LoRa Lopy2Lopy and Lopy2GW

42080 - Comunicações Móveis

Universidade de Aveiro

Bárbara Moreira 104056, Gonçalo Leal 98008, Sebastian González 103690, Tiago Mostardinha 103944

TP3

2023/2024

DETI - Departamento de Eletrónica, Telecomunicações e Informática

Contents

1 Introduction **1**
- 1.1 Project . . . 1
- 1.2 What is LoRa? . . . 2
- 1.3 Pycom . . . 2

2 Architecture **3**
- 2.1 Database . . . 4
- 2.2 Dashboard . . . 5
- 2.3 Controller . . . 5

3 Protocol **8**
- 3.1 Packets . . . 8
- 3.2 Messages . . . 9
  - 3.2.1 LED functionality . . . 10
  - 3.2.2 Buffer . . . 10

4 Project Folder Structure **12**
- 4.1 Node . . . 12
- 4.2 Gateway . . . 12
- 4.3 Server . . . 13
- 4.4 Running the Program . . . 13

5 Results **14**
- 5.1 Packet Loss . . . 14
- 5.2 RTT . . . 15
- 5.3 Throughput . . . 15
- 5.4 Test Conclusions . . . 15

6 Final Analysis and Considerations **17**
6.1 Positive and Negative Points 17
6.2 GitHub 17

List of Figures

1.1 Pycom board 2

2.1 Project Architecture 3
2.2 InfluxDB 4
2.3 Dashboard interface 5
2.4 Controller flow diagram 7

3.1 Exchanged Packets 9
3.2 ARP Request and ARP Reply exchange 10

5.1 Graph Packet Loss results 14
5.2 Graph results 15
5.3 Graph results 15

Introduction

1.1 Project

This project consists on examining the various connection types within LoRa environments. Our objective is to analyze the impact of interference on these connections and how it affects the transmission of packets.

Through our research, we developed and explored the following types of connections:

- **Connection between a node device and a gateway device**

  ⇒ In this setup, two devices simply exchange information. The communication is typically done using LoRa frequencies, with the device sending information to a gateway and waiting for an acknowledgment.

- **Direct connection between two devices using LoRa frequencies**

  ⇒ In this scenario, one device sends information to another device using LoRa frequencies. The devices are within the range of a LoRa gateway, which receives data from the end devices and forwards it to the LoRaWAN network server.

Firstly, our main goal was to ensure the correct message delivery between devices and understand the factors that can cause interference. We also explored direct messaging to ensure faster and more reliable communication. However, in the end, our goal was not to test the limitations of the Pycom boards but instead to create a proprietary system were our Pycom boards only communicate between themselves and not with other devices nearby.

1.2 What is LoRa?

LoRa stands for Long Range and is a wireless protocol that uses radio waves to encode and transmit information within long distances. It is specifically designed for Internet of Things (IoT), allowing devices to communicate with low power consumption since they use ultra-low power radios.

LoRaWAN, on the other hand, is a protocol developed to enable LoRa end devices to communicate. LoRaWAN stands for Long Range Wide Area Network and is a MAC software layer protocol built on top of LoRa modulation that allows devices to use the LoRa hardware and defines methods to do so. LoRaWAN gateways can transmit and receive signals over long distances and allows bidirectional communication, making it suitable for various applications.

1.3 Pycom

Pycom is a company that produces a range of multi-network IoT development boards and modules, including those that support LoRa and LoRaWAN. For our project, we used three Pycom boards and used Pymakr extension in VS Code for programming and debugging MicroPython code on our Pycom devices.

