Advanced Topics in Computer Networks
Receive and Transmit LoRa packets

Koutsoukis Nikolaos

February 9, 2023

Contents

1 Introduction of LoRa

1.1 Definition

LoRa is a type of low-power radio technology used for Long Range (LoRa) communication, allowing for long-distance transmission of data over a long range of distances [1]. It is often used in the Internet of Things (IoT) to enable communication between different types of devices, such as sensors, actuators and controllers. It is based on a spread spectrum modulation technique that is specifically designed to enable long-range communication with low power consumption.

1.2 Frequencies Bands

LoRa works in frequencies ranging from 137 MHz to 1020 MHz, depending on the region. In Europe and the US, LoRa networks typically operate in the 868 MHz frequency band, while in Asia-Pacific the 915 MHz frequency band is used. LoRa networks also support the 433 MHz frequency band, which is used in some parts of Europe. More specific in Europe, the 868 MHz frequency band is divided into 8 channels, with each channel having a bandwidth of 300 kHz.

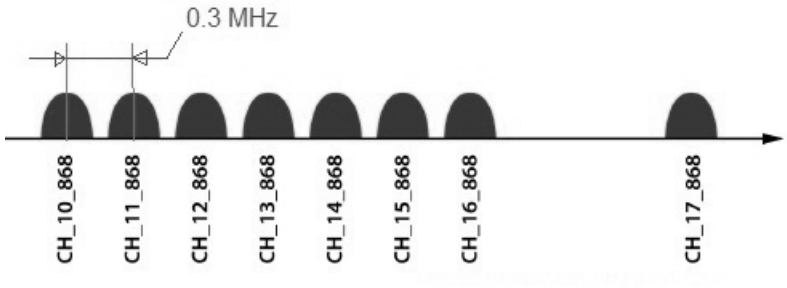

Figure 1: 868 MHz frequency band

1.3 Basic Parameters

The LoRa modulation appears to be defined by the following basic parameters:

- **Spreading factor (SF)** is the amount of data that can be sent. It is represented by a number, which indicates the amount of data that can be sent in a given time. The higher the spreading factor, the more data that can be sent. The spreading factor is also used to determine the signal strength and reliability of the transmission and takes values from 7 to 12.
- **Bandwidth (BW)** refers to the width of the frequency band occupied by a single data transmission. It is a measure of how much of the available radio spectrum is being used by the technology to transmit data. LoRa supports multiple bandwidth options, including 500kHz, 250kHz, and 125kHz. The bandwidth setting determines the maximum data rate and range that can be achieved for a given LoRa transmission. In general, a wider bandwidth allows for a higher data rate, but it also increases the likelihood of interference from other radio sources. Conversely, a narrower bandwidth allows for a longer range, but with a lower data rate.
- **Coding Rate (CR)** is the rate at which data is encoded for transmission. LoRa networks typically use a coding rate of 4/5, 4/6, 4/7 or 4/8. A higher coding rate provides more reliable data transmission, but at the cost of lower data rates.

2 Hadrware of Project

For the implementation of the project, two devices were used:

2.1 PlutoSDR

The ADALM-PLUTO [2] Active Learning Module (PlutoSDR) is an easy-to-use tool available from Analog Devices Inc. (ADI) that can be used to introduce fundamentals of software-defined radio (SDR) or radio frequency (RF) or communications as advanced topics in electrical engineering in a self- or instructor-led setting.

Figure 2: PlutoSDR

2.2 Waspmote LoRa SX1272 module

The SX1272 chipset [3] has been developed by the company Semtech. Based on this chipset, Libelium created the Waspmote-compliant LoRa module (or SX1272 module). In Figure 3 we can see a LoRa module with 4.5 dBi antenna. This module is connected to the SOCKET0 placed in the Waspmote board (Figure 4).

Figure 3: SX1272 module

Figure 4: Waspmote board-SX1272 module

The combination of the values SF, BW, CR, defines the transmission mode. There are ten predefined modes in the API, including the largest distance mode, the fastest mode, and eight other intermediate modes that Libelium has found interesting. The predefined modes and its properties are shown in the next table (Figure 5).

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 5: LoRa Modes

3 Purpose of Project

The project's goal is to receive and transmit packets from PlutoSDR to the SX1272 module. At this point, we'll give a quick explanation of how the devices work. The chapters that follow will go into further detail.
Initially for PlutoSDR, throughout the duration of the project, GNU Radio was used [4], which is an open-source software development toolkit . But also some open-source GitHub repositories, which were necessary for receiving and transmitting Lora packets.

