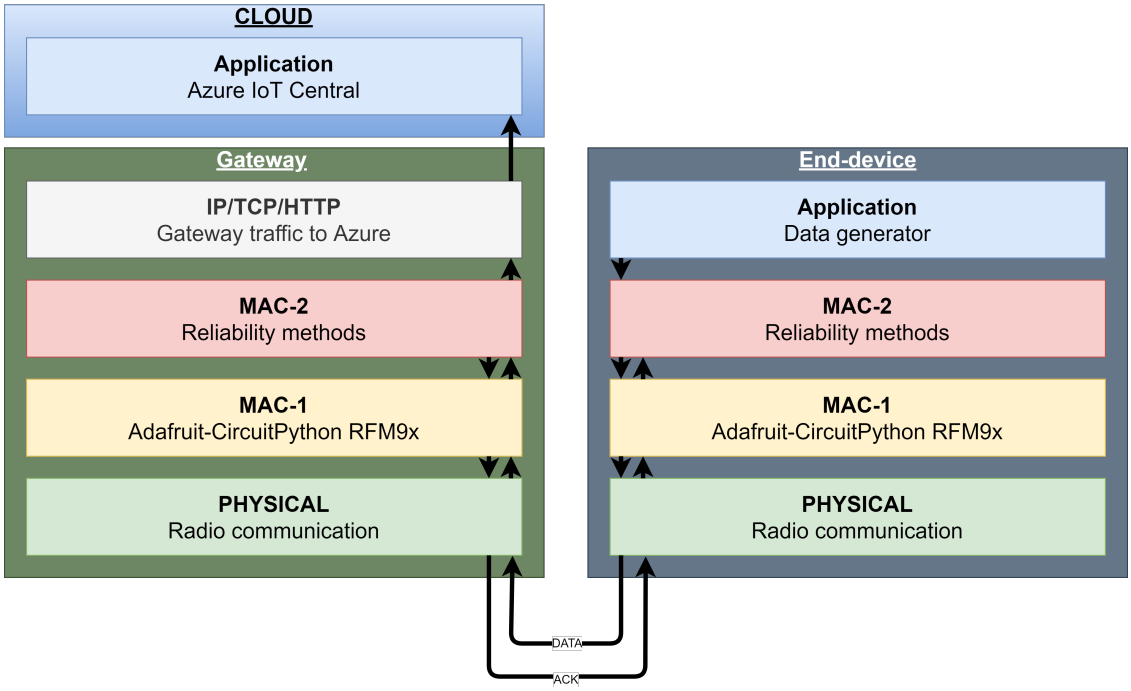
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Reliable Communication using LoRaWAN for Industrial IoT devices

Master's thesis in Computer science and engineering

Joar Blom Rydell

Oliver Otterlind

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Reliable Communication using LoRaWAN for Industrial IoT devices

Joar Blom Rydell
Oliver Otterlind

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Reliable Communication using LoRaWAN for Industrial IoT devices
Joar Blom Rydell
Oliver Otterlind

Supervisor: Ismail Butun, Department of Computer Science and Engineering
Examiner: Marina Papatriantafilou, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Reliable Communication using LoRaWAN for Industrial IoT devices
Joar Blom Rydell
Oliver Otterlind
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The Internet of Things (IoT) area is growing by the minute, with more and more devices being connected to the internet. Long Range Wide Area Network (LoRaWAN) is one technology that makes this possible. It is made to deliver long-range coverage while at the same time using low power. However, with the increasing demand for low latency requirements on connected devices, the same requirements must be fulfilled for the network technology. In particular, the Industrial Internet of Things (IIoT) devices requires the network link to be of low latency and high reliability.

This thesis aims to evaluate LoRaWAN in terms of reliability. This evaluation will be done by comparing the default reliability method against an alternative method. In order to evaluate reliability, latency, Packet Delivery Rate (PDR) and Packet Acknowledgment Rate (PAR) are important metrics to see how reliable the communication is. Furthermore, having concrete values on these metrics will help companies determine whether LoRaWAN is suitable for IIoT devices.

The proposed alternative reliability method is one found in the literature. It alters the uplink and downlink windows of LoRaWAN by introducing a redundant retransmission scheme. This scheme utilizes the Time-on-Air (ToA) of the previous transmission for retransmissions instead of a static value LoRaWAN uses in the default protocol. It further adds a redundant retransmission step if the previous one failed. This thesis shows that the alternative reliability method provides a significant reduction in latency by 69%, an increase in PDR by 1.1% but at the cost of a slightly lower PAR by 3%. Lower PAR is an indication of more network traffic and power usage.

This thesis concludes that there are pros and cons to both methods. The default method is better suited for dense networks with fewer requirements on latency and higher requirements on battery power. In contrast, the alternative method is better suited for less dense networks and higher latency requirements. These trade-offs are ones the manufacturers need to consider when analyzing whether or not LoRaWAN is suitable, that is, both in terms of the manufacturers' environment requirements but also the IIoT device requirements.

Keywords: Azure, Cloud, Gateway, IoT, IIoT, Latency, LoRa, LoRaWAN, LPWAN, Reliability

Acknowledgements

First of all, we would like to thank our supervisor Ismail Butun for continuously providing valuable support and feedback throughout our thesis work. We also want to thank our examiner Marina Papatriantafilou for providing valuable input and pointers throughout this thesis.

We want to thank the Department of Computer Science and Engineering for our educations and support with the thesis. Special thanks to Lars Norén for helping us acquire the necessary hardware that was needed to make this thesis doable.

Finally, we want to express our utmost gratitude to our friends and families for their support throughout our years at Chalmers, as well as throughout this thesis. We would not have succeeded if it was not for you. Thank you.

Joar Blom Rydell, Gothenburg, June 2021
Oliver Otterlind, Gothenburg, June 2021

Contents

List of Figures

List of Tables

List of Abbreviations

ABP	Activation by Personalization
ACK	Acknowledgment
ADR	Adaptive Data Rate
AES	Advanced Encryption Standard
ALOHA	Additive Links On-line Hawaii Area
AM	Alternative Method
AS	Application Server
BPS	Bits Per Second
BPSK	Binary Phase Shift Keying
CR	Coding Rate
CRC	Cyclic Redundancy Check
CS	Chip Select
CSS	Chirp Spread Spectrum
DC	Direct Current
DDR	Data Delivery Rate
DM	Default Method
EN	Enable
ERP	Effective Radiated Power
EIRP	Effective Isotropic Radiated Power
E2E	End-to-End
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction
GND	Ground
GPIO	General-Purpose Input/Output
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IIoT	Industrial Internet of Things
JS	Join Server
JSON	JavaScript Object Notation
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
LTE	Long-Term Evolution
mA	milliampere
MAC	Medium Access Control
MB/s	Mega Byte per second
MHDR	Message Header

MISO	Microcontroller In Serial Out
MOSI	Microcontroller Out Serial In
NS	Network Server
OFDMA	Orthogonal Frequency Division Multiple Access
OTAA	Over the Air Activation
OSI	Open Systems Interconnection
PAR	Packet Acknowledgment Rate
PDR	Packet Delivery Rate
PER	Packet Error Rate
PHDR	Packet Header
PHDR_CRC	Packet Header_Cyclic Redundancy Check
PSK	Phase Shift Keying
POC	Proof-of-Concept
RDM	Reliable Datagram Mode
RTT	Round Trip Time
Rx	Receiver
RSSI	Received Signal Strength Indicator
RST	Reset
SBC	Single Board Computer
SCK	Serial Clock
SF	Spreading Factor
SNR	Signal-to-noise ratio
SPI	Serial Peripheral Interface
SSH	Secure Shell
ToA	Time-on-Air
ToF	Time-of-Flight
TCXO	Temperature Compensated Crystal Oscillator
Tx	Transmitter
UI	User interface
UNB	Ultra Narrowband
VIN	Voltage Input

1
Introduction

The Internet of Things (IoT) is a growing area where many IoT devices are making their way into people's homes. It is a network of different types of devices connected together, which in turn connects to other networks [1]. The concept of IoT devices is to bring intelligent capabilities to otherwise basic devices, with the ultimate goal to, as Perera *et al.* notes, *"create a better world for human beings"*. This concept is also starting to make its way into the industrial area called the Industrial Internet of Things (IIoT). IIoT is about extending the internet to physical devices and also connecting different parts of industrial assets. These devices include, for example, information systems, control systems, and robotics [2].

IIoT devices connected to a network will commonly require the communication to be, apart from secure, also reliable. This means that when data is sent, it is also received, and in the best case, the sender is notified about a successful reception[3][4]. A problem with this is if the data in question is time-critical. Then the latency of the communication is of interest as well. A real-world example would be a wireless motion sensor sending data to a door. Here, the latency is of importance since it is the time it takes for the data sent from the sensor to be received by the door. When the motion sensor is triggered, this information needs to be transmitted to the door to open. The data needs to be received, but the data is also time-critical because a delay in opening the door is not acceptable.

Another aspect of IIoT devices is that some need to communicate long distances or in environments that require specific types of communication methods. The communication can be done using the Long Range Wide Area Network (LoRaWAN), which has been developed and standardized by the LoRa Alliance [5]. LoRaWAN is one of several Low Power Wide Area Network (LPWAN) technologies that exist today and it can be used in many different scenarios. One example is in Nessa *et al.'s* [6] paper, where they discuss using LoRaWAN in mines.

There exist already popular technologies that are deployed on a significant scale like cellular networks. However, one main difference between LoRaWAN and cellular networks is that LoRaWAN targets devices that are battery-powered and do not need to transmit and receive a large amount of data. Since these devices that utilize LoRaWAN usually run on a battery, long battery life is preferable (up to years) while simultaneously being able to send data a long distance. Thus, LoRaWAN aims to cover the part of the network topology where long range is sought, but a high bit rate is not the primary aim. This network topology can be seen in Figure 1.1.

With the increasing amount of IoT and IIoT devices, new types of communication technologies like LoRaWAN emerge. With sometimes strict latency and reliability constraints for the communication of these devices, there needs to be a way of determining whether LoRaWAN is suitable for fulfilling these constraints. Thus having numerical results on how LoRaWAN performs in terms of reliability and latency, can be an indicator of how LoRaWAN performs for different IIoT device settings. Therefore, this thesis aims to be a way for IIoT device manufacturers to see if the numerical results fit their reliability and latency requirements.

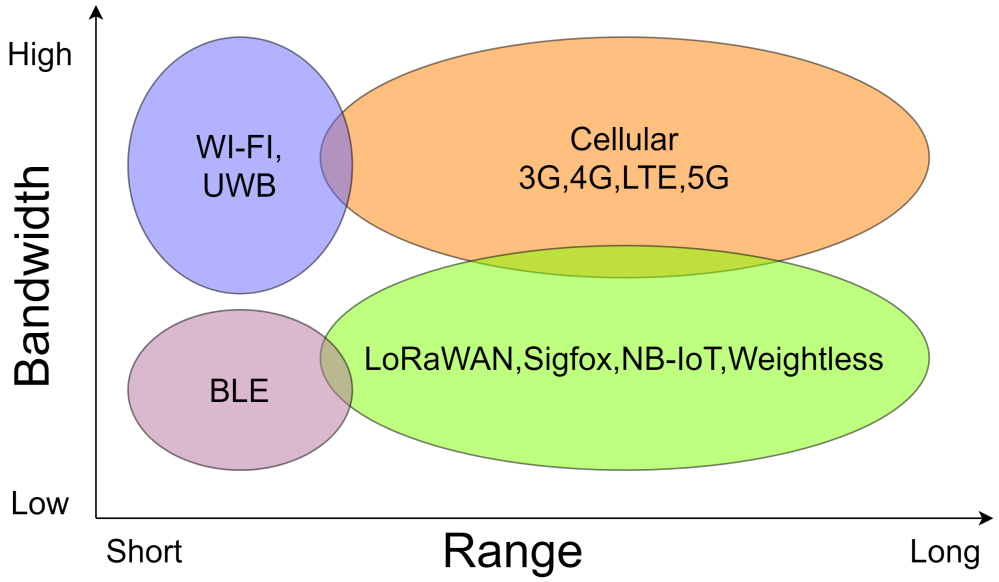


Figure 1.1: The network gap in which LoRaWAN and other LPWAN technologies are trying to fill.

1.1 Problem statement

The reason for evaluating LoRaWAN in terms of reliability and latency is because it is a new technology. Combined with the fact that data is being sent a long distance through radio signals, it can be significantly affected by outside interference [7]. Furthermore, IIoT devices require a more stable and reliable communication link between transmitter and receiver due to their sometimes mission-critical deployment within the industry. Therefore, having numerical results on how LoRaWAN performs in terms of latency, Packet Delivery Rate (PDR), and Packet Acknowledgment Rate (PAR) will help companies determining if LoRaWAN is suitable for their particular device use case.

Additionally, comparing the default reliability method in LoRaWAN to an alternative method can be used as an indicator to see if the default method can be improved. With this said, this thesis aims to be a way for IIoT device manufacturers to see if

LoRaWAN fits their reliability and latency requirements. It will also show if there are improvements to be made to the default reliability method in LoRaWAN.

1.2 Method

To achieve these aims, the default reliability method in the LoRaWAN protocol will be compared with an alternative method found in the literature. Both of these methods are located in the upper part of the LoRaWAN MAC protocol. This will be covered more in Section 5.1. Once the alternative method has been selected, the evaluation will be done using several Single-Board Computers (SBC). It is then possible to imitate a LoRaWAN-network using dedicated Long Range (LoRa) modules.

Suppose one of the SBCs is an IIoT device with some sensor, for example, a temperature or proximity sensor. When this sensor receives input, we want to send this input through the LoRaWAN network in a reliable and low latency fashion. During the transmission of the input data, it is then possible to monitor the latency. Furthermore, it is also possible to monitor how many packets are delivered to the recipient and the network traffic. Thus reliability in this thesis will be measured by how many % of sent packets an end-device can deliver to its gateway.

1.3 Research questions

The following research questions have been answered in this thesis:

1. How does the default reliability method compare to an alternative reliability method for LoRaWAN?

2. How do the default and alternative reliability methods affect the time it takes for a message to be delivered to a gateway from an end-device?

3. What can the default and alternative reliability methods achieve in terms of latency, PDR, and PAR for LoRaWAN?

1.4 Restrictions

This thesis work will be restricted to already existing reliability methods. This thesis will not change the underlying original LoRaWAN protocol for reliability. Instead, it is supposed to be an easy way to add the alternative reliability method to the original protocol if desired. Furthermore, this thesis will not have security aspects in it but instead focusing on making sure that there is a reliable communication link and that the data is intact.

1.5 Report structure

The following thesis is divided into eight chapters, structured as follows.

The first chapter introduces the reader to the thesis topic through the background, problem, and aim sections.

The second chapter gives the reader a technical background on the topics related to this thesis. It will begin with giving the necessary definitions for the metrics used in this thesis for the evaluation. It will then give an overview of the cloud service Azure and provide some signal theory that will give the reader the necessary knowledge for the rest of the thesis. Finally, it will provide an in-depth look at how the LoRaWAN network works.

The third chapter will cover the related work to this thesis, including which alternative reliability method is evaluated in this thesis.

The fourth chapter will define the methodology of how the thesis work has been performed. It will begin with the requirements that a potential reliability method needed to fulfill to be considered for evaluation. It will then give an in-depth look into how the design of the physical testbed was done. The chapter is then concluded with the evaluation methodology.

The fifth chapter will start with showing wherein the network stack the implementation took place. It will then go through the whole implementation phase of the custom LoRaWAN protocol for the physical testbed. It will also cover the implementation of the alternative reliability and a comparison between the two methods. The chapter is then concluded with the implementation challenges that came up during the implementation phase.

The sixth chapter will showcase the results of the evaluated methods from the tests done on the two physical testbeds. It will go through the numerical results collected for this thesis. It will then conclude with a result comparison with Nessa *et al.'s* results.

The seventh chapter will discuss different aspects of this thesis. First, it will go through the obtained results from the tests and the custom-built LoRaWAN infrastructure. It will then conclude with some future work that can be performed on this thesis.

Finally, the eighth chapter concludes this thesis.

2
Technical Background

This chapter will go through the technical aspects regarding the technologies used in this thesis. The first section defines the most used metrics in this thesis and how reliability is defined. The second section will give general insight into how the cloud service Azure works. In the third section, some general signal theory is provided that will cover the most relevant parts of this thesis. The fourth section will give detailed information on how the LoRaWAN network works. The fifth section covers the hardware used in this thesis. The chapter is then concluded with information on other technologies that are similar to LoRaWAN.

2.1 Metrics and Reliability definition

To be able to conduct this thesis, there are a few metrics that are required for the evaluation of the reliability methods. Furthermore, a definition of what reliability means is essential to understand the following chapters. Therefore, this section will explain each of these metrics and define what reliability means. The way each metric will be evaluated will be covered later on in Section 4.2.4.

2.1.1 Latency

In general network terms, latency is the measured time it takes for a data packet or a data request to go from sender to receiver. It is measured in milliseconds, and a latency that goes towards zero is preferable. In the Transmission Control Protocol (TCP), for example, acknowledgments are sent by the receiver to the sender to confirm that it has received the data. Therefore, the so-called Round Trip Time (RTT) is an essential factor in the performance of a network. RTT is the time it takes for a data packet to be sent to a receiver, plus the time it takes to acknowledge that same packet to be received back at the sender.

In this thesis, latency is defined as the time it takes for an end-device to send its data and it to be received by a gateway. This is also known as Time-on-Air (ToA) or Time-of-Flight (ToF) for radio-sent data. A simple illustration of this can be seen in Figure 2.1.

2.1.2 Packet Delivery Rate

Packet Delivery Rate (PDR) is a network performance metric to show how many % packets a sender has sent and how many packets have been received by a recipient.

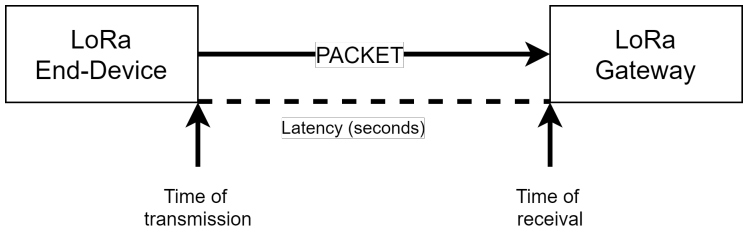


Figure 2.1: Simple illustration of what is defined as latency in this thesis.