For our tests and main setup we used one board as a gateway and the other two acting as nodes.
hfill

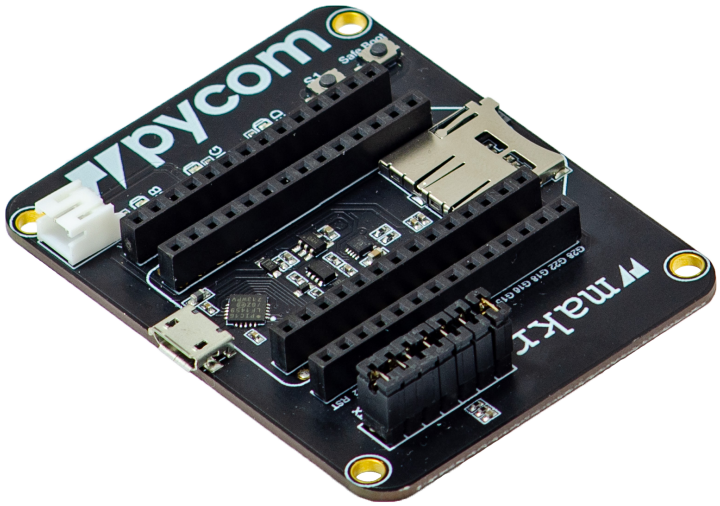

Figure 1.1: Pycom board

Architecture

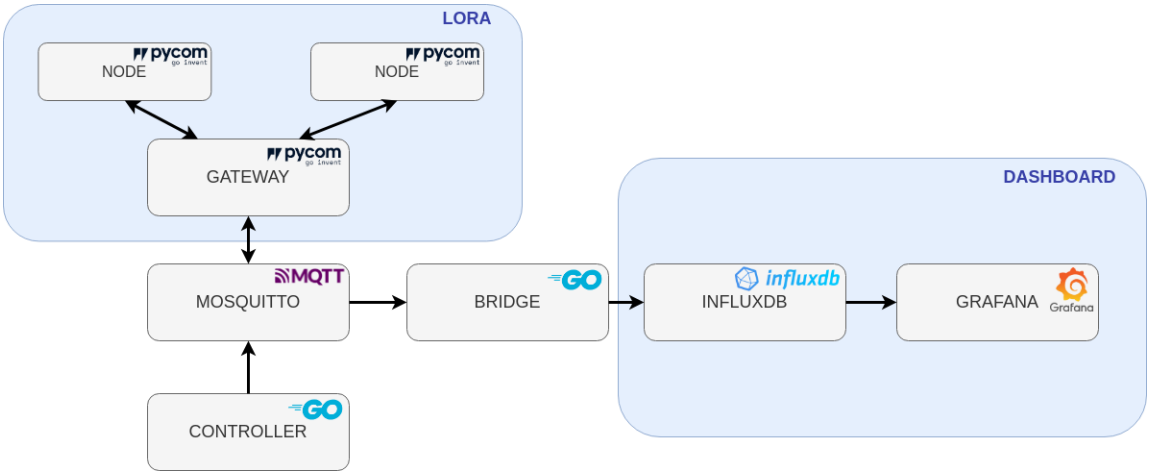

Figure 2.1: Project Architecture

The Figure 2.1 represents a high-level architecture of our project that we will get into detail.
The **LoRa section** includes the nodes and the gateway, and the exchanged messages between them, we will get into detail in the **??** chapter. Mosquitto, an MQTT broker, is responsible for receiving certain values from the gateway, such as the status of LEDs, round-trip time (RTT) values, and throughput data. These are recieved by the bridge that interprets and handles these data and stores them in the **InfluxDB**, database. The results are shown on the **Grafana Dashboard** for monitoring and real-time visualization. The **controller** allows users to modify the LED values, collected by the MQTT, on the LoPy devices.

2.1 Database

The InfluxDb includes all stored data displayed on the dashboard, it's a time series database designed for high-performance data querying and storage. When selecting InfluxDB as the designated database for our LoRa network data, we prioritized several key technical considerations that align with the specific requirements of our project. InfluxDB is a time-series database, renowned for its efficiency in handling large volumes of timestamped data, making it an optimal choice for managing the periodic data transmissions characteristic of our LoRa network.
Additionally, InfluxDB integrates with Grafana, the chosen dashboard solution for visualizing the stored data. This integration not only streamlines the development of our monitoring system but also enhances the accessibility of the data for stakeholders.
Furthermore, InfluxDB's scalability and robust architecture position it as a reliable foundation for future expansion and integration with additional devices. The Figure 2.2 shows the interface of our Database. The graphs display data that the InfluxDB is tracking, over a time range. Bellow, the query builder, shows the data received from our project and the different filter that can be applied to it for easier visualization. These filters include measurement filters, the data being monitored, (LED status, packet loss, RTT and throughput), field filters, configuration filters and device filters, related to the device we wish to specify the data.