- On the Receiving side:
  The SX1272 module transmit a series of packets, where the mode that sends them, changes every two minutes. The modes are shown in figure 5.
  While PlutoSDR's task is to discover which transmission mode the SX1272 module is using and then to decode the message and print the payload.

- On the Transmission side:
  The SX1272 module is configured to Mode 5 and is waiting for a Lora message with payload Nitlab_Ping. After receiving it, it replies with Nitlab_Pong.
  While, PlutoSDR creates a Lora message with the Nitlab_Ping payload and broadcasts it using the transmitting mode 5. Meanwhile, the Receiver of the PlutoSDR is listening to all messages transmitted during the communication. So at the end of the communication the Receiver has messages with both Nitlab_Pong and Nitlab_Ping payloads.

4 Receive LoRa Packets

4.1 Capture LoRa Packets

The first step of the project was to discover at which frequency channel the SX1272 module emits signals. Knowing that the band in Europe is 863 to 870 MHz. This was easily found at channel 13 with a center frequency of 866.1 Mhz (CH_13_868), thanks to the help of the SDRconsole application [5]. The figure below (Figure 6) is a screenshot from the SDRconsole when we capture a LoRa signal.

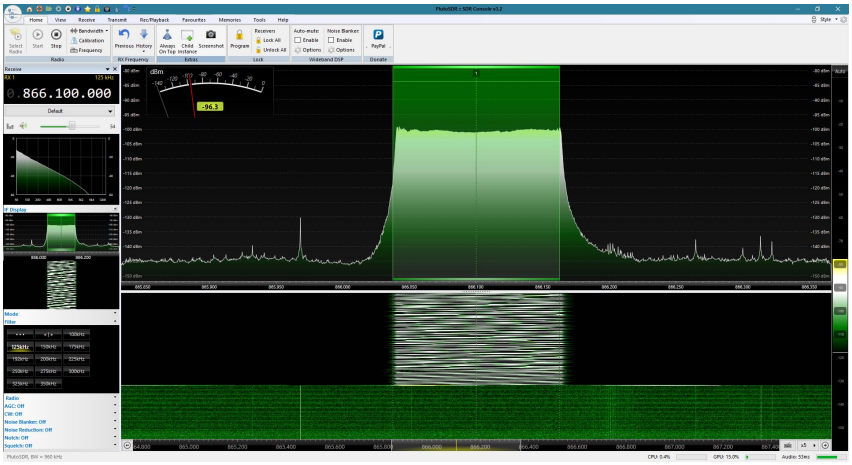

Figure 6: Capture LoRa Signal

4.2 Implementation of LoRa Receiver

As we mentioned above, GNU Radio was used for the implementation and especially the GNU Radio Companion (GRC), which is a graphical user interface (GUI) for creating GNU Radio flow graphs. So now we will analyze which blocks were used to build the Receiver. Also, one of the Github repositories that were used, is from rpp0 [6], which provided us with two GNUradio blocks, the LoRa Receiver and the Message Socket Sink. Below we'll analyze the functionality of each block.

- The **PlutoSDR source block** is used to connect a PlutoSDR device to a GNUradio system. This block allows the PlutoSDR device to be used as a source of RF signals to be processed by GNUradio.
- The **LoRa Receiver block** is converting the coded signal received from PlutoSDR source block into its original form. The LoRa Receiver processes the received signal, demodulates it to extract the informations, performs error correction if necessary, and finally outputs the original data payload
- The **Message Socket Sink block** forwards the payload of the message, that LoRa Receiver finds, to local port 40868 of the host via UDP socket.

Finally, the figure below (Figure 7) shows the flowgraph of this implementation:

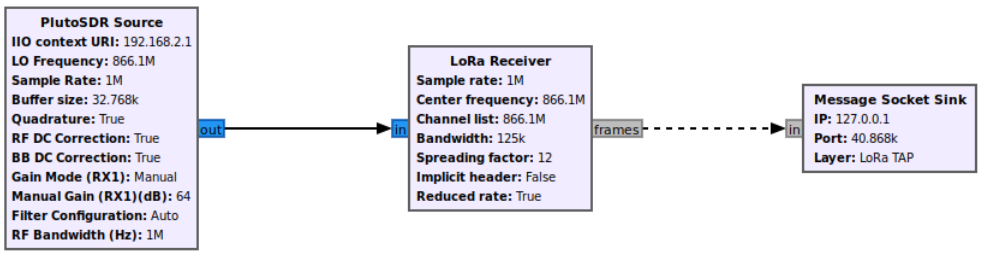

Figure 7: LoRa Receiver

4.3 Implementation of Dynamic LoRa Receiver

In the previous section it was shown the LoRa Receiver implementation, but this is convenient if we know the mode to which the SX1272 module transmits. So for this, on top of the generated GNU Radio Python script of the Receiver, a new Dynamic Receiver was implemented. Whose functionality is to switches between transmitting modes until he discovers the mode in which the SX1272 module is transmitting.
To achieve this, two threads were created, which are connected to two UDP sockets. The message payload was transmitted via socket_1 to local port 40868. Meanwhile, socket 2 is sending a notification type message to local port 4000.
The first is the Controller Thread, whose job is to start the LoRa Receiver with the settings of a mode. After this is done, it waits on socket_2 until it receives a notification type message from the other thread. After receiving a response, it restarts the LoRa Receiver with the next mode.
The second thread initially receives the payload of the message through socket_1 and prints it. Then if it does not receive any message in the specified seconds (in our case it is 10 seconds). It sends a notification to the Thread controller, through socket_2. In order to continue to in the next mode. In the figure below (Figure 8) we see how the Dynamic Receiver is implemented with a pseudo code diagram.

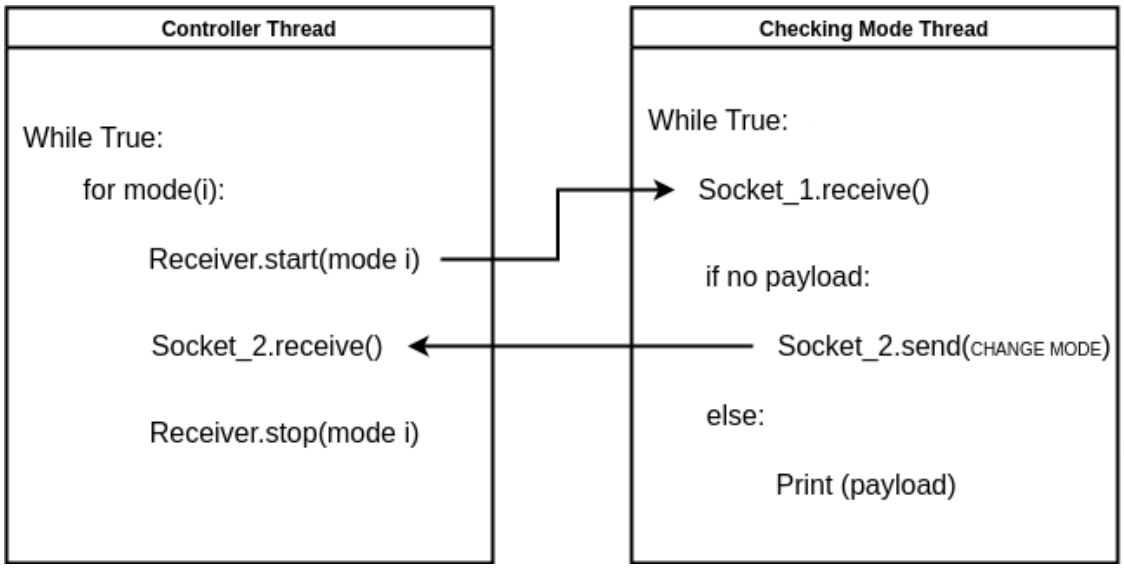

Figure 8: Dynamic Receiver Diagram

4.4 Results of the LoRa Receiver

Using the Dynamic Receiver, we confirmed that mode changes were totally successful. As a result, it is always in the proper mode and receives messages from the transmitter.
By testing the Receiver in all modes, according to Figure 5. The following conclusions on the effectiveness of the receiver in each mode were made.
The Receiver was highly effective in modes 1, 2, 3, 5, 7 and especially in 3, 5, 7. On the other hand, it had barely any success in modes 4, 6 and no success in 8, 9, 10, as he could not decode these messages. Possibly no proper decoding is achieved in modes 4, 6, 8, 9, 10 because that modes use a 500kHz bandwidth which offers a higher data rate, allowing more information to be transmitted in a shorter amount of time than 125kHz or 250kHz bandwidth. In the figure below we see the payload of the message printed by the Receiver for each mode.

