1208	2120	-
4	500	4/5	12	-128	1167	2040	-
5	250	4/5	10	-126	674	1457	-
6	500	4/5	11	-125,5	715	1499	-
7	250	4/5	9	-123	428	1145	-
8	500	4/5	9	-120	284	970	-
9	500	4/5	8	-117	220	890	-
10	500	4/5	7	-114	186	848	min range, fast data rate, minimum battery impact

Figure 4: LoRa modulation characteristics

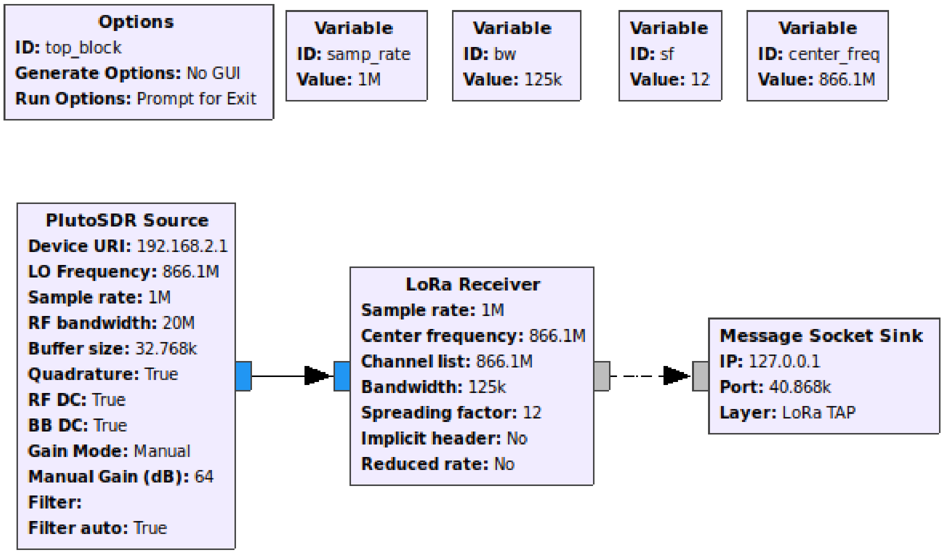
Trying different combinations from the above configurations we were finally able to see packets in the GNU Radio console by tuning in the same modulation as the transmitter. The image below depicts an execution of our .grc file by changing the receiver to SF12BW500 configuration:



```
Bits (nominal) per symbol:      9.6
Bins per symbol:        4096
Samples per symbol:     8192
Decimation:             2
 16 31 e0 04 02 5f 36 44 65 6c 6c 6f 3e 66 72 6f 2c 28 4e 4b 44 4d 57 a1 00 3f e3
 16 d1 e0 08 02 10 b2 4c 40 4d ee 1b 40 e9 f1 88 4a 90 cd 8a f3 59 a2 f5 9d 49 73
 16 d1 e0 08 0a 11 b2 4c 40 2d be 15 45 69 f9 88 48 9e 6d 1a e3 10 62 69 59 3f 20
 15 31 e0 08 02 52 36 44 64 6c 6c 6f 3e 66 72 6f 2c 28 4e 4b 44 0f 73 a1 00 29
 16 31 e0 08 02 53 36 44 64 6c 6c 6f 3e 66 72 4f 2c 28 6e 4b 54 0e 73 a1 00 18 73
 16 31 e0 08 02 54 16 48 65 6c 6c 6f 20 66 72 7f 7c 31 4f 48 55 4f 53 21 00 cd e2
 16 31 e0 0c 02 55 36 44 64 6c 6c 6f 3e 66 72 4f 2c 28 6e 4b 54 4e 77 a1 00 fc 11
 15 31 e0 0c 02 56 36 44 64 6c 6c 6f 3e 66 72 6f 2c 28 4e 4b 54 4c 57 a1 00 9e
 16 31 e0 08 02 57 16 48 65 6c 6c 6f 20 66 72 6f 6d 20 4e 49 54 4f 53 21 00 bf e2
 16 d1 70 08 0a 18 b2 4c 40 2d be 15 45 69 f9 88 aa b3 69 ea 33 1b 62 63 4c b9 16
 16 d1 e0 28 02 19 b2 4c 40 2d be 15 45 e9 b9 88 aa bb e9 69 33 99 c7 21 14 6b a9
 15 31 e0 00 02 5a 36 44 64 6c 6c 6f 3e 66 72 6f 2c 28 4e 4b 54 4c 5f 21 00 46
 16 31 e0 00 02 5b 36 44 64 6c 6c 6f 3e 66 72 6f 2c 28 4e 4b 44 4d 5f 21 00 77 82
 16 31 e0 04 02 5c 36 44 64 6c 6c 6f 3e 66 72 6f 2c 28 4e 4b 54 0c 5b 21 00 a2 41
 16 d1 70 88 0a 1d b2 4c 40 2d be 15 45 69 f9 88 a8 b2 68 dc d3 93 aa eb 11 51 c1
 15 31 70 8c 8a 5e 36 44 64 6c 6c 6f 3e 66 72 4f 2c 28 6e 4b 44 0f 7b 21 00 f1
```