In this thesis, this translates into how many % of packets have successfully been transmitted from a LoRaWAN end-device to a LoRaWAN gateway. PDR can be calculated by dividing the number of messages received by the number of messages sent, as seen in Equation (2.1) where P are packets.

$$\frac{P_{\text{Received}}}{P_{\text{Sent}}} = \text{PDR} \tag{2.1}$$

2.1.3 Packet Acknowledgment Rate

Packet Acknowledgment Rate (PAR) is a metric to see how much the recipient successfully acknowledges % of sent packages. The PAR can be calculated by dividing the number of acknowledged packets by the number of received packets, as seen in Equation (2.2) where P are packets.

$$\frac{P_{\text{Acknowledged}}}{P_{\text{Received}}} = \text{PAR} \tag{2.2}$$

The PAR metric can be an indicator of how many packets are unnecessarily sent but also an indicator of how much traffic the reliability method generates. This is because a reliability method will continue to retransmit until either a specific time threshold has passed or x retransmissions have occurred.

2.1.4 Reliability

The concept of general network reliability involves many things—everything from network structure, tolerance against node - and link failures to the communication protocol. In particular, communication is an important part. Reliable communication is communication where all sent application data is received intact and with no duplicate data [3][4]. Therefore, a reliability protocol that performs at maximum efficiency will have a PDR of 100%. This means that measuring a reliability method's efficiency can be done with PDR.

2.2 Azure

Microsoft Azure is a cloud service with over 200 services, and products [8]. It is a cloud platform where users can streamline application development and host

existing applications [9]. Azure also makes it easy to host small applications, with the option to expand if necessary. Furthermore, Microsoft state that Azure offers high availability, as well as reliability [8]. Azure has several services that can be utilized for different purposes. The following section will cover IoT central, Azure functions, and IoT Central Device Bridge.

2.2.1 IoT central

IoT Central is a platform that helps the user maintain, develop and manage IoT solutions [10]. The IoT central has a web User Interface (UI) that lets the user manage numerous devices, and create rules and handle the data that the devices produce [10]. In the web UI, the user can define different types of devices that connect to the user's platform through device templates [11]. The device templates can be formed using already implemented devices or created as a template for the development of the device [10].

When a device starts sending telemetry data to the IoT Central, a device template needs to be applied to this device. After the device has been assigned a device template, the telemetry data from this device can be displayed in various ways. A few of these graphical ways can be seen in Figure 2.2. Additionally, if the connected devices support it, the IoT central can be used to control the connected devices as well.

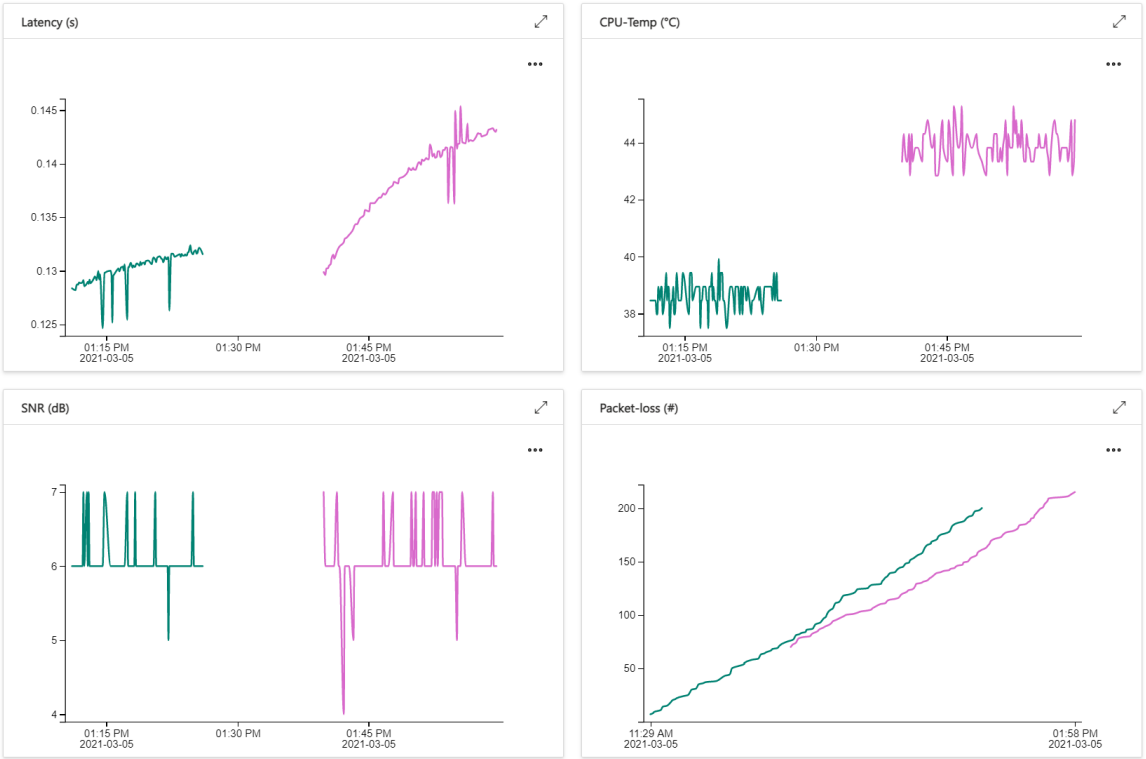


Figure 2.2: Example of different ways to display data on Azure IoT Central. In the different graphs, the last 100 values are displayed for the respective metric and telemetry data.

2.2.2 Azure Functions

Azure functions let the user create event handlers through Azure. It offers a serverless way to write the code needed for an application, meaning that Azure can allocate computing resources on demand. The functions' code execution can be triggered by so-called webhooks, which are methods that can alter either web pages or web applications. They can also be triggered by cloud service events or on a defined schedule [9]. This means that as long as the user can, for example, make HTTP post requests, the code can be coded in any programming language that supports HTTP requests.

2.2.3 IoT Central Device Bridge

An IoT Central Device Bridge is an open-source solution that utilizes the Azure function to connect an IoT network to a user's IoT Central [12]. It works by forwarding the messages sent to it to the IoT Central. When a device bridge receives a message from its network, it simply makes an HTTP post request and sends it to the user's Azure function. This Azure function is essential to making this work. The data in the HTTP post request is in JSON format and can be seen in Listing 2.1. When the Azure function receives this request, it forwards the ID and the telemetry data to the IoT Central, where the device and its data appear.

```
{
    "device": {
        "deviceId": "lora-end-device-id"
    },
    "measurements": {
        "cputemp": 40.31,
        "latency": 0.128,
        "rssi": -45,
        "snr": 6
    }
}
```

Listing 2.1: The JSON-format of the HTTP-post request to the Azure function.

2.3 Signal theory

When it comes to LoRa and LoRaWAN, there are many types of signals and parameters that play a significant role in making sure messages can be sent and received. Furthermore, there are many ways that data gets converted into signals before transmission, known as modulation. This section will define some of the most important parameters to know regarding reliability in LoRaWAN and signaling in general. It will further go into different modulation techniques used by different LPWAN technologies.

2.3.1 Received Signal Strength Indicator

The Received Signal Strength Indicator (RSSI) measures how strong the power is in the signal. It is used as a way to see the quality of the received signal from an

end-device. It is measured in decibel milliwatts (dBm) and has a negative number indicating the signal quality. A value that is approaching 0 is considered perfect, while a signal lower than -100 is considered weak [13]. Thus, the RSSI value can be used to determine the signal's quality and can be used to estimate the distance between receiver and transmitter.

2.3.2 Signal-to-noise ratio

The Signal-to-noise ratio (SNR) is defined as the ratio between the power of the received signal and the power of all other signal interference, also known as the noise floor power level [13]. It is measured in decibel (dB). A value greater than 0 dB is considered good since that means that there is more signal than noise and that the signal is above the noise floor. Typical values in LoRa ranges from -20dB to +10dB. The SNR value can be used as a first indicator to see how corrupted the received signal is.

2.3.3 Tx-Power

Tx-Power is the power of the transmitted signal, and it is measured in dBm. This parameter can be manually set when configuring the LoRa transceiver module. Usual limits in LoRaWAN are between 5-20, which corresponds to 5 to 20 dB gain in the signal [13]. Thus, Tx-power will have an effect on the signal range and affect the signal's stability. However, it also affects the device's battery life since a higher Tx-power requires more power from the LoRa module.

2.3.4 Spreading factor

The Spreading factor (SF) is a way to decide the number of so-called chirps to be sent per second [14]. These chirps are the data carriers when being sent from a transmitter to a receiver. Hence, it decides the data rate of the link. In LoRaWAN, the SF can be set to a value between 7 to 12. A low SF means more chirps can be sent per second, and a high SF value means fewer can be sent. A low SF implies that the transmission rate increases, meaning that more data can be sent at a given time. However, sending data at a fast rate means less time for the receiver to sample the signal, which means that noise can then affect the signal [14].

If a higher SF value is used, the transmission rate is lowered and increases the energy consumption since the module is active longer due to increased Time-on-Air (ToA). However, there is more time to sample the signal, so the receiver's ability to distinguish the signal from noise increases. This also yields a more extended range and hence increased network coverage [14].

The takeaway from this is that a lower SF should be used if the end-device is rather close to its gateway. If the end-device is far away from the nearest gateway, a higher SF is required for the data packet to be reached by the gateway. However, as mentioned, this is at the cost of a lower data rate which will be covered more in-depth in Section 2.4.3.

2.3.5 Link Budget

A link budget measured in dB can be used to determine the performance of the entire network link. It consists of the sum between all the gains and the losses of the link, i.e., from the transmitter, through the free air, and then to the receiver. For example, LoRaWAN has a link budget of 154 dB, which is higher than any other standardized communication technology [15].

2.3.6 Effective Radiated Power

When it comes to transmission power (Tx-power), there are a few variables that can affect it. The Effective Radiated Power (ERP), measured in dB, is the total amount of power that is radiated by a real antenna [13]. This value can be compared to a theoretical value called Effective Isotropic Radiated Power (EIRP), also measured in dB. The relation between ERP and EIRP is that an additional 2.15 dB is added to the EIRP value. That is:

$$EIRP\ (dBm) = ERP\ (dBm) + 2.15 \tag{2.3}$$

This is because, in ERP, the power from a real antenna is relative to a so-called half-wave dipole instead of a theoretical isotropic one [13]. Thus, depending on which type to use, either subtract 2.15 dB for ERP or add 2.15 dB for EIRP. Both of these can be calculated as:

$$EIRP\ (dBm) = Tx\ power\ (dBm) + antenna\ gain\ (dB) - cable\ loss\ (dBm) \tag{2.4}$$

The purpose of having ERP and EIRP is because there are regional regulations on how high these values can be. For the 433 MHz bands, for example, the maximum EIRP is 12.15dB [16].

2.3.7 Modulation Techniques

Modulation is how data bits are converted into electrical signals, done at the physical layer of the OSI model or the data link layer if it is an access method. There are many different types of modulation techniques for wireless transmissions, all with their pros and cons. This subsection will outline some of the commonly used techniques for LPWAN technologies.

2.3.7.1 Chirp Spread Spectrum

Chirp Spread Spectrum (CSS) modulation is a wideband technique and is used in LoRaWAN. It is designed for LPWAN technologies to be able to send and receive data at a long distance while at the same time using little power. By having a variable bandwidth of 125kHz, 250kHz, and 500kHz, CSS uses this bandwidth along with the SNR value to determine the optimal SF value to use [17]. This makes it able to spread the signal over its entire bandwidth and is robust against noise interference.

2.3.7.2 FDMA, TDMA, and OFDMA

Frequency Division Multiple Access (FDMA), Time Division Multiple Access (TDMA), and Orthogonal Frequency Division Multiple Access (OFDMA) are three types of multiple-access techniques. Multiple-access means that multiple nodes can be connected to the same communication channel and use it to transmit over.

The main difference between FDMA and TDMA is that FDMA's bandwidth is divided into multiple frequency bands used for transmission, where these bands usually are reserved for a particular station to send and receive data. TDMA instead divides its bandwidth into time slots given to each station to send and receive data. This method, however, requires synchronization since every station needs to know the beginning of its time slot [18].

OFDMA is the technique used in Long-Term Evolution (LTE) and 4G. It divides its bandwidth into multiple narrow orthogonal bands, which are in turn divided into thousands of sub-carriers. These sub-carriers are then assigned as a group to users. All these bands are spaced in a way that they do not interfere with each other, meaning that they are not prone to signal noise [18]. Both OFDMA and FDMA are being used in another LPWAN technology, namely NB-IoT, which will be covered later in Section 2.6.3.

2.3.7.3 Phase shift keying

Phase Shift Keying (PSK) is a modulation technique that performs changes to the signal's phase instead of changing the amplitude or the frequency like other techniques. It works by changing the sine and cosine inputs of the fixed frequency signal at a specific time. The most common one is the Binary Phase Shift Keying (BPSK) and is being used by the LPWAN technology SigFox covered later in Section 2.6.1.

2.3.8 Line-of-sight and Non-line-of-sight

Line-of-sight (LOS) and Non-line-of-sight (NLOS) are two terms related to wireless communication. It has to do with the fact whether the sender and receiver have a clear and unobstructed LOS between them or not [19]. If there is an obstacle such as a wall, a tree, or a building between them, there is an NLOS that can lead to many things. Objects absorb the signal's energy, hence lowering the effective range of that signal and the ERP. This can be dealt with by using, for example, a better antenna at both ends. Usually, however, there need to be alternative ways for the signal to propagate due to path loss. This can be done with relays or by using the so-called multipath method. This method involves the signal being sent in different directions, meaning that many versions of the same signal can reach the receiver but have taken different paths along the way. All these effects, however, ultimately affect the total link budget.

2.4 LoRaWAN

LoRaWAN is a network protocol meant for Low Power Wide Area Network (LPWAN), and it is designed for long-range wireless communication over large areas. The first section will start by going through the network architecture of LoRaWAN. The second section will go through the different kinds of device classes that exist and what they mean in terms of usage. The third section covers the meaning of having a duty cycle and variable data rates. The fourth section will go through what reliability is for LoRaWAN specifically. The fifth and sixth sections will cover the device join process and security aspects in the LoRaWAN protocol. Finally, the last section defines the different message structures used in both standard LoRaWAN and custom implementations.

2.4.1 The LoRaWAN architecture

LoRaWAN is a Media Access Control (MAC) protocol that builds on top of the LoRa physical layer since LoRa lacks the link and network layer. This means that LoRaWAN is defined for the second and third layers of the OSI model. Therefore, it gives LoRaWAN the means to act as a routing protocol, handling communication between gateways and connected devices. The transmission and receipt of data are done via radio frequencies using LoRa modulation, and depending on where the devices are located, different frequencies are allowed. For example, in Europe, the 433 MHz and 863-870 MHz can be used. For the United States, Canada, and South America, the range is between 902-928 MHz [20].

The entire network is built using a star-of-stars network topology, and it consists of the end-devices, gateways, Network Servers (NS), Join Server (JS), and Application Server (AS) [21]. These five device types serve different purposes in the LoRaWAN network, and they are illustrated in Figure 2.3. The end-devices are typically a form of sensor device and can be connected to multiple gateways. They send and receive data to and from the gateways through LoRa modulation. These gateways then relay the data from the end-devices to the NS, and they are also responsible for sending data back to the end-devices if so needed. The communication between the gateways and the NS is done via an established medium like fiber, Ethernet, Wi-Fi, or satellite through TCP/IP [22].

The NS is the brain of the network. Since the end-devices are usually battery-powered, all power-consuming tasks like packet-filtering and security checks are done at the NS. It is also responsible for managing all the gateways and ensuring that all connected devices are running at their optimum settings. This involves the NS continuously checking and adjusting the end-devices different parameters like its SNR value, SF, Tx-power, and frequency sub-channels. Since an end-device located closer to the gateway does not need to have a high SF or high Tx-power, that saves both air time and energy. This is what is called Adaptive Data Rate (ADR) [23].

In the scenarios where the NS cannot control and send ADR instructions to the

end-devices, the end-device's application layer should instead take care of the SF and Tx-power [21]. In this case, the application layer should try to minimize the ToA but maximize the data rate for the network setting. Another role of the NS is to route data from the gateways to the AS, which can, for example, be an Azure function. It also relays the join procedure between the end-devices and the JS when the end-devices connect to the LoRaWAN network.

As mentioned, LoRaWAN is a star-of-stars network topology, meaning that the NS is in the absolute center of the network with gateways connecting to them. In turn, all of the gateways have their star topology, with them being in the center and all end-devices connecting to them. Hence, the star-of-stars topology. Using this kind of topology has its advantages and disadvantages. Firstly, compared to another typical topology like the mesh network, where all nodes are connected together [24], the end-devices do not constantly have to be in listening mode. This is because they only need to send data to the gateways and not relay data to other end-devices. This, in turn, yields low energy consumption, which is desirable.

Secondly, as previously mentioned regarding power-consuming tasks, the end-devices do not need routing since they only communicate with the gateways. The result of this is that the end-devices do not have to be complex, making them cheap to manufacture.