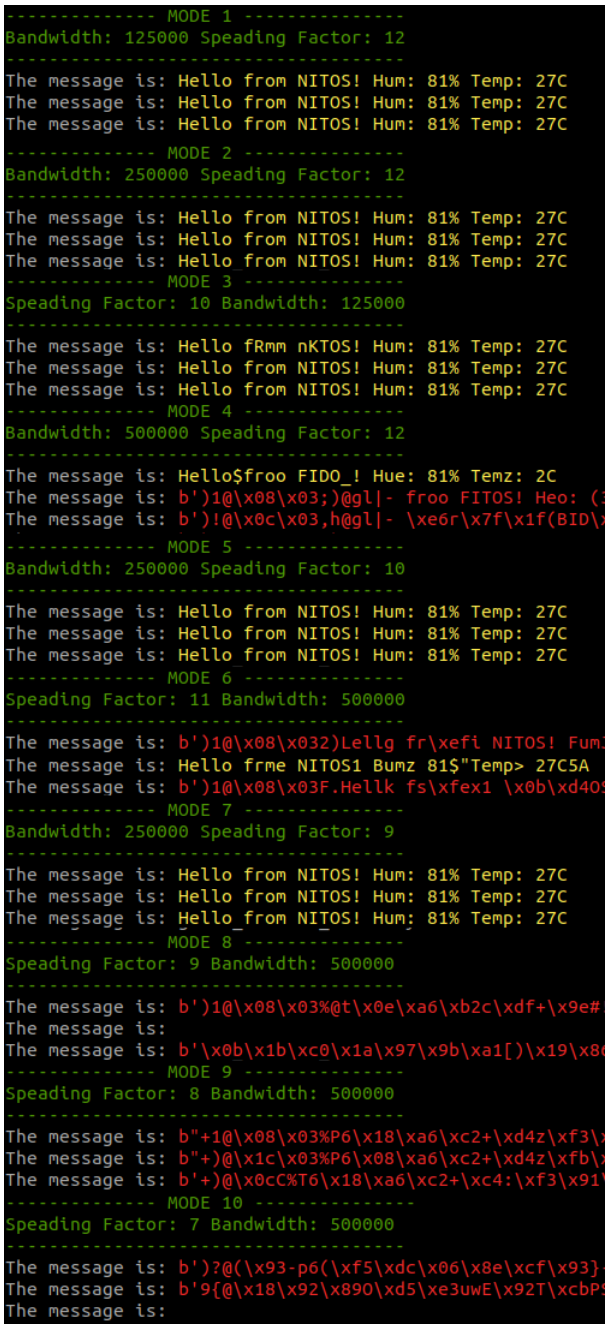

Figure 9: Results of each Mode

5 Transmit LoRa Packets

5.1 Implementation of LoRa Transmitter

For the implementation of the LoRa Transmitter, which was extensible to GNU Radio, we also used the open source github repository from tapparelj [7]. Which provided us with the LoRa Tx block. Below we analyze the usage of each block used in this implementation

- The **Message Strobe block** is to send a message at a specified time. It receives a PMT message and sends it at every period of time we have defined, in this case it is 2 seconds.
- The work of the **LoRa Tx** involves encoding the data, modulating the signal and determining the channel and bandwidth.The data is first encoded to ensure its reliability and correctness, then converted into a radio signal for transmission, and finally the channel and bandwidth are selected to optimise the transmission range and data rate and avoid interference. Then it forwards the signals to the next block.
- The **PlutoSDR Sink block** is used to connect a PlutoSDR device to a GNUradio system. This block allows the PlutoSDR device to be used as a destination for RF signals.

Finally, the figure below (Figure 10) shows the flowgraph of this implementation:

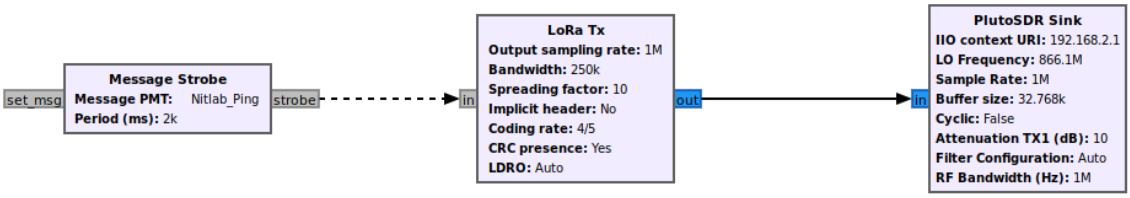

Figure 10: LoRa Transmitter

5.2 Results of the LoRa Transmitter

The results obtained from using the transmitter in all modes of figure 5 were highly favorable. The transmitter demonstrated excellent performance in all modes and successfully transmitted all messages without any issues. The SX1272 module was able to receive all of the messages transmitted in each mode, indicating the high efficiency and reliability of the transmitter. These results demonstrate the effectiveness of the transmitter and its ability to deliver consistent and high-quality performance in all modes of operation. Overall, the results of this experiment demonstrate the exceptional capabilities of the transmitter and its ability to successfully transmit messages in a range of different modes.