Dynamic LoRa Transmission modes

Until now we were able to capture incoming LoRa signals by changing manually the two transmission mode parameters (sf,bw).To make our project more efficient, we altered the python file produced by the GNU Radio schema shown above in order to produce an dynamic algorithm that changes the transmission mode every n seconds in case no traffic is detected during execution until we find the mode we are on. The initial python file was produced by the following GNU Radio schema:



Options
ID: top_block
Generate Options: No GUI
Run Options: Prompt for Exit

Variable
ID: samp_rate
Value: 1M

Variable
ID: bw
Value: 125k

Variable
ID: sf
Value: 12

Variable
ID: center_freq
Value: 866.1M

PlutoSDR Source
Device URI: 192.168.2.1
LO Frequency: 866.1M
Sample rate: 1M
RF bandwidth: 20M
Buffer size: 32.768k
Quadrature: True
RF DC: True
BB DC: True
Gain Mode: Manual
Manual Gain (dB): 64
Filter:
Filter auto: True

LoRa Receiver
Sample rate: 1M
Center frequency: 866.1M
Channel list: 866.1M
Bandwidth: 125k
Spreading factor: 12
Implicit header: No
Reduced rate: No

Message Socket Sink
IP: 127.0.0.1
Port: 40.868k
Layer: LoRa TAP

The python file that includes both the depicted flowgraph and the dynamic algorithm is named **lora_dynamic.py**. To process the received hexadecimal LoRa data further , we have included a Message Socket Sink block in the Gnuradio schema to forward incoming messages to port 40868 over UDP . In order to process these transmitted packets over UDP we used the class LoRaUDPServer(), which creates a Datagram UDP socket and binds it to the loopback address and port 40868. If any data is received it's saved in an array initialized before the connection .After the connection has expired we check if that array has any data in it. In case no data is captured we assume the transmitter is on a different mode and we initialize a new flowgraph with the next mode's specifications (SF,BW). In case data has occurred we keep the current mode's parameters and we move on to the message decoding.

3.2 Decoding

The packets we receive from this connection are shown on the console. The problem is that they are hexadecimal. In order to understand the message we need to convert them to string using the functions binascii.hexlify() and binascii.unhexlify() on the received data. After that, we are able to see the message the transmitter is sending us. Using the UDP socket format we can also see the Source IP address on the packet. To do that we use function recvfrom() which returns an array with the Source IP of the packet it received on the first element and the data on the second. This is also how we capture the data of the packet.
Now the final step of our project is to use delimiters to split the received data string into it's respective substings and print these new strings. Delimiters are usually included in the payload in order to inform the receiver about the quantity of the information included in this specific payload by separating them from each other. Such information could be temperature measured by sensor, remaining battery life etc. The delimiters we used are the symbols: '/', '\', space (' '), line change ('\n') and comma (',').

4 Experiment Results

While observing the dynamic algorithm setting different transmission modes, we detected some malfunctions of the Pluto-SDR device. More specifically we noticed that the packet that includes the message "Hello from NITOS!" sent by our transmitter is often corrupted on modes 8,9 and 10 which have bandwidth equal to 500kHz. Moreover in mode 6 we seldom observe any transmission, but when we do the packets are not corrupted. Friendly modes for our SDR device are those with bandwidth equal to 250kHz with the exception of mode 3 which has bandwidth equal to 125kHz. Finally, in mode 1 in order to capture packets the receiver must be approximately 1.5 meter away from the transmitter.

5 Future Work

The next step is the development of a LoRa transmitter using our Pluto-SDR device. The implementation is based on the repository [2], which is inspired by the repository of our project.

References

[1] GNU Radio blocks for receiving LoRa modulated radio messages using SDR https://github.com/rpp0/gr-lora/tree/gr3.7

[2] GNU Radio software-defined radio (SDR) implementation of a LoRa transceiver https://github.com/tapparelj/gr-lora_sdr/tree/unmaintained/gr-3.7

[3] Testing LoRa with SDR and some handy tools https://penthertz.com/blog/testing-LoRa-with-SDR-and-handy-tools.html

[4] https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/