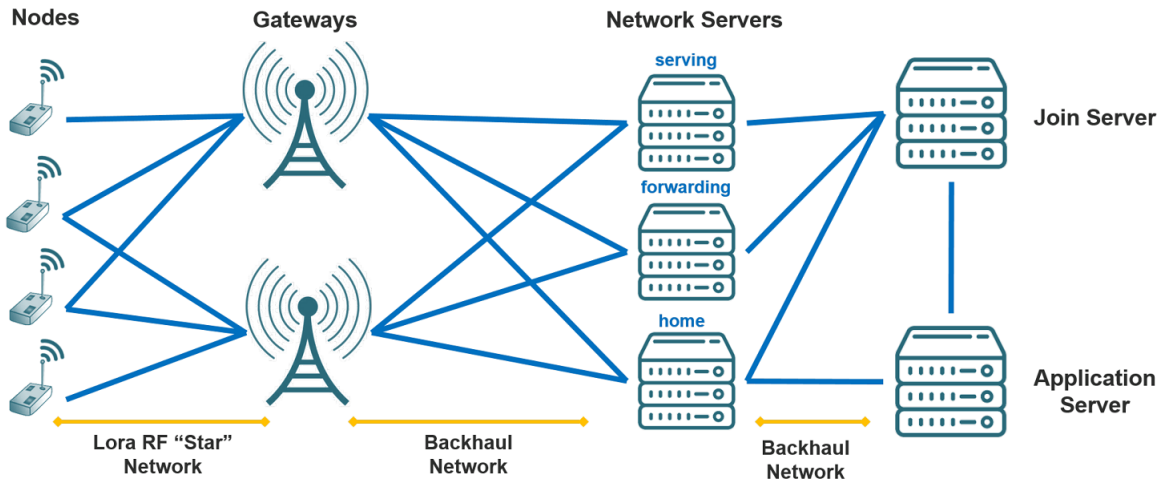


Figure 2.3: The LoRaWAN architecture. Image by Butun *et al.* [25].

2.4.2 Different device classes

There are different classifications on the end-devices in LoRaWAN. This comes from the fact that end-devices have diverse requirements and are used for different types of applications [26]. Class A is the base class and has to be fulfilled by all end-devices. It is for battery-powered sensors with high energy efficiency with the possible need for bi-directional communication. The end-device opens two short downlink receive

windows after transmitting the data uplink, after which it then stops listening for incoming messages. This bi-directional communication has slight variations based on a random time basis, originating from the Additive Links On-line Hawaii Area (ALOHA) protocol. If the Class A device is supposed to receive data, then the data must be sent after the Class A device has initiated an uplink transmission. See Figure 2.4 for an illustration of the transmission windows.

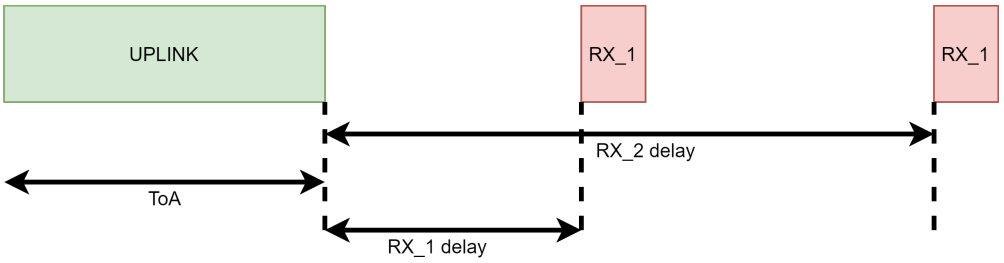


Figure 2.4: Class A receive window diagram.

Class B works in the same way as a Class A device, implementing all of its features. It also has extra scheduled receive windows, and these slots are decided and synchronized with the connected gateway [26]. This synchronization is done using beacon messages that are sent out to the end-devices. This can be seen in Figure 2.5.

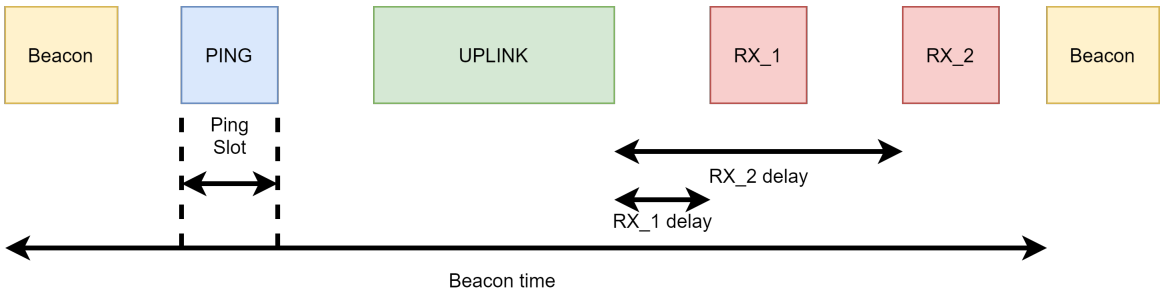


Figure 2.5: Class B receive windows. Contains a ping slot uplink and two receive windows.

The final Class C can receive continuously except for when it is transmitting data. See Figure 2.6 for an illustration of its transmission windows.

2.4.3 Duty Cycle and Data rates

The duty cycle exists to reduce the risk of messages colliding, often leading to packet loss. This is a common occurrence in radio communication [27]. To cope with this, LoRaWAN uses duty cycles that specify how much an end-device is allowed to transmit within a period. It should be lower than 10%, usually around 1%. This means that if a 1% duty cycle is used, an end-device is only allowed to transmit during a one-time unit for every 100th. This, along with the allowed data rate and

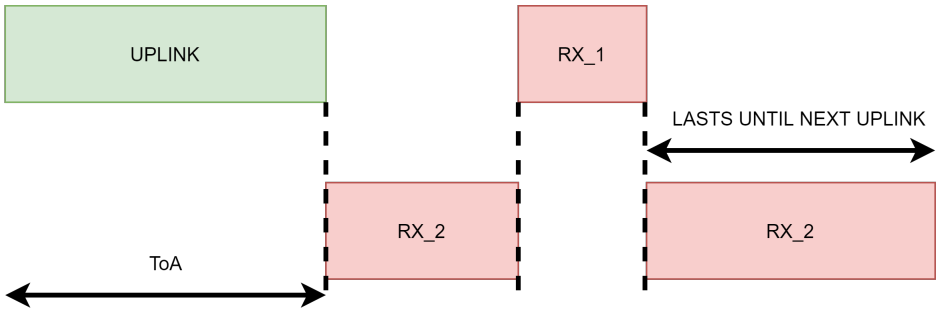


Figure 2.6: Class C receive window diagram.

bit rate, depends on the country and used frequency. In Sweden, 433MHz can be used since it is license-free. This thesis also uses LoRa modules that can only operate at 433MHz. The values can be seen in Table 2.1.

Table 2.1: The data rate for the EU433 band determined by the LoRa alliance [16].

Data rate	Spreading factor	Bit rate [bits/s]
0	SF12	250
1	SF11	440
2	SF10	980
3	SF9	1760
4	SF8	3125
5	SF7	5470

2.4.4 Default reliability in LoRaWAN

In Section 2.1.4, the general definition of reliability in networks was defined. This subsection defines the default reliability method specifically used in LoRaWAN. The LoRaWAN specification states that there should be two types of data messages; one that has to be acknowledged and one that does not. Because of this, a receiver has to respond to a data packet with the ACK bit set in the header. If the receiver is an end-device, this response can either be sent directly with an empty packet or piggyback its subsequent transmission. The gateways, however, will try to send an acknowledgment when the sender has a receive window open [21].

The default reliability method in LoRaWAN uses a field called `nb-Trans` that ADR sets. This field is the number of transmissions that should be done for an acknowledged, or non-acknowledged packet [21]. This field can assume the values between $[1 - 15]$. When retransmissions occur, the end-device should perform frequency hopping between retransmissions. That is, the end-device should use a new sub-frequency within its range between the retransmissions. After each retransmission, the device should wait to retransmit until the receive window has closed. Otherwise, the delay between the retransmissions is decided by the end-device and may

be different for different end-devices [25].

As mentioned in 2.4.1, the NS cannot send ADR instructions. It is up to the end-device's application layer to configure the spreading factor and Tx-power. Thus, the default reliability protocol can utilize SF, Tx-power, `nb-trans`, and retransmission delay with the application layer controlling these attributes.

2.4.5 Device join process

The end-devices need to be added to the LoRaWAN network before they can start using the network. This process is done using either the so-called Over the Air Activation (OTAA) or Activation by Personalization (ABP) [21]. However, these two differ in how they perform the join process to the network.

In OTAA, each end-device has two root keys that exist throughout the lifetime of the device. They are the Network Key, which is known to the NS, and the Application Key, which is known to the AS. They are used to establish the connection to the network and create the necessary session keys for that particular connection [28].

The end-devices start by communicating to the NS that they want to join the network. This is done with a join request message. This message contains which network the end-device wants to join (AppEUI), its unique identifier called DevEUI, along with a hash of the network key, join request, and the DevEUI. This request is then signed using the Message Integrity Code (MIC). With this, the NS can then forward this request to the JS, verifying the signed request and returning a join answer to the NS if the verification went well. This join-accept message contains all the necessary information for the end-device to generate the session keys needed.

In ABP, the end-devices instead has all of the required root and session keys, as well as the required information to join a specific network from the start [28]. However, this means that joining a non-predefined network can not be done on the fly. This is a more straightforward method than OTAA, but it makes the devices more restrained as to which network they can join.

2.4.6 Security

Butun *et al.* [29] discuss in their paper how the security in LoRaWAN improved from the initial release of version 1.0 to version 1.1. The underlying security in LoRaWAN 1.1 uses the 128-bit cryptography standard Advanced Encryption Standard (AES-128) and is applied on the network and application levels [30]. In addition, mutual authentication is performed on the network level. This means that all the devices connected to the network are authenticated and validated during the join process. LoRaWAN MAC and application messages are origins authenticated, ensuring that the traffic is legit and has not been tampered with. Furthermore, on the application layer, confidentiality is applied since LoRaWAN uses End-to-End (E2E) encryption on the payload sent from end-device to the application server.

Every end-device in the LoRaWAN network has its private key called an AppKey along with a globally unique identifier DevEUI [30]. Both of these are used during the authentication process of the end-device. When an end-device has successfully joined the LoRaWAN network, each message sent through it is encrypted. Two security steps are taken for each message. Firstly, the message payload from the end-device to the application server is encrypted using the Application session key, established during the join process. Secondly, a Message Integrity Check (MIC) needs to be performed. It is calculated using the encrypted message and the network session key, which is also generated during the join process. This makes sure that the message is protected against outside listeners.

2.4.7 Payload and headers

In this thesis, two types of message formats can be mentioned. The first one is the LoRaWAN specific, and the second one is Adafruit's message format. These will be discussed in this section.

2.4.7.1 Standard LoRaWAN

In the standard LoRaWAN network, there are two types of messages. All of these are defined by the LoRa Alliance in their specification document for version 1.1 of LoRaWAN [21]. The first one is an uplink message: the message sent from an end-device to the NS using the gateway as a relay. The other one is a downlink message: the message sent from the NS to the end-device using the gateway as a relay.

Both these messages contain the fields Preamble, Physical Header (PHDR), the Physical Header Redundancy Check (PHDR_CRC) as well as a PHYPayload. Table 2.2 shows the downlink message structure. For the uplink message, an additional field called Cyclic Redundancy Check (CRC) is added to protect the integrity of the message payload, which can be seen in Table 2.3.

Table 2.2: The downlink message fields.

Preamble	PHDR	PHDR_CRC	PHYPayload

Table 2.3: The uplink message fields.

Preamble	PHDR	PHDR_CRC	PHYPayload	CRC

The PHYPayload field in both of these messages contains a Message Header (MHDR) field, a data field that depends on the type of message sent, and a Message Integrity Check (MIC) field. Table 2.4 shows the PHYPayload message structure. There are three types of PHYPayload, and the MHDR determines these. The MHDR field, in turn, has a total of eight bits. Three are used to determine different message types,

three are reserved for future potential use, and two are used to signal the format of the messages. These can be seen in Table 2.5.

Table 2.4: The PHYPayload fields. The maximum MACPayload size is region specific. Join-Accept is already an encrypted message, so MIC is included in it.

Bytes	1	7..Maximum	4
PHYPayload type 1	MHDR	MACPayload	MIC
PHYPayload type 2	MHDR	Join-Request or Rejoin-Request	MIC
PHYPayload type 3	MHDR	Join-Accept	X

Table 2.5: The content of the MHDR field.

Bit	7..5	4..2	1..0
MHDR	Message Type	Reserved For Future Use	Major Version

The last three bits in the MHDR field determine eight different types of messages, which can be seen in Table 2.6. As can be seen, when comparing it with Table 2.4, the PHYPayload type one corresponds to the Confirmed and Unconfirmed message types. The difference between them is that Confirmed requires the receiving end to acknowledge the data. PHYPayload type two corresponds to bit values 000 and 110, and PHYPayload type three corresponds to bit value 001. As for bit value 111, which is the proprietary message type, it can be used as a non-standard message format. It is still required, however, that all devices understand the standard format.

Table 2.6: The eight different message types determined by the last three bits in the MHDR field.

Bit value	Message Type
000	Join-Request
001	Join-Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	Rejoin-Request
111	Proprietary

2.4.7.2 Adafruit CircuitPython RFM9x library

In this thesis, the LoRa module RFM9x is used. This module has its own MAC layer library made in Python[1], which is needed to use the module. This library has all functionality needed, with functions to send, receive and change parameters. In

Listing 2.2, the functions of interest are defined.

```
#Reliable Datagram mode
def send_with_ack(self, data)

def send(self, data, *, keep_listening=True, destination=
    None, node=None, identifier=None, flags=None)
#inputs of interest: flags

def receive(self, *, keep_listening=True, with_header=False
    , with_ack=False, timeout=None)
#inputs of interest: with_header, with_ack and timeout

def spreading_factor(self, val)

def rssi(self)

def snr(self)
```

Listing 2.2: Functions used from the Adafruit RFM9x library.

The library implements a reliability method called Reliable Datagram Mode (RDM). This can be utilized by the functions `send_with_ack` and `receive`, which can be seen in Listing 2.2. For the `receive` function, RDM is activated by setting `with_ack=True` whereas when sending, `send_with_ack` is simply called along with the data to send. The `send_with_ack` function follows the LoRa specification except that it does not use frequency hopping. The parameter `nb-trans` is by default set to five but can be changed if necessary. The `ACK_TIMEOUT` is set to 0.5 seconds.

Table 2.7: The 4-byte header used in the Adafruit library for the RFM9x module.

Bytes	1	1	1	1
Header-fields	Destination	Sender	Identifier	Flags

The function `receive` responds directly with an ACK if not specified otherwise. Thus, it does not follow the LoRa specifications. This is bypassed by instead handling the sending of an ACK in the application layer. By calling the `receive` function with `with_header=True`, the application layer can decide when to send the ACK, as this returns a read packet with its header and the flags in the header. The contents of the header can be found in Table 2.7, and the different flags can be found in Table 2.8. The input `timeout` to the function `receive` is used to specify how long the function should listen for a packet.

Table 2.8: The 1 byte flag header-field.

Bits	7-4	3-0
Flags	Proprietary	Reliable Datagram Mode

[1]https://github.com/adafruit/Adafruit_CircuitPython_RFM9x

2.5 Hardware

The hardware used in this thesis work consists of the SBCs Raspberry Pi 2B and 4. One of these SBCs can be seen as an end-device in Figure 2.7. These two SBCs have different specifications but essentially the same functionality. Pi 4 has 4GB of ram and a clock frequency of 1.5GHz, while Pi 2B has 1GB of ram and a clock frequency of 0.9 GHz [31]. Although there is a difference in RAM and core speed, this does not make a big difference since they all use the same LoRa module. The reason for choosing the Raspberry Pi's is because of their ease of use, and because of the GPIO pins, it is easy to connect external sensors and modules.

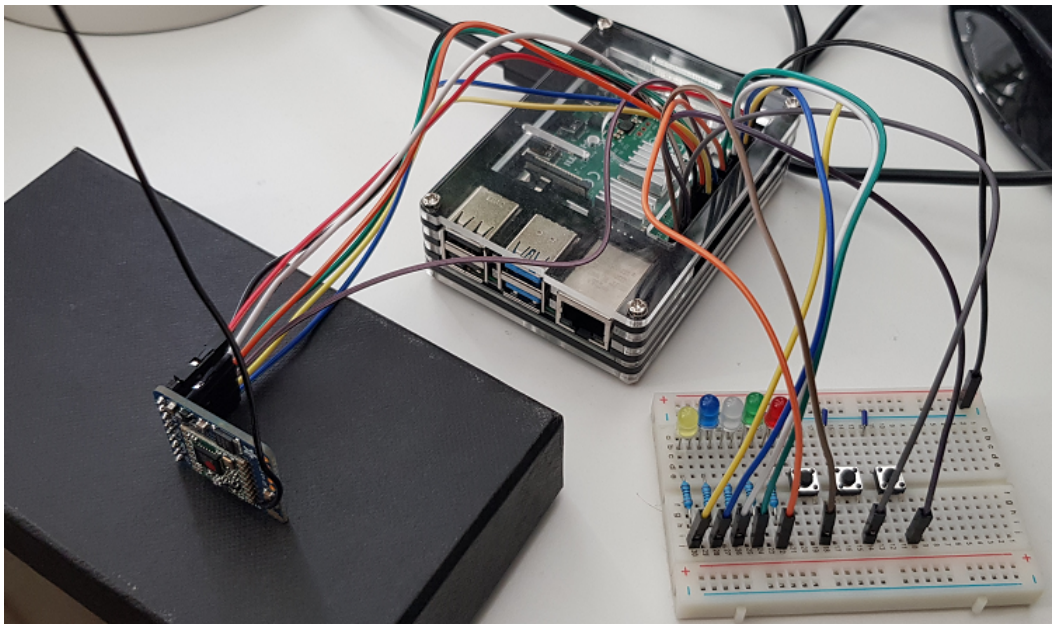


Figure 2.7: One of six end-devices. Here, the Raspberry Pi, the LoRa RFM9x module and the connector board are connected together.

The LoRa modules SX1276 with RFM9x radio chips are used for the LoRa communication. The modules have pins compatible with the Raspberry Pi's pins, making them easy to install and program. Furthermore, they are relatively cheap, making it a reasonable way to achieve LoRa communication quickly. See Figure 2.8 for an illustration of the module.

However, there are limits to these RFM9x modules: they only provide a single channel, meaning that the chip cannot send and receive data simultaneously. Therefore, both the end-devices and the gateways can only use one channel. This, in turn, means that the gateway cannot serve more than one end-device at any given time.