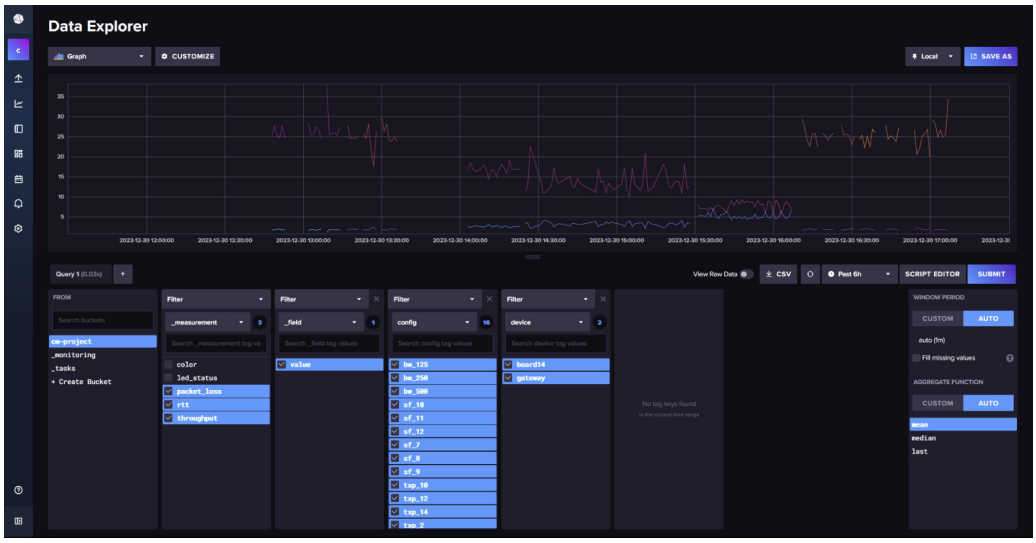

Figure 2.2: InfluxDB

2.2 Dashboard

As previously referenced, the Figure 2.3 shows the Grafana dashboard which displays info from the active nodes. For context, Grafana is an open-source platform used to monitor and analyze metrics from various data sources, for our case, these include, packet loss, led status, throughput and RTT. The last two are shown in graphs, where we can see its variation across the time, this is crucial for later conclusions and network diagnosis.