5.3 Transmit-Receive Experiment

Since the Receiver and Transmitter were implemented, the project's final goal is to exchange messages between the PlutoSDR and the SX1272 module. Before the experiment began we decided to set all of the devices to Mode 5. With the Transmitter sending a Nitlab_Ping message and the SX1272 module Nitlab_Pong. Therefore, by running the two simultaneously we saw the Receiver receiving both Nitlab_Ping and Nitlab_Pong messages (Figure 11). Therefore, we understand that the message exchange is done correctly. Below we can see the output prints of the Receiver.

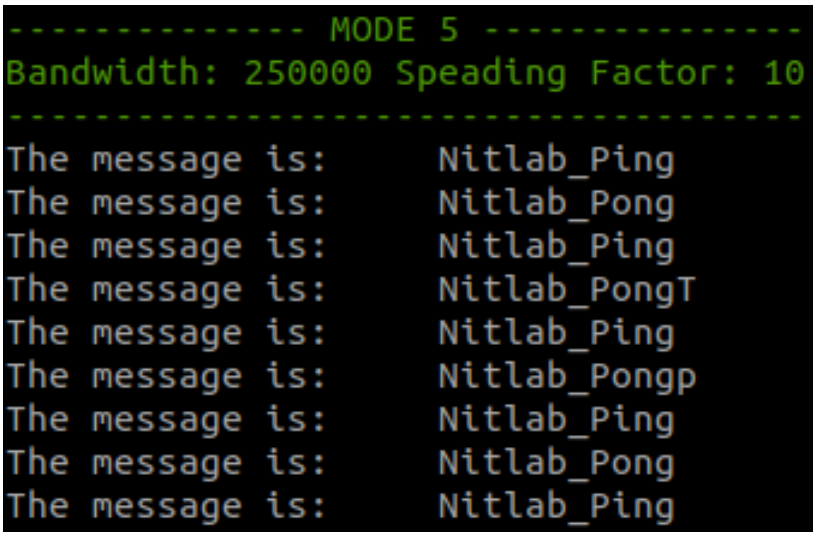

Figure 11: Receiver console

In the following photo we see a snapshot from the device's console, the moment which receives a message which include the phrase Nitlab Ping in the payload and starts sending its response.

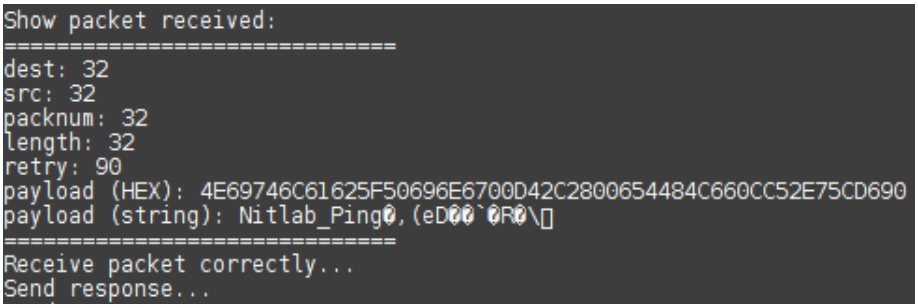

Figure 12: SX1272 module console

6 Conclusion

Also with the report are included the following scripts: the Lora Receiver (Receiver.py), the LoRa Receiver which it visualize signals in a graphical format (Receiver_with_sink.py), the dynamic LoRa Receiver (Dynamic_Receiver.py) and the LoRa Transmitter (Transmitter.py).There is also a README.md which contains information on running the scripts.

References

[1] *A study of LoRa low power and wide area network technology*
https://ieeexplore.ieee.org/document/8075570.

[2] *ADALM-PLUTO*
https://wiki.analog.com/university/tools/pluto.

[3] *Waspmote LoRa SX1272 module*
https://development.libelium.com/lora_networking_guide.

[4] *GNU Radio (3.10.2)*, https://www.gnuradio.org.

[5] *SDR Console*, https://www.sdr-radio.com/console.

[6] *LoRa Receiver*, https://github.com/rpp0/gr-lora.

[7] *LoRa Transmitter*, https://github.com/tapparelj/gr-lora_sdr.

[8] *DecodingLora*, https://revspace.nl/DecodingLora.

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

[10] *A Multi-Channel Software Decoder for the LoRa Modulation Scheme*
https://www.scitepress.org/papers/2018/66684/66684.pdf.