Apart from the Pi's and the LoRa modules, some basic electrical components are needed, like wiring, resistors, lights, and buttons. Once installed on a connector board, the boards can connect to the Pi's General-Purpose Input/Output (GPIO) pins. These lights and buttons are used for message checking and errors during the

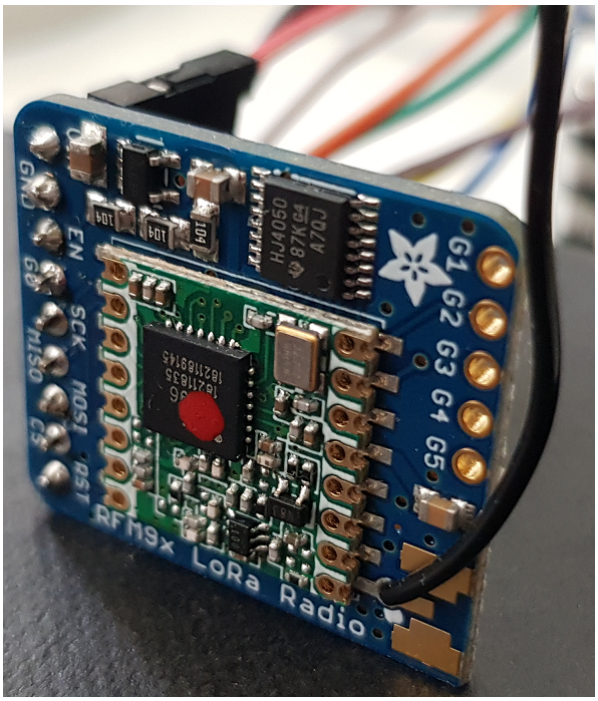


Figure 2.8: The RFM9x LoRa module.

transmission and receipt of messages.

2.6 Similar technologies

There are a few LPWAN technologies similar to LoRaWAN that are deployed and in use. Although they all are LPWAN technologies, there are differences in how they function. The following section will go through some of them.

2.6.1 SigFox

SigFox is a service that offers end-to-end (E2E) LPWAN connectivity by letting SigFox Network Operators deploy base stations. These base stations have IP capabilities and can therefore serve as a way for the end-devices to reach the backend. The devices in SigFox utilize the Binary Phase Shift Keying (BPSK) modulation technique through Ultra Narrowband (UNB). It operates in a SUB-GHz ISM band carrier, within the 863 to 870 MHz frequencies in Europe. Using UNB gives SigFox the ability to efficiently use bandwidth and experience very low noise levels. This, in turn, leads to low power consumption, high receiver sensitivity, and a more straightforward antenna design. However, there is a downside to these benefits since the data rate is limited to between 100 to 600 bps [32].

2.6.2 Weightless

Weightless is a set of LPWAN technologies made by a particular interest group that proposed three LPWAN standards. These three standards provide different

features, but all of them can operate in licensed spectrum as well as license-free [32]. Weightless has two variants of its physical layer: low data rates and high data rates. Some of the technologies and techniques that both variants use are spreading factor, forward error correction, interleaving, and PSK control. Transmission rates of the protocol can vary from 125 Kbps to 16 Mbps [33].

Weightless-N is one of the Weightless standards based around UNB for one-directional communication. This communication goes from end-devices to base stations and then to the backend. The utilization of UNB lets Weightless-N have a better power efficiency and lower cost than Weightless's other standards. Weightless-N utilizes a form of BPSK modulation technique called Differential-BPSK [32]. With a focus on lower energy consumption and smaller distances, Weightless-N has data rates ranging from 30 Kbps up to 100 Kbps [33].

However, if different bandwidths and signals are of interest, then Weightless-P could be an alternative. With several multiple access technologies, Weightless-P can achieve 4923 bps for Narrow Band (NB), 1404 bps for UNB, and finally 93 bps for spread spectrum. In addition, it does not require the Temperature Compensated Crystal Oscillator (TCXO) that the W and N standard requires. This makes P somewhat cheaper and less vulnerable to de-synchronization that is dependent on the TCXO [33].

Weightless-W works in the television white space spectrum (470-490Mhz) and is bidirectional communication technology. Using frequency hopping and TDMA, it can coordinate and separate the uplink and downlink intervals. As a result, the Weightless-W can achieve a data range from 1 Kbps to 10MB/s [33].

2.6.3 NB-IoT

Narrowband IoT (NB-IoT) is another popular LPWAN technology, which operates in the licensed frequency spectrum of LTE [5]. Therefore, it can utilize the same infrastructure as cellular. Since NB-IoT is an LPWAN technology, it is similar to LoRaWAN, but there are differences. Firstly, since it uses the same frequency spectrum as LTE, it uses Frequency Division Multiple Access (FDMA) for transmission and Orthogonal FDMA (OFDMA) when receiving. This is compared to CSS for LoRaWAN.

Secondly, NB-IoT does not use a duty cycle, meaning that if devices require frequent transmission and receiving, then NB-IoT is more suitable. However, being a synchronous protocol means that more power is consumed when communicating since it is full-duplex, and both ends have to synchronize between them. This, in turn, means that it is also more costly since the hardware requires more complexity [5]. On the other hand, the benefits are that it is a fast and reliable way of communication. LoRaWAN, on the other hand, is an asynchronous protocol.

3
Related Work

There has been much research on how to improve the reliability part of the LoRaWAN technology. This chapter will present some of the related work that is connected to this thesis. It will further mention the papers with reliability methods that are potential candidates for evaluation in this thesis.

3.1 Adaptive Latency Reduction in LoRa for Mission Critical Communications in Mines

Nessa *et al.* [6] propose a type of enhanced redundant retransmission scheme based on the original protocol in LoRaWAN. The original employs retransmission of the previous data frame if the end-device has not received an acknowledgment (ACK) from the receiver. If that retransmission fails as well, then the end-device increases its SF. However, this is not the optimal solution as the authors argue that it will reduce the data rate, and more packet loss will occur due to collisions.

The LoRa Alliance specifies that there should be a random delay called `ACK_TIMEOUT` [21] before retransmission, which does not depend on the air time of the previous transmission. The authors instead propose that the `ACK_TIMEOUT` should depend on the air time of the previous transmission. If an end-device does not receive an ACK, it waits for `ACK_TIMEOUT` before initiating retransmission of the frame twice using the same SF. The time between the first retransmission and the second one is a random time the authors call *"Arbitrary spacing between retransmission and redundant retransmission"* (`ASRTX_TIMEOUT`). If the end-device still has not received an ACK after these two retransmissions, it increases its SF by one and enters a new stage. This process is then done again until a set transmission limit or a maximum tolerable delay has been reached.

The authors used a simulator called LoRaSim when running their tests. They evaluated their solution based on the Data Extraction Rate (DER), another term for PDR, and the average delay per confirmed packet. The authors also simulated multiple end-devices, upwards of 500 end-devices in total. The author's simulations show that their proposed method reduces the average transmission delay significantly compared to the original LoRaWAN method. Furthermore, their DER is close to 100% even with an increasing number of end-devices. They also achieve substantially lower average latency compared to the original LoRaWAN method.

3.2 Allocation of Repetition Redundancy in LoRa

In Borkotoky *et al.'s* paper [34], the authors mean to improve LoRa sensor networks reliability without using ACKs. They propose sending several measurements in a frame. This results in several copies of sensor data being sent in different frames. How many measurements that go into a frame are decided with a strategy that considers interference and fading. Using the simulator LoRaSim, their proposed method was compared to a method that utilized maximum redundancy delay and duty cycling constraints. After this comparison, the authors concluded that their method achieves a 30% energy reduction to ensure sensor data is received by a gateway.

The authors cite that it is essential to look at redundancy without ACKs because LoRa end-devices of a specific class can only listen for ACKs during a particular time window. Because of this, a gateway can respond with an ACK, but the end-device is not listening. Furthermore, the authors argue that if many end-devices need ACK, this is infeasible. Thus they opt for a method that does not rely on ACKs.

3.3 High Reliability in LoRaWAN

Coutand *et al.* [35] propose in their paper a protocol that enhances the Adaptive Data Rate (ADR) algorithm in LoRaWAN, which they argue will make LoRaWAN more reliable. As mentioned in 2.4.1, the ADR exists to change different parameters for optimal transmission dynamically. The authors first evaluated the legacy version of ADR in LoRaWAN, and with those results, then created and evaluated their enhanced ADR method. The authors used a physical testbed and ran tests while randomizing the different transmission parameters like Tx-power, SF, and channel frequency.

The authors evaluated their result based on how SNR and Tx-power affect the Packet Delivery Rate, which showed that although Tx-power and SF significantly impact reliability, they can not be relied on for everything. A maximum of 85% in Data Delivery Rate (DDR) was obtained during these tests. To enhance the legacy ADR, the authors used more precisely calculated values for the PDR boundaries than those used in the legacy ADR algorithm. This, along with Forward Error Correction (FEC), yielded the authors a maximum DDR or 98%.

3.4 Improving Reliability and Scalability of LoRaWANs Through Lightweight Scheduling

Reynders *et al.* [36] propose in their paper a new MAC layer for LoRa, called RS-LoRa. It is a distributed and two-step lightweight scheduling method that is done in two steps. Firstly, the gateway schedules nodes using a coarse-grained manner which specifies the Tx-power and SF end-devices can use. Secondly, the end-devices

will then use this information to set their own Tx-power, SF, as well as when and on which channel it can transmit on. Finally, all end-devices are then grouped based on their Tx-power to better handle the capture effect, which is the phenomenon that only the strongest signal gets demodulated at the receiver. The authors claim that the overall network reliability and scalability are increased by doing this due to better capture effect and reduced packet loss.

The authors evaluated their method in the network simulator NS-3, in which they created their module for the new MAC layer RS-LoRa. Their result shows that their method can reduce Packet Error Rate (PER) by up to 20%. However, since a big part of their method is also about increasing the scalability of LoRaWAN, this method is difficult to evaluate on a physical testbed. Nevertheless, the NS-3 module is a good complement to test implementations since the authors have also implemented a legacy version of LoRaWAN and made it open source.

3.5 DaRe: Data recovery through application layer coding for LoRaWAN

Marcelis *et al.* [37] propose a redundancy method in their paper, that sends previous data with every frame. The amount of redundant data in the frame is decided by the coding rate. The redundant data is selected with fountain coding in combination with convolution coding achieved with sliding window size. The authors argue for this solution because they want gateways to remain unchanged and avoid downlink communication with acknowledgments.

The author's method was tested in three different scenarios: end-devices on cars, bicycles, and stationary end-devices. ADR was active during their data collection. However, 95% of their data was collected with a spreading factor of 12. The max distance to a gateway for all of the scenarios was 7.5 km. After the authors had analyzed their data, they claim that their method can successfully recover 99% of all lost data. This is achieved with a coding rate of 1/2 and when transmissions had a 10% erasure probability [37].

3.6 Conclusion

Following the related work, it can be concluded that there has been much work done to make LoRaWAN a more reliable protocol. There are plenty of different methods available that can be evaluated. The proposed method by Nessa *et al.* is the one that was evaluated in this thesis. This method was picked because it satisfies all of the requirements defined in Table 4.1 in the coming chapter. It was also possible to compare their results on the simulator with this thesis's results on the physical testbed, which will be covered later on in Chapter 6.

4
Methods

This chapter will explain the different methods related to this thesis. The first section will go through methods of finding reliability methods in literature and the criteria for them to be relevant for evaluation. The second section will explain how the physical testbed was designed and set up and how the telemetry for the data collection was collected. The chapter then concludes with the evaluation methodology for the physical testbed and the testing environments in which the tests were performed.

4.1 Finding alternative reliability method for LoRaWAN

The first step to find a suitable reliability method that could potentially be implemented in LoRaWAN was to look at published articles from authors within the network field. IEEE and Google Scholar are good platforms for this since they are easy to use and search for relevant keywords. The words searched for were general network-related ones relevant to prior university courses and the relevant words already mentioned in this thesis, like latency, PDR, PAR, congestion, and reliability.

The second step was to sift out the most relevant papers from all the possible ones found. To do this, a set of requirements had to be determined. These requirements that determine whether or not a method was worth evaluating can be seen in Table 4.1. A method had to fulfill all these requirements to be evaluated.

Table 4.1: The requirements that a possible reliability method had to fulfill.

R1	Is the method implemented in LoRaWAN or in another technology?
R2	Does the method in question improve latency when reliability is active?
R3	Does the method according to its source improve the reliability of LoRaWAN?
R4	Is it a reasonable method for implementation given the time frame?
R5	Is the reliability method targeted for specific types of end devices?
R6	Besides being feasible analytically, the method should be shown to be deployable and able to achieve the desired goals in actual system settings.

As mentioned in Section 3.6, the method made by Nessa *et al.* is the method that was evaluated in this thesis. After studying the author's paper, it checked all of the requirements in Table 4.1. The paper is about reducing the latency in LoRaWAN using a redundant retransmission scheme. Their simulation test results show that the method improves both the latency and the overall reliability for LoRaWAN.

The method did not seem to be an overly complicated solution, meaning that it would be possible to implement it in a reasonable time frame. The authors implemented their reliability method on a simulator. However, investigating a possible implementation on a physical testbed, it was concluded that it would be doable.

Another method mentioned in Section 3.6 was Coutandet *et al.'s*, which also seemed to fulfill all the requirements upon studying their paper. However, they are using ADR in their method, which would be a large time investment to implement. Therefore, it did not fulfill the requirement **R4**.

Reynders *et al.'s* proposed method has a large emphasis on the scalability of nodes in the LoRaWAN network. Furthermore, since the authors have implemented a whole new MAC layer in the NS-3 simulator, it would take too long to implement another whole MAC layer on the physical testbed. Hence, it did not fulfill the requirement **R4**.

Marcelis *et al.'s* proposed method is a relatively simple yet effective method, where sending the previous data with every subsequent data frame. However, their testing method involves ADR, which would be a too big time investment to implement for this thesis. Hence, it did not fulfill the requirement **R4**.

In Borkotoky *et al.'s* proposed method, the authors argue about the infeasibility of using ACKs in LoRaWAN. However, the author's method does not take latency into account, meaning that there is no way of knowing whether their method improves on latency or not before implementation. Hence, it did not fulfill the requirement **R2**.

4.2 The design of the testbed

To evaluate the default reliability method and the alternative as realistically as possible and to fulfill the aim posed in Section 1.1, a physical testbed was needed. Therefore, a whole LoRaWAN infrastructure had to be implemented. The following subsections will go through each of the different components of the testbed used in this thesis.

4.2.1 Azure IoT Central as back-end

When the tests had been performed, a suitable solution was needed to handle the collected telemetry data and display the information needed, such as the PDR, PAR,

latency, and other measurements. In this work, Azure IoT Central was used. This was used to show that it was both doable for LoRa devices to be connected to the cloud and an effective yet simple way to display data. An example of how data can be displayed can be seen in Figure 2.2, and also in Figure 4.1 where the same data is displayed in four different ways.

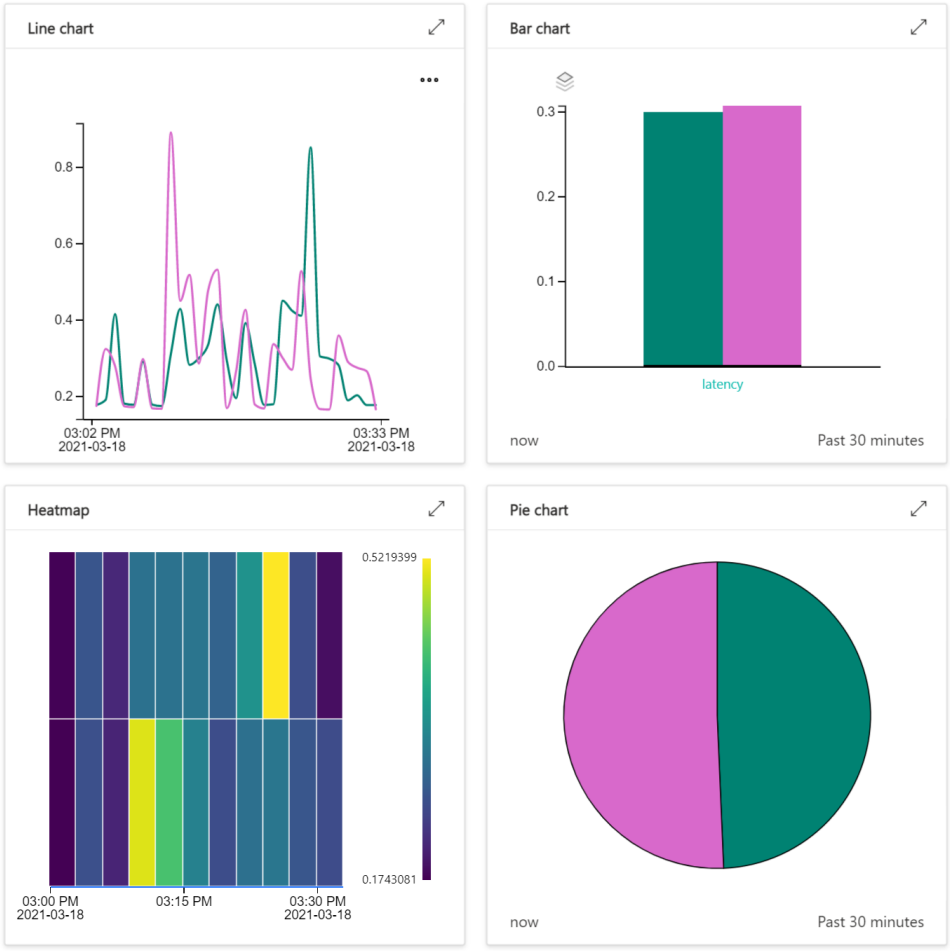


Figure 4.1: An example of how Azure IoT central can be used to display the same data in different ways.

When the small LoRa network was connected to Azure, it was believed that it had to go through The Things Network. However, this step could be bypassed entirely by using an Azure IoT Central device bridge and Azure IoT Central, which was discovered early on in the development phase. This made it possible to set up an Azure function that could then be used to post data to Azure IoT Central. As soon as the function receives a data request, the data will be forwarded to Azure IoT Central, which can then be displayed. Once data could be uploaded, functionality for this was implemented into the physical testbed to collect valuable sensor telemetry data. After this, device templates were created to fit the properties of the end-devices.