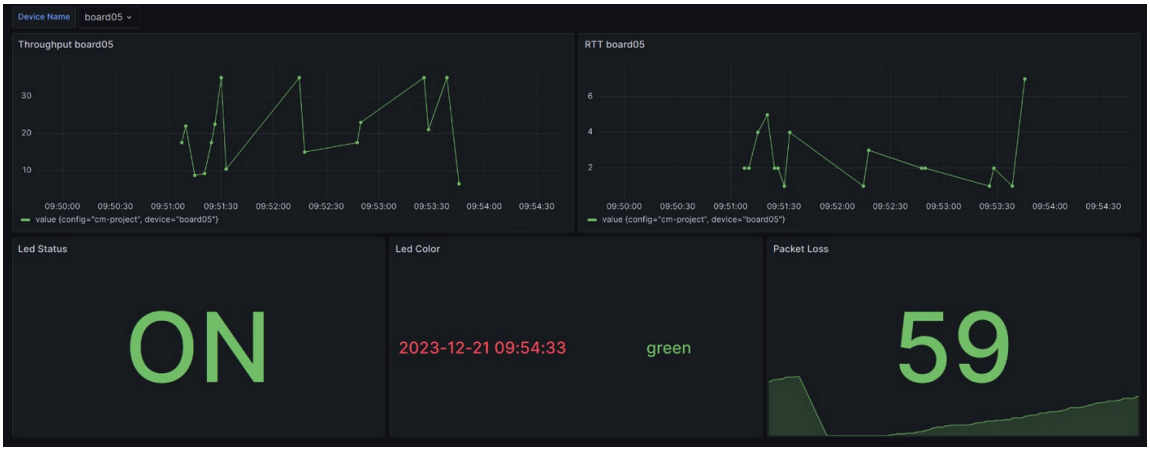

Figure 2.3: Dashboard interface

2.3 Controller

The controller is responsible for interacting with the MQTT broker to specifically control the LEDs of the Pycom boards. This is useful to observe differences between both boards, for example, delay, and to allow information to be sent specifically to one board or the other, instead of just in broadcast, as it is by default.
The controller was implemented using GO programming language due to its efficiency with handling concurrent processes. External packages such as *github.com/eclipse/-paho.mqtt.golang* and *github.com/influxdata/influxdb-client-go/v2* (for InfluxDB integration) were also used.

Here is an example how the Controller should work:

```
NFLUXDB_NAME: cm-project
INFLUXDB_USER: cm-project
Devices:
```

```
0: All
1: board05
2: board14
3: gateway
9: Exit
Select an option: 0
Commands:
0: Turn off LED
1: Set color to red
2: Set color to orange
3: Set color to yellow
4: Set color to green
5: Set color to cyan
6: Set color to blue
7: Set color to purple
8: Set color to white
Select a command: 4
cm-project/controller/all/color
Command sent

Devices:
0: ALL
1: board05
2: board14
3: gateway
9: Exit
Select an option: 2
Commands:
0: Turn off LED
1: Set color to red
2: Set color to orange
3: Set color to yellow
4: Set color to green
5: Set color to cyan
6: Set color to blue
7: Set color to purple
8: Set color to white
Select a command: 0
```

```
cm-project/controller/board14/status
Command sent
```

Listing 2.1: Controller terminal

A Menu is displayed, showing all the devices connected in the network and the possibility to choose the color of the LED, to send to all the active devices.

The following diagram shows the stages involved of the controller:

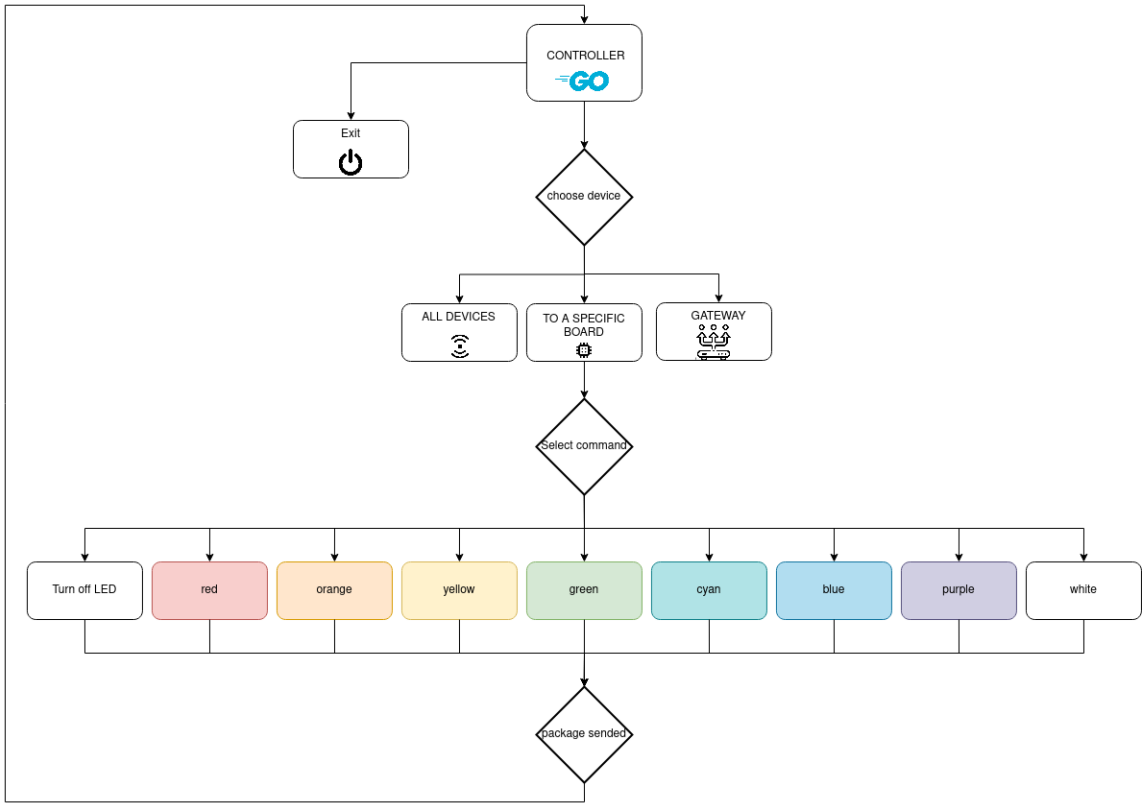

Figure 2.4: Controller flow diagram

Protocol

We developed a protocol to exchange messages between the LoPys and with the gateway that is inspired by previously studied ones like TCP and ICMP and was also adapted to suit the characteristics of LoRa technology.

3.1 Packets

To manage packet types, it's essential to identify their corresponding protocols. For this purpose, each packet has an ID associated, corresponding to the first byte in each frame. When parsed, this ID allows us to determine the type of the packet, according to the **PROTOCOLS** dictionary, and its following behaviour. If the protocol is not valid an exception is raised, signaling an unrecognizable packet.
When composing a packet, we need its' data as well as the ID. If the protocol specified is not recognized, an exception is raised, ensuring that only known packets are constructed. Bellow is each packets' frame composition.

- **ARP REQUEST** it includes:

  ⇒ Id, timeout, size, MAC src, MAC dest

- **ARP RESPONSE** it includes:

  ⇒ Id, timeout, size, MAC dst, MAC src

- **ICMP REPLY** it includes:

  ⇒ Id, timeout, size, MAC dst, MAC src

- **ICMP RESPONSE** it includes:

  ⇒ Id, timeout, size, MAC dst, MAC src

- **TCP SYN** it includes:

⇒ Id, timeout, size, MAC src, MAC dest, synID

- **TCP ACK** it includes:

  ⇒ Id, timeout, size, MAC src, MAC dest, ackID, data

- **TCP SYN ACK** it includes:

  ⇒ Id, timeout, size, MAC src, MAC dest, synID, ackID

When parsing or composing a packet, a flag *param* is always provided as input. It is important to note that only the TCP ACK packet has *param* = *True* meaning it should be handle as a packet with variable size, represended by the *%d* at the end of the protocol. This is done by subtracting the size of the protocol's header from the total size extracted from the packet.

3.2 Messages

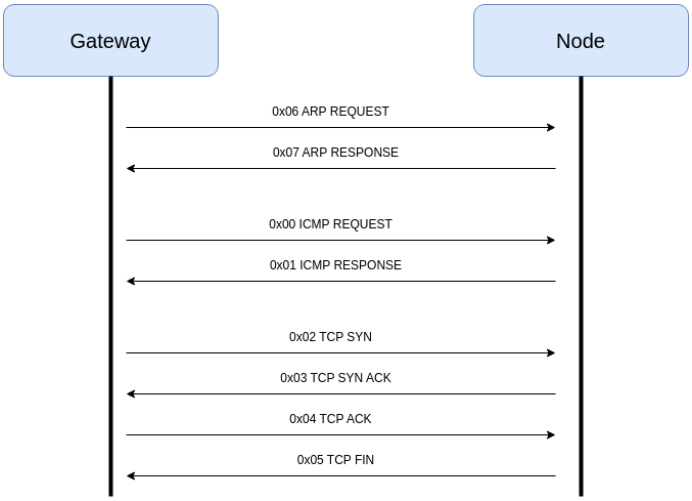

Figure 3.1: Exchanged Packets

The diagram Figure 3.1 shows the communication between the devices. The gateway starts by sending an Address Resolution Protocol (ARP) request to identify new nodes within range, the node then checks if the packet is already in the buffer, if not, it sends an ARP reply. The new nodes are now stored in a **Known Nodes** list, representing all nodes within range, found by the ARP's packet exchange.