4.2.2 The set up of hardware

In order to set up the physical testbed, a few things had to be done. Firstly, the LoRa modules did not have compatible pins with the GPIO pins on the Pis installed from the factory, which meant they had to be soldered before use. The RFM9x module has a total of nine pins. Three are power pins, and the remaining six are used for the Serial Peripheral Interface (SPI). All pins serve a different purpose, defined as follows:

- **Voltage Input (VIN):** Is the power input. This supports 3.3 to 6.0 V Direct Current (DC) and has a peak current of 150 milliamperes (mA).
- **Ground (GND):** Ground pin needed for the logic and power.
- **Enable (EN):** Enables the regulator on the module and can be used as a power switch of the module. It is set to high for the VIN by default, and if it is set to low, it cuts the power to the module.
- **G0:** Is the module's GPIO 0 pin and is used for the interrupt request notification from the radio to the connected microcontroller. In this case, to the Raspberry Pi.
- **Serial Clock (SCK):** Used as an input pin to the chip.
- **Microcontroller In Serial Out (MISO):** Used for data sent from the module to the connected microcontroller. In this case, to the Raspberry Pi.
- **Microcontroller Out Serial In (MOSI):** Used for data that is sent from the connected microcontroller to the module. In this case, from the Raspberry Pi.
- **Chip Select (CS):** Used as an input pin to the chip. It needs to be set to low to start an SPI transfer.
- **Reset (RST):** Used as a reset pin for the module. It is set to high by default which translates to reset. If it is set to low, it turns on the module.

.
As for the Raspberry Pi's, they have 40 GPIO pins divided into two rows. The corresponding pins between the LoRa module and the Pis can be seen in Figure 4.2.

In addition to the pin soldering, an antenna also needed to be soldered on. For this, a basic copper wire was used. The quality and type of antenna are important factors when taking measurements of RSSI and SNR since they can affect the total ERP. However, for this thesis, a basic quarter-wave whip antenna is sufficient due to the compact test environments, which will be covered in Section 4.3.2. Furthermore, the length and positioning of the wire are important. For a module operating at

433 MHz, the length of the wire can be calculated from the following equation:

$$\frac{wave\ velocity\ in\ air\ (m/s)}{frequency\ (Hz)} = wavelength\ (m) \tag{4.1}$$

Since the antenna is a quarter-wave antenna, 1/4 of the wavelength is sufficient:

$$\frac{wave\ velocity}{frequency\ *\ 4} = 1\ /\ 4\ wavelength \tag{4.2}$$

This then yields the following length of the wire:

$$\frac{299792458}{433000000\ *\ 4} \approx 0.173\ m = 17.3\ cm \tag{4.3}$$

From [38], it is recommended to use an approximated length of 16.5 cm which was used in this thesis. As for the positioning of the wire antenna, the ideal way is to have it pointed upwards for the best overall coverage.

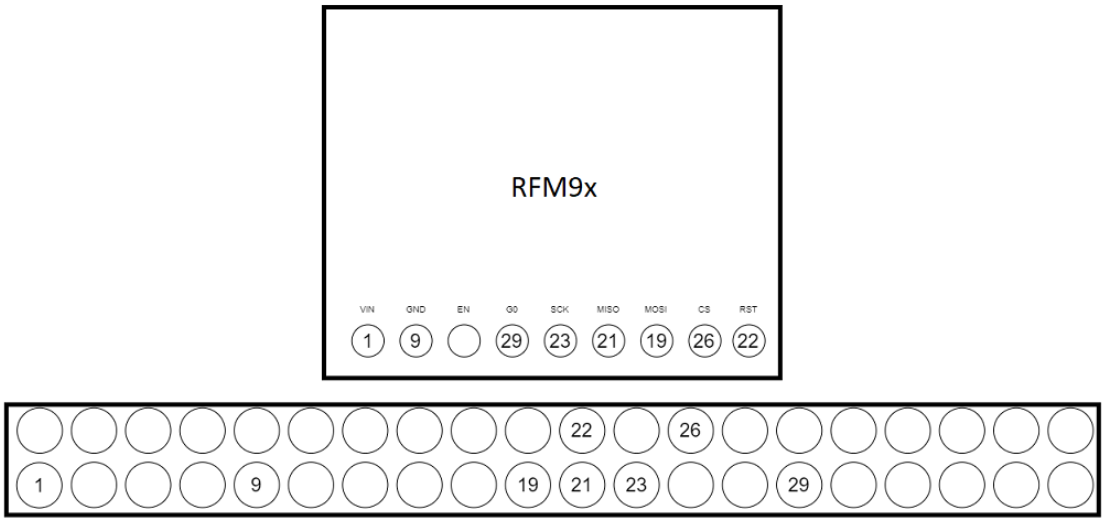


Figure 4.2: The pin-outs on the LoRa module RFM9x and the Raspberry Pis.

In order to have an easy way of knowing that things are functioning the way they should, a connector board with lights and buttons was set up. For this, an electrical kit from ELEGOO was used that contained everything needed. Each Pi has its connector board with three buttons that can easily be programmed to do different tasks if needed. There are also five lights used as different indicators when the devices are operational.

4.2.3 The testbed infrastructure

As mentioned in Section 2.5, the whole testbed consists out of a total of six Pi's; five Pi 4 and one Pi 2B. Each is equipped with its own LoRa module and its connector board with lights and buttons. This solution is a specifically designed infrastructure based on the budget available. It would have been possible to go with a pre-built solution, but it would have been harder to make the necessary changes. The evaluated alternative method is supposed to be easy to implement to real production versions later on. All the parts of LoRaWAN that were implemented and were not implemented will be covered more in-depth in the next chapter.

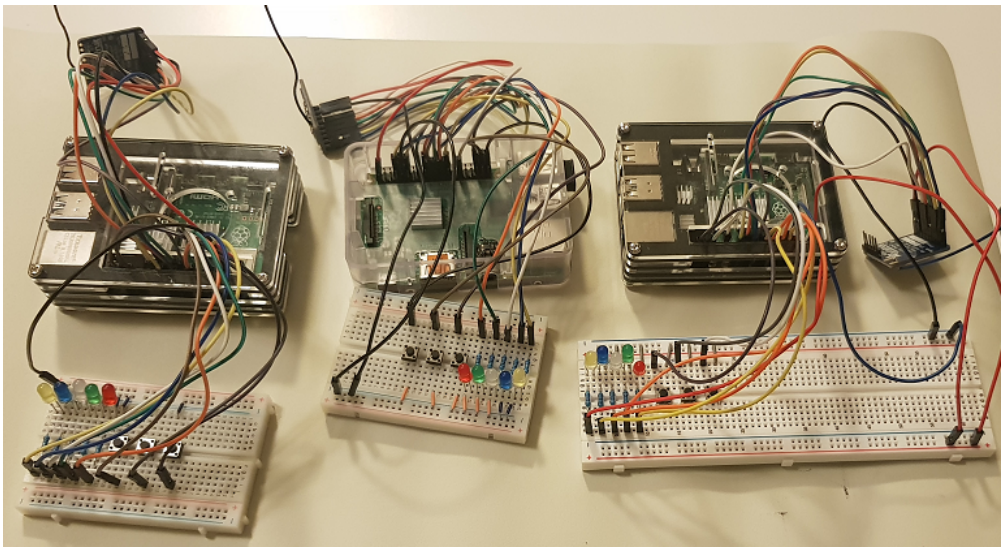


Figure 4.3: One of the two testbeds containing two Pi 4 and one Pi 2B.

A set of two testbeds was used on two different locations with different environments while still having the possibility of combining them later on if that would be desirable. One of the testbed setups can be seen in Figure 4.3. A custom protocol solution for the gateway and the end-devices was implemented, based on Adafruit's library[2]. This customization was necessary so that the end-devices could communicate with the gateways and vice versa and with Azure.

The end devices will first send a form of join request to the gateway, where the message payload can be seen in Table 4.2. If successfully added, the gateway will respond with an ACK. If the end-device receives the ACK, it can then start sending sensor telemetry to the gateway, which can be seen in Table 4.3. The gateway will receive and decode the message and then relay it to the Azure function via HTTP post requests.

[2]https://github.com/adafruit/Adafruit_CircuitPython_RFM9x

Table 4.2: The message payload that an end-device sends when trying to join the gateway. `S-Freq` is the sending frequency, i.e., the speed at which the end-devices sends telemetry data.

Bytes	4	8	8	8	8	8	8
Add-to-gateway	Header	Host	DeviceID	"ADD"	Tx-power	SF	S-Freq

Table 4.3: The message payload of the telemetry data that the end-devices send. `TransCount` is the transmission counter of that particular message, and the `Time` is the timestamp at which the message was sent.

Bytes	4	8	8	8	8	8
Telemetry	Header	DeviceID	CPU-temp	CPU-freq	TransCount	Time

4.2.4 Data collection

The data collection will be done on the metrics mentioned in Section 2.1. The latency of each message will be measured using the time since epoch method. The end-device takes the time exactly when the data is about to be sent and appends that to the payload. The gateway then decodes the payload, takes a new timestamp, and takes the difference between them. This then gives the latency for each message in seconds, from the actual time the message was sent through the air to when the receiver actually received the data, i.e ToA or ToF.

For measuring packet loss, the end-device sends with each message a counter which is incremented for each send. This is the total number of messages sent. The gateway also keeps a message counter for each end-device, which is incremented if it successfully receives a message from an end-device. This is the total of received messages. If a packet is lost during transmission from the end-device, then these two counters will differ, and a % of packet loss can be calculated. This is what is called Packet Delivery Rate (PDR).

The end-device also keeps a counter for acknowledgments, which it increments when it successfully receives an ACK from the gateway. This counter is always contained in each message sent by the end-device. Once the gateway receives the counter, it can calculate a fraction. This fraction is what is called Packet Acknowledgment Rate (PAR). Lastly, the RSSI and SNR values will be collected. They will, however, not be used for the evaluation of the reliability method. Instead, they will be used to see how the overall signal quality differs between the respective reliability methods.

4.2.5 Data logging

All the data results will be uploaded to Azure, but they will also be logged and stored locally on the gateways. There are a few reasons for this. The first one is to have a backup of all the tests in separate files if there is a problem with the HTTP post requests up to Azure.

Another reason is, although Azure is good for displaying data fast, there is no possibility to choose the time frame of the data that should be displayed. This means that Azure is limited to displaying the latest test data that has been sent to it, even though it has stored all the previous test data. Therefore, to display all the test data when all tests have been run, the log files are essential. However, with what was mentioned in Section 4.2.1, it is proven that Azure is still a Proof-of-Concept (POC) for the possibility of a cloud-based implementation for LoRaWAN.

When the log files were in place, it was easy to extract the relevant data fields from the log files by using the Matplotlib library for Python. It is a large plotting library designed for Python to make it easy to visualize large amounts of data in different kinds of ways [39]. First simple graphs were made for latency on each log file. However, it was decided that scatter plots, box plots, and violin plots would be a better alternative since it is easy to compare different samples. These plots are displayed in Chapter 6. Plots of latency, PDR, PAR, RSSI, and SNR were made using these types of plots.

4.3 Evaluation methodology

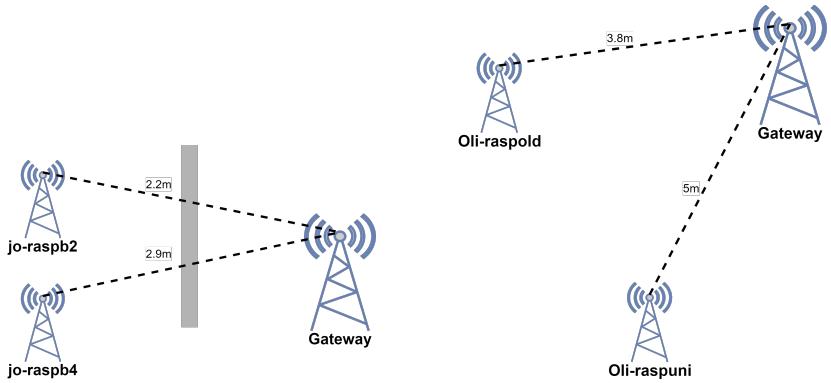
This section will define how the evaluation of the methods was done regarding the metrics defined. In addition, it will give an overview of the methodology for the physical testbed and test environments in which the tests were performed.

4.3.1 Physical testbed

The evaluation of the default reliability method and the alternative was done in a few steps. All of the Pi's have Secure Shell (SSH) enabled to remotely control and execute the respective program for the end-devices and the gateways. Both methods were evaluated the same way but with the duty cycle as the changed parameter. This was because, since each physical testbed only contained two end-devices, there needed to be some way to create congestion on the data link.

As mentioned in 2.4.3, the duty cycle exists to reduce the risk of messages colliding. However, to evaluate the reliability methods, the duty cycle needed to be high to simulate many end-devices sending to the gateway. This, in turn, was to be able to evaluate each method on how they coped with congestion and therefore see how latency, PDR, and PAR were affected. The RSSI and SNR values were taken to see whether or not the signal quality became affected between the two methods.

The tests involved both end-devices sending a fixed amount of 1000 messages containing telemetry data to the gateway, with Tx-power and SF set the same on all devices. In order to handle data fluctuation, each test was performed five times in the same testing round. Furthermore, the average between the two devices for all five tests was calculated.



(a) Testbed 1 **(b)** Testbed 2

Figure 4.4: The two testbeds and the distances that the end-devices had to their gateway.

This testing method was then performed on the respective implemented reliability method. The same tests were performed on the two separate testbeds, which yielded two separate data results to analyze later and compare. With the results at hand containing the obtained data points, it will then be possible to draw conclusions and answer this thesis's aim, mentioned in Section 1.1.

4.3.2 Test environments

For the evaluation, two different environments were used on two separate locations, which can be seen in Figure 4.4a. The first environment was an apartment with two rooms and a 15cm thick concrete wall separating them. The gateway was located in one room, and then the two end-devices were located in the other room. With the wall between, the end-devices had NLOS to the gateway. How this affected the signal when running the tests can later be seen by the RSSI and SNR values.

The second environment was an apartment with an open floor plan as can be seen in Figure 4.4b. The gateway was located at one end of the room, and the two end-devices were located at separate corners on the other side. This meant that all end-devices had LOS to the gateway. How this affected the signal when running the tests can later be seen by the RSSI and SNR values.

5
Design and Implementation

This chapter will present the work process of implementing the custom protocol for LoRaWAN. It will go through the whole implementation process. Thus, this chapter works as an in-depth version of Section 4.2. The first section will start by showing wherein the network stack implementation has been performed. The second section will then cover the underlying custom protocol for the physical testbed, including Azure. The third section will go through the implementation of the alternative reliability method. The fourth section will go through how the two methods differ from each other in terms of functionality. Finally, the last section will cover different challenges that came up during the implementation phase.

5.1 The network stack

To get a better picture of where in the LoRaWAN protocol the implementations took place, understanding the network stack helps. The entire network stack of this thesis is that of the gateways, the end-devices, and Azure. A figure of all the layers can be seen in Figure 5.1. The stack starts with the application layer in the end-device. This is because the telemetry data generated here should end up in Azure. After this layer, the telemetry data generated need to be transmitted reliably to the gateway. Thus the application layer communicates with the MAC-2 layer to send the data. This layer implements the reliability methods by using the MAC-protocol methods defined in the MAC-1 layer. This layer, in turn, communicates directly to the physical layer and modulates the data into radio signals.

When the gateway receives these radio signals at its physical layer, it demodulates the radio signals received back into data bits, which are then returned to the layer above. The layer above is the MAC-1 layer which is utilized by the MAC-2 layer. Thus, the data is passed from the physical layer to the MAC-1 layer, which is passed to the MAC-2 layer.

From the MAC-2 layer, the telemetry data then needs to be sent to Azure through HTTP post requests. This means that the gateway needs to use TCP/IP to forward the telemetry data up to Azure. More importantly, however, is that in the MAC-2 layer, an ACK is created to be sent to the end-device that sent the data. Thus communication occurs back and forth between the two MAC-2 layers on the gateway and the end-device, respectively. This can be seen in Figure 5.1 by following the black arrows between the layers.

To conclude, the telemetry data is generated at the end-devices, which are then delivered to Azure in a reliable way made possible by the MAC-2 layer. That layer makes sure that the data is both sent and acknowledged.

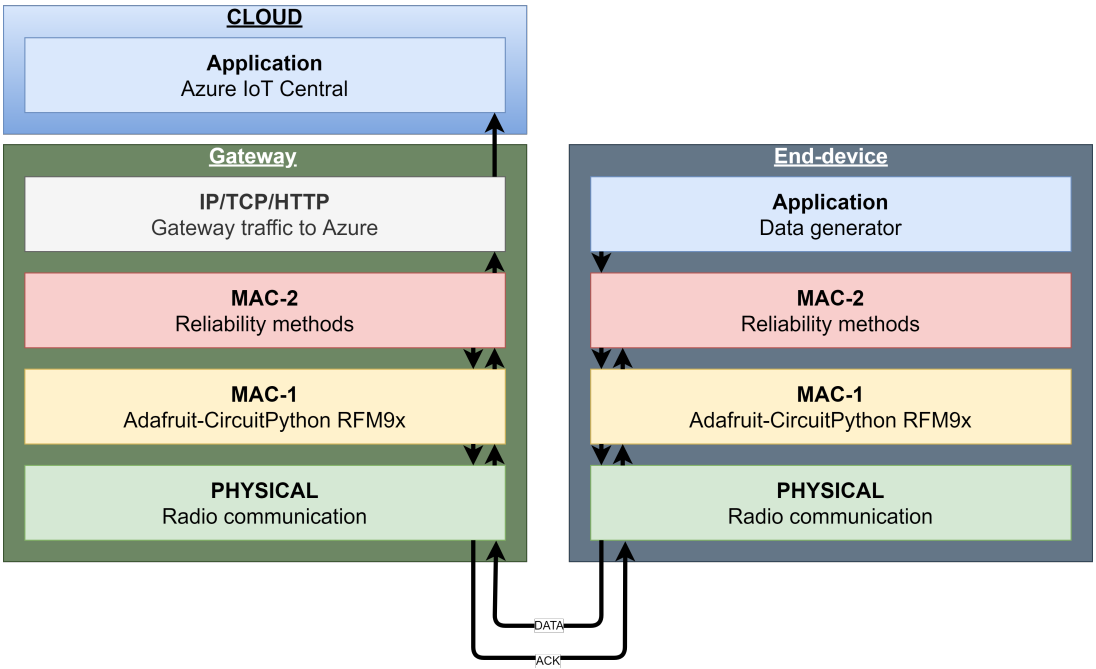


Figure 5.1: The network layers in the gateway as well as the end-devices. Important to note is that the reliability is between the MAC-2 layers. The arrows illustrate this communication.

5.2 Implementing the underlying custom protocol

As mentioned in Section 2.4.7.2, the LoRa module RFM9x has a pre-built python library[1]that implements a MAC-layer protocol. This library, however, does not implement the MAC-layer protocol as the LoRa alliance has specified it. Because of this, it had to be decided if the entire MAC protocol or parts of it would be implemented. It was decided that only the most relevant parts for this thesis would be implemented. The following subsections go through which parts were not implemented and which ones were.

5.2.1 Parts not implemented

The following parts of the LoRa specification were not implemented:

[1]https://github.com/adafruit/Adafruit_CircuitPython_RFM9x

- Class types
- Adaptive Data Rate (ADR)
- Network server
- Join server
- Security

Firstly, class types were not implemented because the only thing it was considered to add to the evaluation was when the different class types could respond to acknowledged traffic with an ACK. Since the evaluation would take place between end-devices and a gateway, it was decided that class types would be skipped.

Secondly, as mentioned in Section 2.5, the used LoRa module RFM9x is a one-channel chip. Therefore, the gateway is unable to listen to different frequencies and spreading factors at the same time. Because of this, Adaptive Data Rate (ADR) was not implemented since the gateway is unable to listen to different end-devices with different ADR settings. Furthermore, the evaluation would be on a small network. All devices would be wall connected, so the dynamic features of ADR would not yield any useful functionality to the testbed.

Thirdly, having the gateway be the last step before posting data to Azure was the best choice for the physical testbed. With this, however, the gateway needed to keep track of which end-device sent what data. Because of this, the gateways needed to know and keep track of all of their end-devices. Thus it became a simple join-server as well, albeit a custom one.

Lastly, implementing the security part of LoRaWAN would be another task on its own. The security aspects in LoRaWAN have to do with the join-process to the network. Since this was decided not to be implemented, then the security aspect was not implemented either.

5.2.2 Parts implemented

The following parts were implemented for the underlying custom protocol:

- Sending and receiving data
- Join-to-gateway
- Default reliability method
- Azure integration

- Data collection
- Data visualization
- Alternative reliability method

With the parts not implemented clarified, the first step in the implementation phase was to analyze the MAC-layer library for the LoRa module. It implements all the basic functionality needed to get the LoRa module working to send and receive messages. Once data could be sent and received between the end-devices and the gateway, functionality for different message types and the custom join-process was added.

The gateway needed a way to keep track of its connected end-devices to handle the telemetry data that the end-devices send. Therefore, when each end-device starts, they send a custom packet shown in the previous section in Table 4.2. The gateway then reads this packet, checks if it has already added this end-device or not, and responds with a reply whether or not the end-device got added. If successful, the end-devices wait a certain random amount of time to comply with ALOHA before sending telemetry to the gateway.

With the custom join procedure between the end-devices and the gateways in place, the next step was to implement a way to keep track of the different telemetry data an end-device had sent. The first iteration of this involved simply sending the CPU frequency and the CPU temperature of the respective Raspberry Pi. When the gateway could successfully extract this data, more essential telemetry data was added. Firstly, the number of messages an end-device has sent. This meant that the PDR could be calculated at the gateway since it records the number of received messages. The RSSI and SNR values of each uplink transmission were included as well.

Secondly was the latency of each message. Using the epoch method, this was done by taking a timestamp before sending a message along with this timestamp in the payload to the gateway. The gateway could then calculate the latency by taking a new timestamp when receiving the message and take the difference between the two. Lastly, the end-devices started recording how many acknowledgments they had received from their gateway. Thus the last telemetry to be added to the payload was PAR.

The next step was to integrate Azure with the physical testbed with the purpose of both displaying data and as a Proof-of-Concept for cloud-based LoRaWAN. The integration started with an article[2]on how to connect LoRaWAN through The Things Network to Azure. An IoT Central was created, as mentioned in the article. However, following the other steps in the article, the solution was ambitious for this thesis purpose with Azure. This resulted in researching for an alternative solution. The result was finding Azure IoT Central Device Bridge. This bridge made it possible to make an HTTP-POST request to post all the collected telemetry data to

Azure. The final JSON format used for posting can be seen in Listing 5.1.

```
json_data = {
                "device": {
                    "deviceId": host
                },
                "measurements": {
                    "cpuTemp"               : cpu_temp,
                    "cpuFreq"               : cpu_freq,
                    "transmissions"         : transmissions_host,
                    "rssi"                  : int(rssi),
                    "snr"                   : int(snr),
                    "latency"               : latency,
                    "transmissionsReceived": received_transmissions
                       ,
                    "transmissionsACKS"     : receivedACKS
                }
            }
```

Listing 5.1: Final JSON-format for the HTTP-post request to the Azure function.

The final step before implementing the alternative reliability method was to add the data logging and visualization. Since Azure became a Proof-of-Concept, there needed to be another way of visualizing the telemetry data. Hence, a local data logging method was implemented where all the devices on the physical testbed log and saves everything they do each time they run their respective program. However, the log files from the gateway are the important ones since it logs all the telemetry it receives from its connected end-devices. The gateway is also the device responsible for posting the data to Azure, so it made sense to log all the important telemetry data on the gateway. See Figure 5.2 for an illustration of a log file.

When all tests had been run and all the log files were in place, the final step was to script a solution for extracting all the data fields from the log files. This was done with the Python library Matplotlib. Each row has the end-device name before the telemetry data, which helped with categorizing each end-device's telemetry data. By looping through every log file, saving each telemetry field for each end-device into their own lists, the results for every telemetry data could then be plotted.

```
2021-04-27 : 17:39:18 - host: oli-raspuni, trans-sent: 536, trans-rec: 535, receviedACKS: 488, latency: 0.161837, rssi: -46, snr: 6.000000
2021-04-27 : 17:39:18 - host: oli-raspuni, trans-sent: 537, trans-rec: 536, receviedACKS: 489, latency: 0.161517, rssi: -47, snr: 6.000000
2021-04-27 : 17:39:18 - host: oli-raspuni, trans-sent: 538, trans-rec: 537, receviedACKS: 490, latency: 0.162410, rssi: -47, snr: 6.000000
2021-04-27 : 17:39:18 - host: oli-raspuni, trans-sent: 539, trans-rec: 538, receviedACKS: 491, latency: 0.161551, rssi: -42, snr: 4.000000
2021-04-27 : 17:39:19 - host: oli-raspold, trans-sent: 419, trans-rec: 376, receviedACKS: 340, latency: 0.374141, rssi: -54, snr: 6.000000
2021-04-27 : 17:39:19 - host: oli-raspold, trans-sent: 420, trans-rec: 377, receviedACKS: 341, latency: 0.165072, rssi: -54, snr: 6.000000
2021-04-27 : 17:39:19 - host: oli-raspold, trans-sent: 421, trans-rec: 378, receviedACKS: 342, latency: 0.163080, rssi: -54, snr: 6.000000
2021-04-27 : 17:39:20 - host: oli-raspold, trans-sent: 422, trans-rec: 379, receviedACKS: 343, latency: 0.164105, rssi: -54, snr: 6.000000
2021-04-27 : 17:39:20 - host: oli-raspold, trans-sent: 423, trans-rec: 380, receviedACKS: 344, latency: 0.162929, rssi: -55, snr: 6.000000
2021-04-27 : 17:39:20 - host: oli-raspold, trans-sent: 424, trans-rec: 381, receviedACKS: 345, latency: 0.165991, rssi: -55, snr: 6.000000
2021-04-27 : 17:39:20 - host: oli-raspold, trans-sent: 425, trans-rec: 382, receviedACKS: 346, latency: 0.163519, rssi: -55, snr: 6.000000
2021-04-27 : 17:39:21 - host: oli-raspold, trans-sent: 426, trans-rec: 383, receviedACKS: 347, latency: 0.167134, rssi: -54, snr: 6.000000
2021-04-27 : 17:39:21 - host: oli-raspold, trans-sent: 427, trans-rec: 384, receviedACKS: 348, latency: 0.163689, rssi: -54, snr: 6.000000
2021-04-27 : 17:39:21 - host: oli-raspuni, trans-sent: 540, trans-rec: 539, receviedACKS: 491, latency: 0.161916, rssi: -47, snr: 6.000000
2021-04-27 : 17:39:22 - host: oli-raspuni, trans-sent: 541, trans-rec: 540, receviedACKS: 492, latency: 0.161900, rssi: -47, snr: 7.000000
```

Figure 5.2: Illustration of a log file.

[2]https://azure.microsoft.com/sv-se/blog/the-things-network-and-azure-iot-connect-lorawan-devices/

5.3 Implementing the alternative reliability method

When the underlying protocol had been implemented, the implementation of the alternative reliability method could start. The method from Nessa *et al.* was the chosen method, as mentioned in Section 4.1. The underlying protocol was made to make it easy to add additional methods to it without making significant changes in the code. To easily switch between the original method and the alternative method helps when performing the data collection. There are, however, a few changes made to Nessa *et al.'s* method to be compatible with this thesis's physical testbed, which will be covered.

The first step was to analyze how the authors have defined their timing diagram for uplink and downlink windows, which can be seen in Figure 5.3. The overall functionality of their implementation has been discussed in Section 3.1. The authors have added redundant retransmission for the transmission and receiving part of LoRaWAN, divided into stages. These are different from the original LoRaWAN protocol, but they can be seen as more additional parts rather than a total change of the original. Realizing this made the initial implementation process more manageable.

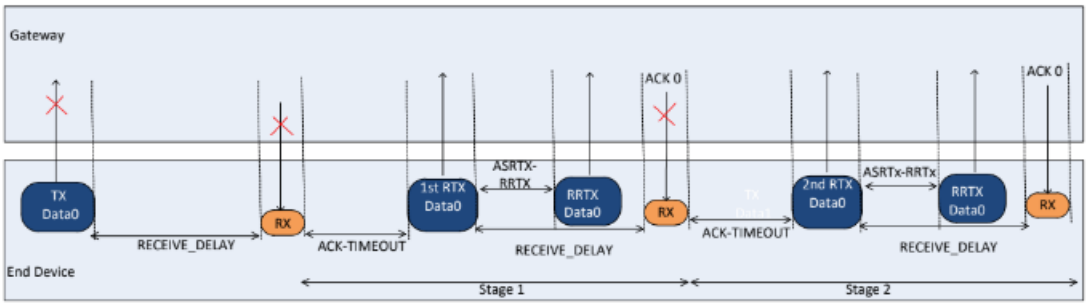


Figure 5.3: Nessa *et al.'s* timing diagram. Image from their paper [6].

The second step was to analyze and understand how the authors have defined their bounds on the different timeouts used in their timing diagram. The main parameters are the `ACK_TIMEOUT` and the Arbitrary Spacing Between Retransmission and Redundant Retransmission `(ASRTX_TIMEOUT)`. The `ACK_TIMEOUT` is dependent on the ToA of the previous transmission, meaning that the latency of the previous transmission determines the `ACK_TIMEOUT`. There needs to be an initial value, however, which the authors set to four. Over time, if there are no successful transmission and the contention stage increases, so will the `ACK_TIMEOUT` exponentially based on the ToA value (latency). The `ACK_TIMEOUT` is calculated using the following formula:

$$\begin{aligned} ACK\_TIMEOUT_{Lb_{(i)}} &= ACK\_TIMEOUT_{Ub_{(i-1)}} \\ ACK\_TIMEOUT_{Ub_{(i)}} &= 2 \times ACK\_TIMEOUT_{Ub_{(i-1)}} \end{aligned} \tag{5.1}$$

This means that the lower and upper bounds on the `ACK_TIMEOUT` are re-calculated

in every i^{th} contention stage using the previous value of ACK_TIMEOUT. This ensures that the value of the ACK_TIMEOUT is never the same as the previous value. As for the ASRTX_TIMEOUT, it is a randomly generated value used between every retransmission and redundant retransmission. The values Nessa *et al.* used in their paper can be seen in Table 5.1.

Table 5.1: Nessa *et al.'s* transmission parameters.

Parameter	Settings
Tx-Power	18
Frequency and Bandwidth	915 and 125 BW
SF	[7, 8, 9]
ACK_TIMEOUT $_{\text{min}}$	4
ACK_TIMEOUT $_{\text{max}}$	64
ACK_TIMEOUT $Ub_{(0)}$	ACK_TIMEOUT $_{\text{min}}$

The third step was then to start implementing the method into the physical testbed. It became apparent quite early on, however, that changes to the authors' method needed to be made to fit the physical testbed used in this thesis. One change was, since the authors' method is evaluated on a simulator with multiple end-devices, their used parameter values could not be used. This is because they were too high for only two end-devices. Furthermore, due to the limitations of the LoRa RFM9x module, standard values for the SF had to be used, which meant leaving Tx-power standard as well. Therefore, modified values had to be used, which can be seen in Table 5.2.

Another change was related to the SF. The authors' method increases the SF by one for each contention stage. This was problematic for the physical testbed since, as mentioned in Section 2.5, the LoRa module is only one channel. Therefore, this functionality was not implemented.

Table 5.2: The transmission parameters used in this thesis.

Parameter	Settings
Tx-Power	13
Frequency and Bandwidth	433 and 125 BW
SF	7
ACK_TIMEOUT $_{\text{min}}$	0.2
ACK_TIMEOUT $_{\text{max}}$	6
ACK_TIMEOUT $Ub_{(0)}$	ACK_TIMEOUT $_{\text{min}}$

When these steps were done, the final phase was to debug the solution. All the parameters in Table 5.2 are the final ones used. However, these were determined through extensive testing on the physical testbed. The values had to work with only two end-devices, but they needed to work when simulated as multiple end-devices

with a high duty cycle. Nessa *et al.'s* values would not stress the reliability method on the physical testbed at all, therefore not creating any congestion on the network link. As for the timing diagram for the physical testbed, it remained the same as Nessa *et al.'s* since the parameters for the send and receive windows were still used, albeit smaller.

5.4 Differences between the two reliability methods

The two reliability methods are both similar and different from each other. The alternative method is designed around the default method, altering the timing diagram for the uplink and receive windows mentioned in the previous section. These two different windows can be seen in Figure 5.4 with a flowchart in Figure 5.5. As can be seen in Table 5.3, the alternative method adds features that are built on top of the default ones. For example, the variable `ACK-Timeout` and the `Redundant Retransmission` are not present in the default method. The `nb-Trans` exists in the default method, which determines how many transmissions should be done for an acknowledged and non-acknowledged packet. Since ADR sets the `nb-Trans`, this feature does not exist in the alternative method.

Table 5.3: The key features for the two methods.

Feature	Default method	Alternative method
ACK-Timeout	✓	✓
Variable ACK-Timeout	×	✓
Acknowledged Retransmission	✓	✓
Redundant Retransmission	×	✓
`nb-Trans`	✓	×
Max-delay	✓	✓
ADR (if available)	✓	×

To conclude, the two methods share most of the features since the alternative method is based on the default one. The main difference is how the methods handle the retransmission of non-acknowledged messages.

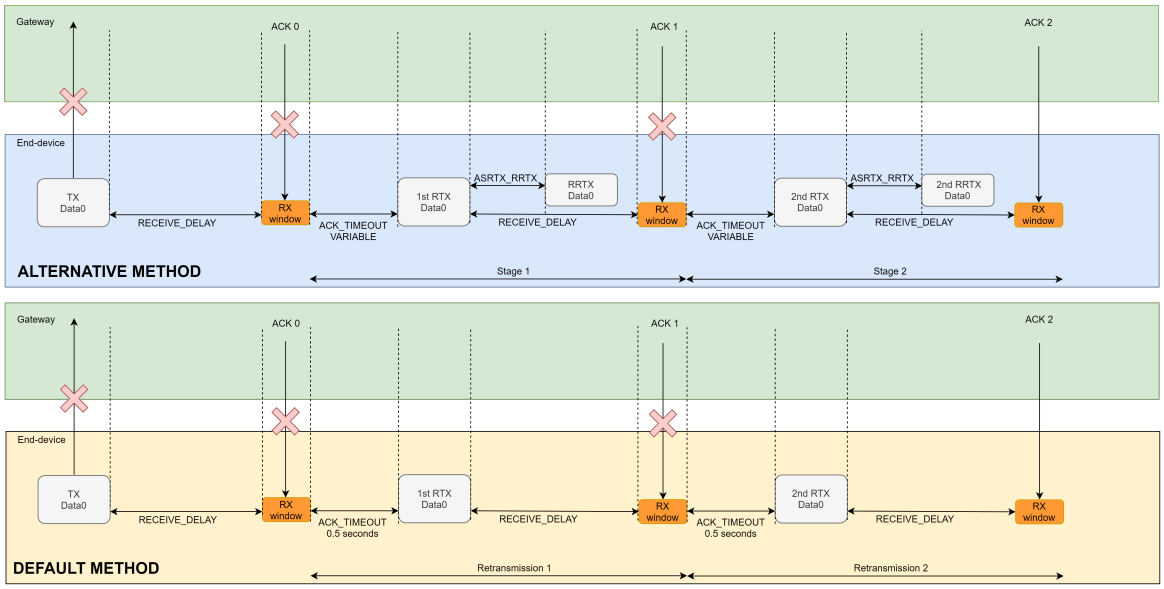


Figure 5.4: The two timing diagrams. One for the default method and one for the alternative method, which is inspired by Nessa *et al.'s* timing diagram [6]. Important to note here is that the `ACK_TIMEOUT` is of different lengths in reality. In the alternative method, the `ACK_TIMEOUT` varies, whereas, in the default, it is static.