The gateway proceeds with sending an ICMP request and the Node repeats the behaviour, sending an ICMP reply. When a node responds to this, it appears in the **Active Nodes** list. This is helpful to understand which devices are in fact participating in the communication itself.
Finally, the TCP section may now be established through a tree way handshake and exchange data. The use of TCP in this context ensures reliable transmission, allowing the monitoring of Round Trip Time (RTT) and throughput values, which are useful for analysing the protocol performance.

3.2.1 LED functionality

In our project, the data exchange can be visualized through the changing colors of LEDs on the nodes. By default, commands are broadcasted and colors are sent sequentially by the gateway to all **active nodes**, notifying them to change their LED. This way, we can visualize which nodes are receiving this command and if there are any delays in the transmission.
The controller can be used for this purpose, by choosing which nodes we want to send information and the configuration we want to apply. Once set, these configurations remain the same until the controller sends a new command. This is done manually by the users, and the feature is useful when wanting to test the communication for a specific node.

3.2.2 Buffer

The buffer ensures efficient data management working as a temporary storage area for incoming packets, which are positioned in arriving order, while a session is active. If no problem occurs, the packets exit the buffer once this session has finished. On the other hand, if a packet stays longer than its timeout value associated, exceeding this value, it is removed and considered lost. The number of packet losses is another aspect we had in consideration when analysing the results.

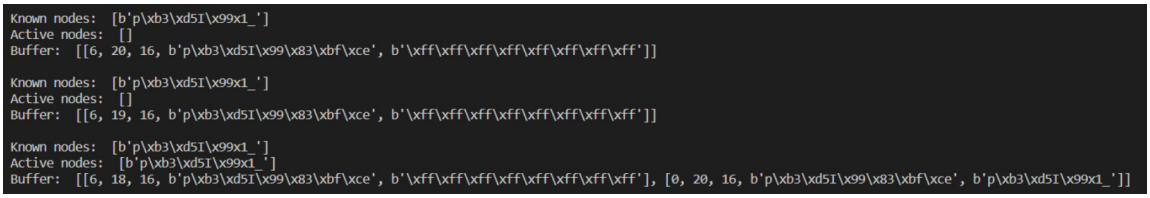

Figure 3.2: ARP Request and ARP Reply exchange

The Figure 3.2 shows the exchange of ARP packets, corresponding to the first pair of exchanged messages. By analysing it, we can observe that only one node was discovered through the ARP request, adding it to the buffer. Following the transmission of an ARP reply, the node is now also included in the list of active nodes, and the most recent packet is appended to the end of the buffer, as it was previously explained.

Project Folder Structure

For better organization we divided the code into three folders: Node, Gateway and Server. Note that, the *utils.py* is common in both Gateway and Node folders and has all the generically used functions. These include parsing and composing packets, managing these within a buffer, and their proper handling.

4.1 Node

The node starts by defining the characteristic of the device itself, including name, mac gateway, status and default led color, which is defined as RED. The code implements a set of operations responsible for defining the node's response based on the incoming packets. It includes the parsing and identification of the received data, checking its presence within the buffer which, as previously mentioned, is managed by reducing the packets' time-out value and discarding if necessary. This process involves interpreting different packet types, such as ICMP, ARP and TCP.
Upon receipt of a packet, the code not only validates its format but also determines if the packet's content is a new event or a repeat. The devices' LED status is dynamically updated based on the received TCP commands that include the color to change to next. Additionally, it prints the buffer to the console and opens a session with the gateway.