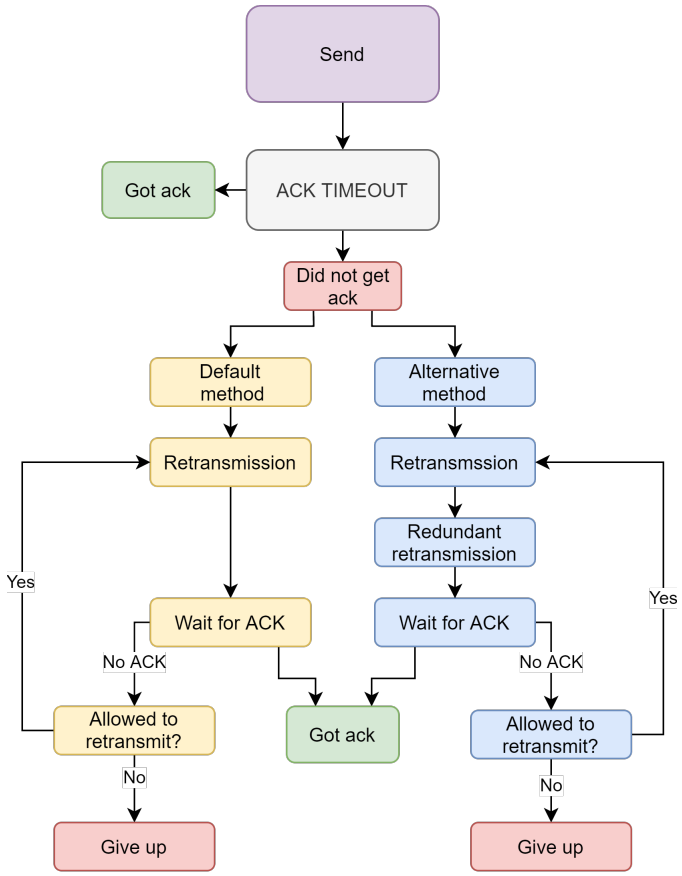


Figure 5.5: Simple flowchart showing what happens in the two different reliability methods. Blue is the alternative method, and yellow is the default method.

5.5 Implementation challenges

Throughout the development of the physical testbed, there were a few challenges faced that had to be resolved. The following section will go through the encountered challenges by presenting the problem and then the solution used to tackle the problem. It will begin with the hardware challenges that were faced. It will then move on to the problems faced with Azure. Finally, this section is concluded with implementation difficulties.

5.5.1 Hardware related challenges

The LoRa module RFM9x is, as previously mentioned, only one channel. Since this means that a gateway can only handle one end-device at any given time, this was not ideal for the physical testbed. This was, however, not a significant problem since it could be solved by not implementing ADR and not using frequency hopping.

However, the initial plan related to the data collection was to change the value range for Tx-power and SF and send all 1000 data messages as permutations of these settings. The problem with this was that the SF did not want to cooperate with the module. Although the settings could be changed and the module reported that it had applied the SF, it was problematic to send and receive even the most minor data between the devices. The SF was changed to the same values throughout the end-devices, but it was still troublesome to get anything consistent when sending and receiving.

Additionally, as mentioned in Section 2.3.4, the SF affects the data rate. Lower SF means more data can be sent, but at the cost of a lower time for the receiver to sample the signal and shorter range. Furthermore, the data rates, as shown in Table 2.1, are still valid up to SF 11 for the maximum payload the physical testbed sends, which is 52 bytes or 416 bits. This meant that ultimately, the SF had to remain at its standard value of seven, which in turn meant that following the reliability method's way of changing the SF for retransmission could not be done to a fault.

5.5.2 Azure related challenges

Azure was decided early on to be used as a form of easily visualize the telemetry data. It took time to get it up and running, but when it was finished, it looked very promising. However, as already touched upon in Section 4.2.5, Azure IoT Central has no way of deciding the time span of when to show the posted telemetry data. This lack of feature was critical in our case since this meant that there was no way of displaying each of the test runs separately. Azure IoT Central can only show a specific time window of data, namely the last 100 values, the last day, the last hour, or the last 30 minutes. Thus visualizing older stored data was not possible. This led to the Azure IoT Central implementation becoming a POC for a cloud-based LoRaWAN. The solution was to instead store all the telemetry data locally through log files and use Matplotlib to display the data.

5.5.3 Implementation difficulties

Although there were no major difficulties when programming the solution, the time investment was higher than initially anticipated. This was because it was challenging to see if the implementations worked correctly and because reading literature and specifications to implement the reliability methods took time.

Another aspect that was challenging with the implementation was the things that were thought to be done with but needed to be changed. For instance, a boilerplate was created for a LoRa-device. This boilerplate would then be used by both the gateway and the end-devices. However, because of the boilerplate structure, if the gateway needed a change in the boilerplate structure, then the end-devices would be affected as well. This happened a few times but in smaller and different formats.

6
Results

This chapter will show the results obtained from all the tests done on the alternative and default reliability methods. It will begin by presenting all the results as an overview for both testbeds and reliability methods. It will then move on to the results for each data field on each testbed, beginning with showing how the latency is affected between the two reliability methods. It will then further look at how the Packet Delivery Rate (PDR) is affected and the Packet Acknowledgment Rate (PAR). It will then conclude with results on how the RSSI and SNR were affected, along with a comparison of Nessa *et al.'s* results and this thesis's results.

6.1 Data collection on two different testbeds

The data collection was done on two physical testbeds. This was done by running ten tests with the settings in Table 6.1. Five of the tests were run with the default reliability method, whereas the other five were run with the alternative method. After the data had been collected, the average value over all the tests was calculated for each end-device. The average was calculated for latency, PDR, PAR, RSSI, and SNR. These averages can be seen in Listing 6.1 for both testbeds. Every data field in these listings will be covered in its own sections in this chapter. All of the results will then be discussed in detail in Chapter 7.

Table 6.1: The settings used for the five tests during the data collection. These settings were used both for the default reliability method and the alternative.

Parameter	**Value**
TX-Power	13
Spreading factor	7
Sending frequency	0.250 s
Time on air	0.160 s
Duty cycle	98.43%
Size of message (bytes)	52
Number of messages	1000

```
######################### Testbed 1 ##########################
Default                            |Alternative
Results for host:  jo-raspb4       |Results for host:  jo-raspb4
     AVG latency: 1.063162         |     AVG latency: 0.305565
     AVG snr: 5.946608             |     AVG snr: 6.085039
     AVG rssi: -43.295624          |     AVG rssi: -43.297718
     PDR: 0.914000                 |     PDR: 0.937800
     PAR: 0.886600                 |     PAR: 0.842400
     AVG Runtime: 1841.2 s         |     AVG Runtime: 846.6 s
Results for host:  jo-raspb2       |Results for host:  jo-raspb2
     AVG latency: 0.644878         |     AVG latency: 0.256612
     AVG snr: 4.811879             |     AVG snr: 5.748439
     AVG rssi: -39.739712          |     AVG rssi: -40.405356
     PDR: 0.986600                 |     PDR: 0.993200
     PAR: 0.896000                 |     PAR: 0.870600
     AVG Runtime: 1794.4 s         |     AVG Runtime: 758.8 s
For these hosts combined           |For these hosts combined
    the averages are:              |    the averages are:
     AVG latency: 0.854020         |     AVG latency: 0.281088
     AVG snr: 5.379244             |     AVG snr: 5.916739
     AVG rssi: -41.517668          |     AVG rssi: -41.851537
     PDR: 0.950300                 |     PDR: 0.965500
     PAR: 0.891300                 |     PAR: 0.856500
     AVG Runtime: 1817 s           |     AVG Runtime: 802 s
######################### Testbed 2 ##########################
Default                            |Alternative
Results for host:  oli-raspold     |Results for host:  oli-raspold
     AVG latency: 0.89286          |     AVG latency: 0.218339
     AVG snr: 6.082547             |     AVG snr: 5.886245
     AVG rssi: -53.300988          |     AVG rssi: -48.713838
     PDR: 0.911000                 |     PDR: 0.958200
     PAR: 0.888400                 |     PAR: 0.879600
     AVG Runtime: 1665.6 s         |     AVG Runtime: 688.6 s
Results for host:  oli-raspuni     |Results for host:  oli-raspuni
     AVG latency: 0.560309         |     AVG latency: 0.209021
     AVG snr: 5.803589             |     AVG snr: 5.928913
     AVG rssi: -40.850171          |     AVG rssi: -48.183746
     PDR: 0.991800                 |     PDR: 0.962200
     PAR: 0.917200                 |     PAR: 0.884400
     AVG Runtime: 1588.4 s         |     AVG Runtime: 684.4 s
For these hosts combined the       |For these hosts combined
    the averages are:              |    the averages are:
     AVG latency: 0.726589         |     AVG latency: 0.213680
     AVG snr: 5.943068             |     AVG snr: 5.907579
     AVG rssi: -47.075580          |     AVG rssi: -48.448792
     PDR: 0.951400                 |     PDR: 0.960200
     PAR: 0.902800                 |     PAR: 0.882000
     AVG Runtime: 1627 s           |     AVG Runtime: 686 s
```

Listing 6.1: The results from running the default and the alternative reliability methods for testbed 1 and testbed 2. Each method was run five times with 1000 transmissions.

6.2 Latency

This section will present the results on how the latency is affected between the two reliability methods. It is worth reminding that lower latency is preferable. The results will be presented from both of the testbeds, respectively. One end-device in each testbed will be presented with its own graphs, along with a scatter plot containing both end-devices. The plots for the two other end-devices can be found in the Appendix, with testbed 1 in Appendix A and testbed 2 in Appendix B.

6.2.1 Testbed 1

Testbed 1 was located in two separate rooms with a thick concrete wall between the end-devices and the gateway. Figure 6.1 and Figure 6.2 contains a violin plot and a box plot of the latency results for one end-device. The figures are results from the alternative and default methods, respectively. When comparing these two figures, it can be seen in the violin plot for the alternative method that its biggest density is located lower on the Y-axis than for the default. Furthermore, the violin plot for the alternative has lower outliers. This means that the alternative method both has overall lower latency and lower individual latency points. It is worth noting that looking at the box plots is that the alternative method is not as consistent with the latency as the default one is. This behavior translates to the other end-device `raspb4` in the testbed as well, which can be seen in Figure A.1 and A.2 located in Appendix A.

Looking at the scatter plot in Figure 6.3, it can be seen that the average latency remains roughly the same for each end-device in each method. The figure also shows that the alternative method has a lower average latency compared to the default one. Another thing of interest is that `raspb4` has a higher latency than `raspb2` in both the default method and the alternative, albeit a substantially larger difference in the default method.

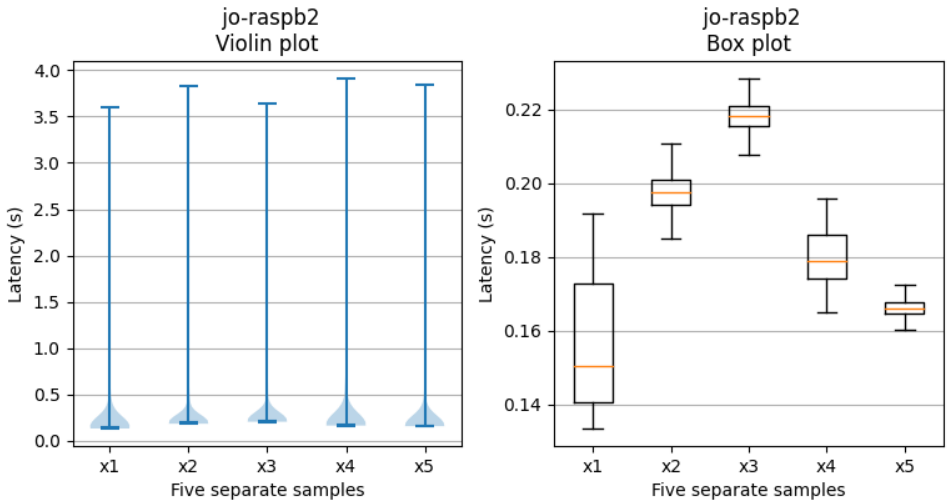


Figure 6.1: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

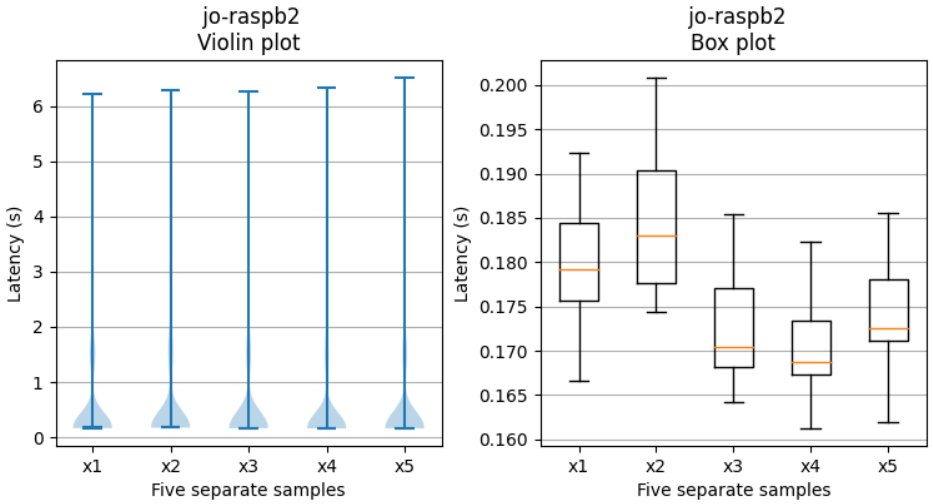


Figure 6.2: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

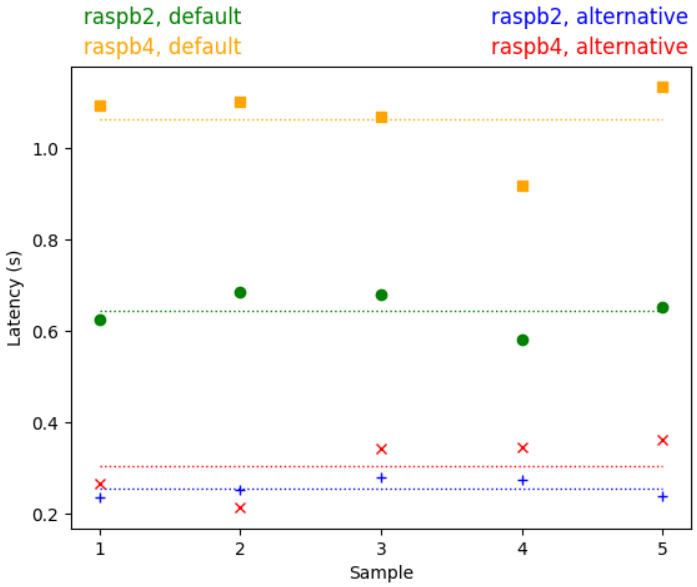


Figure 6.3: A scatter plot of the **average latency** value for each sample. Here, both end-devices are shown for both reliability methods. The dotted line is the average for all the samples combined.

6.2.2 Testbed 2

Testbed 2 was the one located in the open floor environment. The Figures 6.4 and 6.5 represent the latency for one end-device. These figures show latency for the end-device in each sample for both the alternative and the default method. Starting with the violin plots, it can be seen that the default method has a higher max latency than the alternative. However, the box plots show that the alternative method has a smaller spread with its latency than the default method.

When it comes to the other end-device in the testbed, `raspuni`, this spread is not reproduced. This can be seen in Figure B.1 and B.2 located in Appendix B. In these figures, there is a smaller difference between the two box plots. The violin plots here, however, still show that the default method has a higher max latency. It also shows a somewhat bigger concentration of values closer to 1 in the default method than in the alternative method.

The scatter plot in Figure 6.6 shows that the average latency for each sample is higher in the default method than it is in the alternative. Furthermore, it can be seen that `raspold` has a higher latency in almost all samples for both the alternative and the default method.