4.2 Gateway

Gateway interacts with all the nodes within the network, integrating the LoRa communication capabilities, as well as interacting with the Mosquitto broker, when a MQTT message is recieved, it is decoded and the command is extracted.
The Gateway is responsible for initiating the LoRa section itself, periodically sending ARP requests to discover new nodes. Upon receiving an ARP response, it sends the ICMP requests to establish a connection and later broadcastes the TCP packets.
As seen, the code tracks the status of all nodes, distinguishing between those that are currently active and those that are recognized but not active. During each TCP

session, it calculates the round-trip-time and throughput metrics, which are then sent to the MQTT broker along with the updated list of active nodes. The Gateway's responsibilities include managing these network interactions and maintaining accurate network tracking.

4.3 Server

The server infrastructure plays a central role in our project, since it is responsible for the monitoring of our network. Housing containerized services such as InfluxDB for data storage, a bridge to fetch and store data from the Mosquitto broker, a Mosquitto broker for messaging, and a Grafana dashboard for visualizing results. Additionally, a controller application facilitates interaction with LoRa nodes, allowing us to dynamically control LED colors through MQTT and LoRa communication. This modular and containerized enhances the scalability of our system.

4.4 Running the Program

To run our project, firstly, its necessary to start our server. Enter the following command in the `/server` directory:

```
docker compose up --build -d
./mqtt_influx_bridge/bridge
```

To start the controller, inside the `/board_controller` directory, enter:

```
export $(cat ../.env) &&  ./controller
```

Results

In the course of our LoRa project, we conducted extensive tests to evaluate the impact of key parameters, namely bandwidth (BW), spreading factor (SF), and transmit power (TXP), on the overall performance of the system. The following section present the conclusions emerged from our thorough analysis.

5.1 Packet Loss

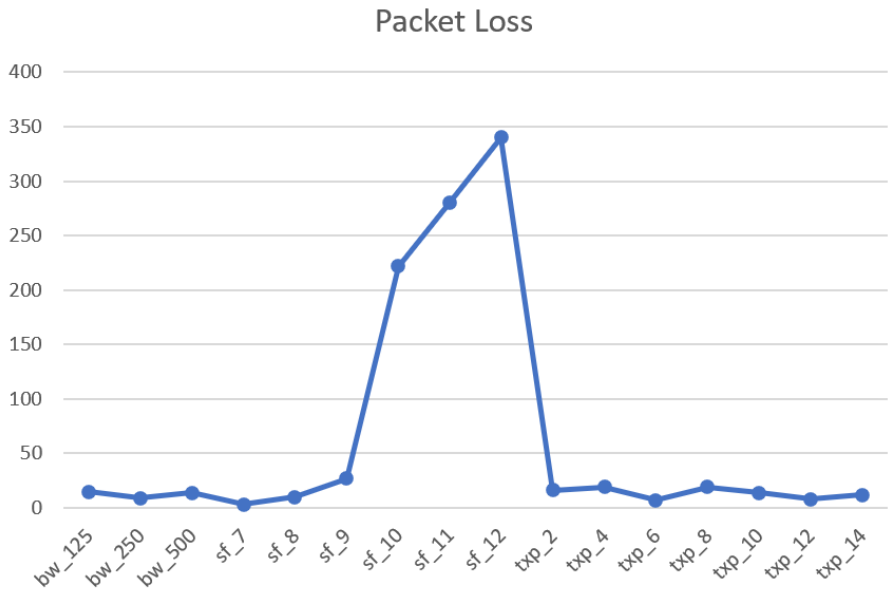

Figure 5.1: Graph Packet Loss results

5.2 RTT

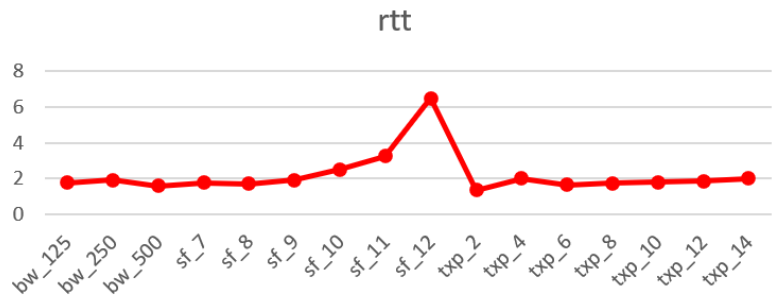

Figure 5.2: Graph results

5.3 Throughput

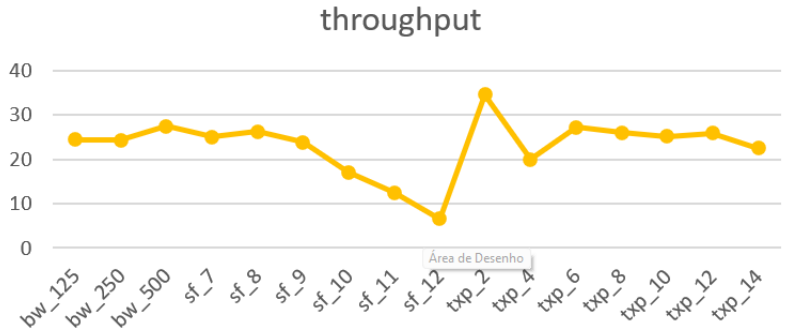

Figure 5.3: Graph results

5.4 Test Conclusions

After conducting testing, collecting data and analyzing the obtained results, it was possible to discern the following correlations:

- **Bandwidth Influence on Signal Sensitivity and Delay**
  Although increasing the bandwidth causes a higher bit rate, this advantage was counterbalanced by an increased susceptibility to noise. The signal proved to be more sensitive, resulting in higher delays during message exchange, as

evidenced by the measured Round-Trip Time (RTT) values 5.2. This shows the importance of carefully considering bandwidth settings to achieve a balance between data rate and system robustness.

- **Spreading Factor Impact on RTT and Packet Loss**
  Our experiments revealed a significant relationship between spreading factor values and key performance metrics. Higher spreading factors were found to increase the RTT values, suggesting an inherent trade-off between communication range and latency. Moreover, higher spreading factors influenced the packet loss which increased exponentially 5.1. These metrics highlight the necessity for thoughtful spreading factor selection to maintain reliable communication links.

- **Transmit Power and Bit Rate Enhancement**
  The influence of transmit power on LoRa performance was evident in the positive correlation with bit rate. Higher transmit power settings generated increased throughput, reinforcing the importance of optimizing transmit power levels to achieve desired data rates 5.3.

In summary, it is possible to observe the subtle interplay between bandwidth, spreading factor, and transmit power in LoRa systems. These insights are crucial when designing real-world applications in order to ensure optimal performance and compliance with the requirements.

Final Analysis and Considerations

6.1 Positive and Negative Points

The project successfully achieved its primary goal, provide us a profound understanding of LoRa technology and the different effects of bandwidth, spreading factor, and transmit power on communication performance.

As an innovation, a proprietary messaging protocol was developed for this project. This protocol, ensured reliable message delivery, facilitated auto-discovery between nodes and the gateway, and implemented health checks through keep-alive messages.

The server-side application added significant value by providing real-time monitoring of network conditions. Metrics such as round-trip time and throughput could be assessed promptly, enabling administrators to optimize network performance efficiently.

The integration of a controller for remote communication with LoRa devices from computers enhanced the project's practicality. The use of MQTT to relay messages between the computer, the gateway, and the destination node over LoRa ensured a seamless and efficient communication flow.

However, the absence of message encryption in the developed protocol introduces a security concern. Operating on a broadcast model, LoRa is susceptible to unauthorized access and man-in-the-middle attacks. Future iterations should prioritize implementing robust encryption mechanisms to address this vulnerability.

Moreover, relying on MAC addresses for device identification introduces a potential liability. This may be exploited, as a device could falsify its identity, leading to trust issues within the network. Future protocols should explore more secure methods for device authentication and identification.

6.2 GitHub

All progress and code can be found '**here**.