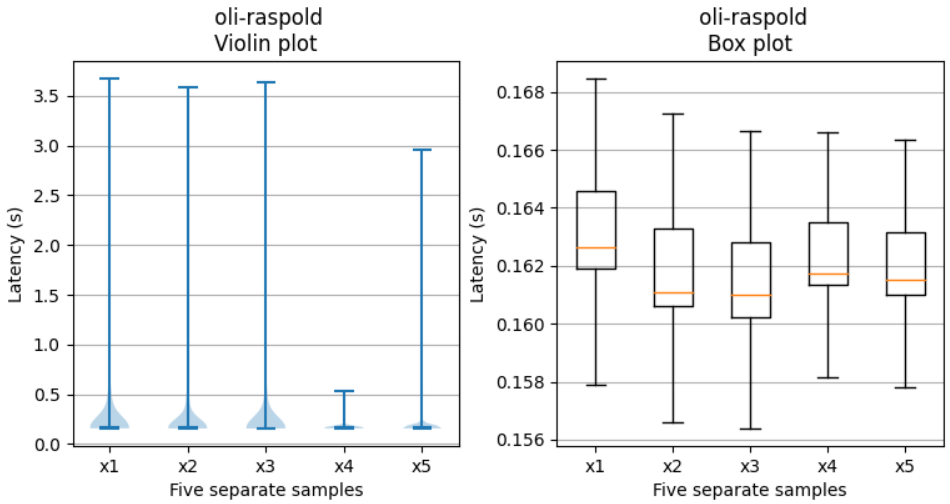


Figure 6.4: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

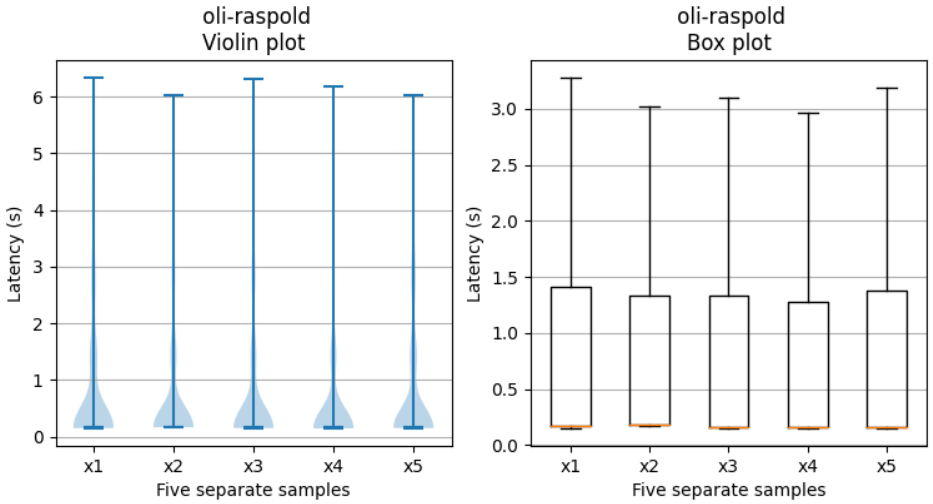


Figure 6.5: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

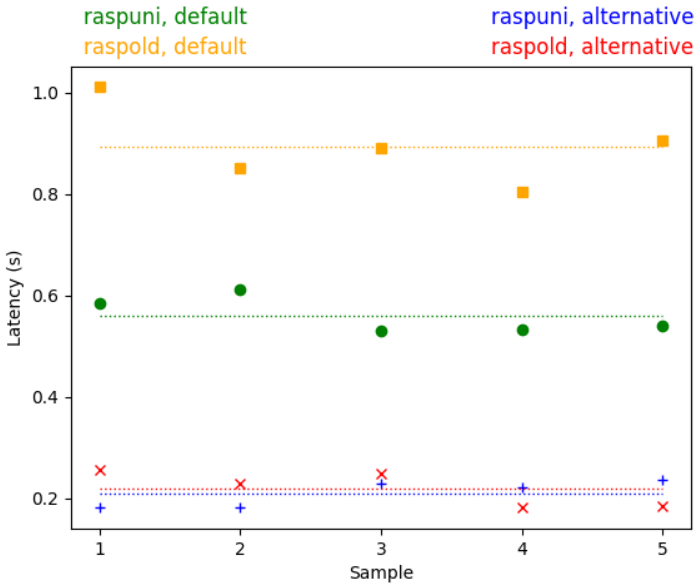


Figure 6.6: A scatter plot of the **average latency** value for each sample. Here, both end-devices are shown for both reliability methods. The dotted line is the average for all the samples combined.

6.3 Packet Delivery Rate

This section will present the results on how the PDR is affected by the two reliability methods. The results will be presented from both of the testbeds, respectively. It is worth reminding that a PDR that goes towards 100 % is preferable since that means a method can successfully deliver more packets.

6.3.1 Testbed 1

In Figure 6.7, the PDR can be seen for both end-devices using both reliability methods. The first thing to notice is that the `raspb2` device is better in all samples for PDR. However, looking at the scatter plot for both end-devices, both have a higher PDR with the alternative method. This can also be seen in Listing 6.1.

6.3.2 Testbed 2

In Figure 6.8, the PDR can be seen for both end-devices using both reliability methods. As can be seen, the default method has the highest average PDR for `raspuni`. The default method also has the most consistent values. However, ignoring the devices and only looking at the alternative compared to the default method, the alternative method has a higher PDR value in all but one sample. For the respective end-device, `raspold` has a better result with the alternative, whereas `raspuni` has a better result with the default method.

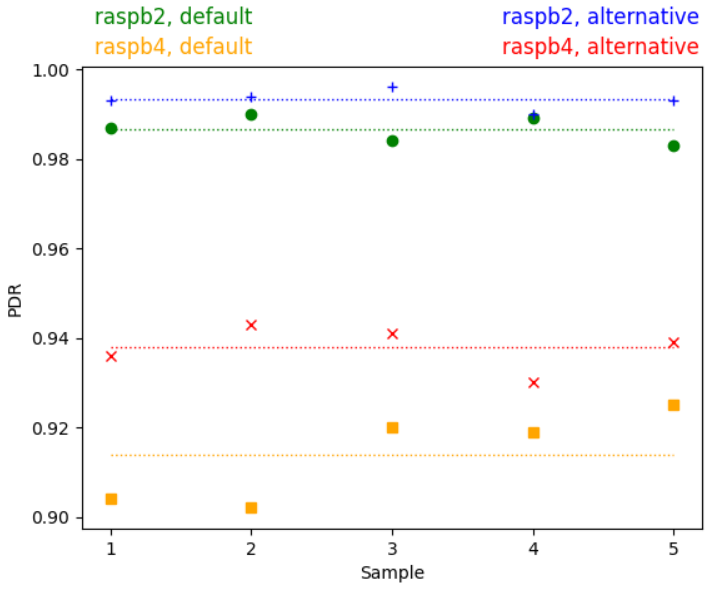


Figure 6.7: A scatter plot of the **average PDR** value for each sample on **testbed 1**. Here, both end-devices are shown for both reliability methods. The dotted line is the average for all the samples combined.

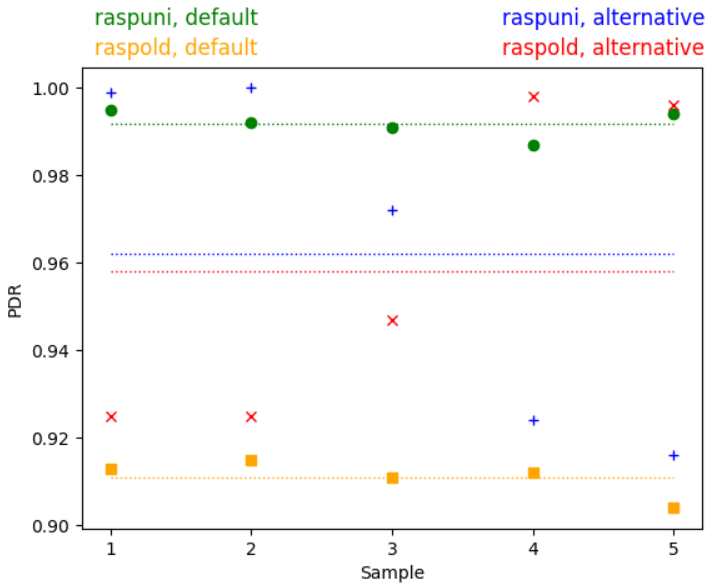


Figure 6.8: A scatter plot of the **average PDR** value for each sample on **testbed 2**. Here, both end-devices are shown for both reliability methods. The dotted line is the average for all the samples combined.

6.4 Packet Acknowledgment Rate

This section will present the results on how the PAR is affected by the two reliability methods. The results will be presented from both of the testbeds, respectively. Worth reminding is that a PAR that goes towards 100 % is preferable since that

means more packets have been acknowledged.

6.4.1 Testbed 1

In Figure 6.9, the PAR can be seen for both end-devices using both reliability methods. As can be seen, the default reliability method has a higher average PAR compared to the alternative method. Worth noting is that it is `raspb2` that once again has the highest values both for the default and alternative method.

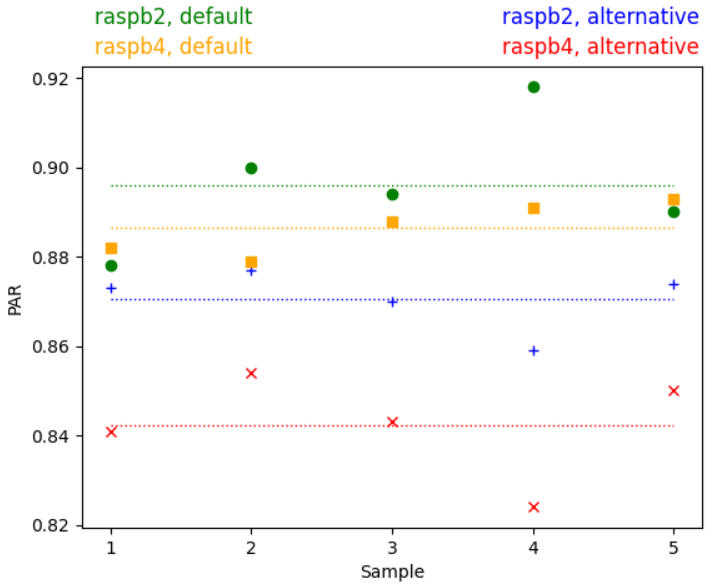


Figure 6.9: A scatter plot of the **average PAR** value for each sample. Here, both end-devices are shown for both reliability methods. The dotted line is the average for all the samples combined.

6.4.2 Testbed 2

In Figure 6.10, the PAR can be seen for both end-devices using both reliability methods. Like for testbed 1, the default reliability method has a higher PAR on this testbed than the alternative method. Comparing Figure 6.10 with Figure 6.8, it can be seen that the PAR follows almost the same pattern as the PDR does for the alternative method.

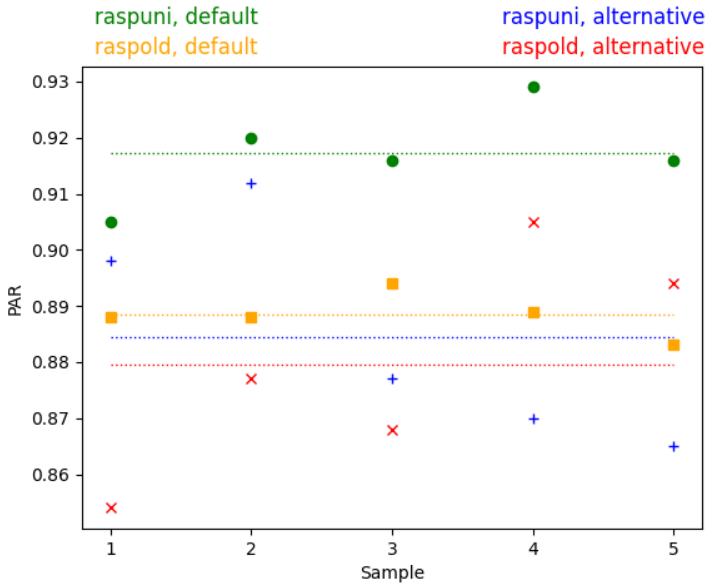


Figure 6.10: A scatter plot of the **average PAR** value for each sample. Here, both end-devices are shown for both reliability methods. The dotted line is the average for all the samples combined.

6.5 Received Signal Strength Indicator and Signal-to-noise ratio

This section will present how the RSSI and SNR are affected by the two reliability methods. The results will be presented from both of the testbeds, respectively. One end-device in each testbed will be presented with its own graphs. The plots for the two other end-devices can be found in the Appendix, with testbed 1 in Appendix A and testbed 2 in Appendix B.

Although the RSSI and SNR are not part of evaluating the reliability methods, it is still important to see how they are affected by the two methods since they reflect the overall signal quality. It is worth reminding that an RSSI value that goes towards zero is preferable since that means that the signal is strong. For SNR, a value above zero is preferable since that means more signal than noise in the signal.

6.5.1 Testbed 1

Starting with RSSI, comparing the violin plots in Figures 6.11 and 6.12, it can be seen that both the default method and the alternative have a high density around -41 dBm for `raspb2`. However, the big difference is the concentration of values around -35 dBm for the default method. Here, it has a higher density of values than the alternative method. This difference can also be confirmed when comparing the two box plots.

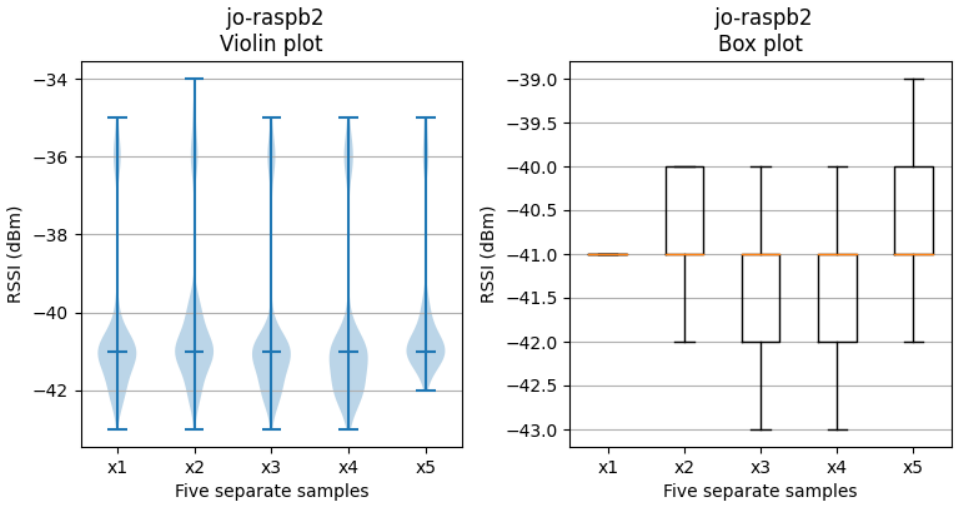


Figure 6.11: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

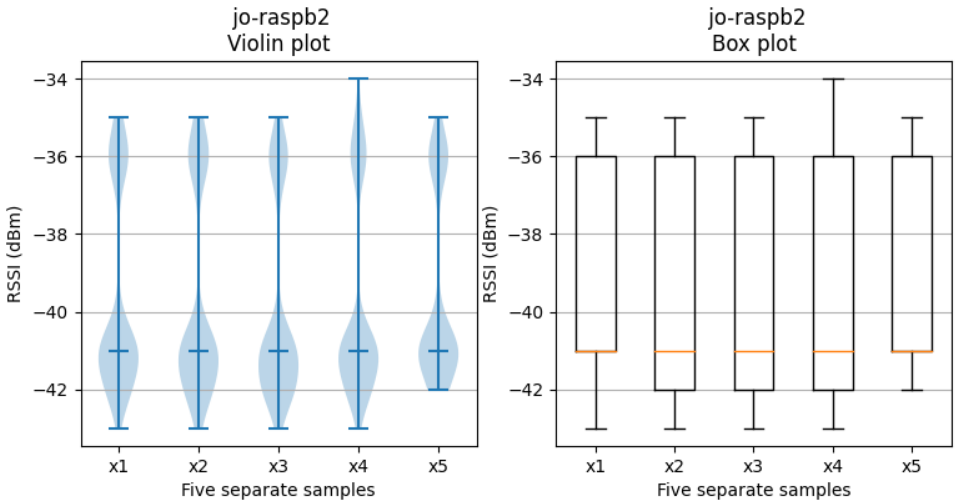


Figure 6.12: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

For the other end-device, `raspb4`, the RSSI is a bit higher where the density of the values for both the default method and alternative method is centered around −43 dBm, and the highest value is −44 dBm. This can be seen in Figures A.3 and A.4. Here both the violin plots and box plots are very similar.

Moving on to SNR, Figures 6.13 and 6.14 show the SNR values for `raspb2` with the alternative and default method respectively. This follows almost the same pattern as RSSI. In the violin plots, the largest density of the values is around 6 dB, but that the default method also has some density of values around 1 dB. This can also be confirmed by looking at the box plots.

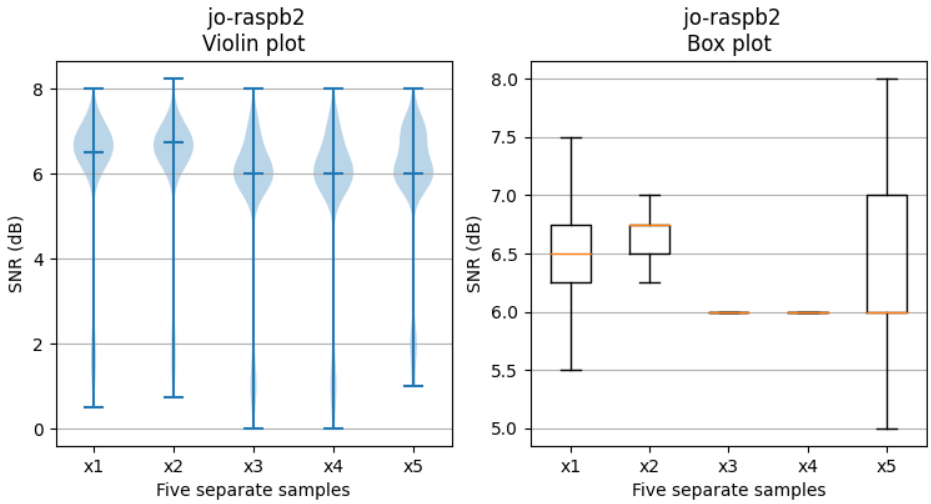


Figure 6.13: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.

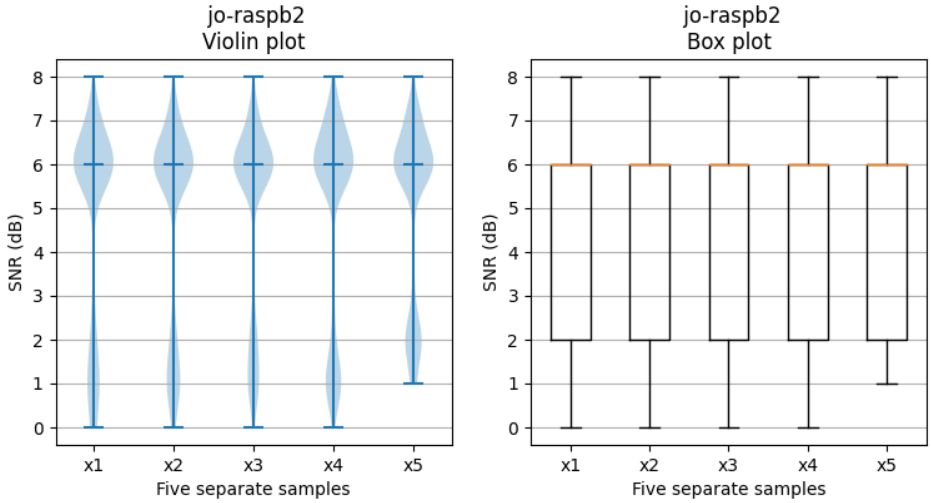


Figure 6.14: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.

For the other end-device, `raspb4`, its SNR values for the alternative method follows the same patterns as for `raspb2` which can be seen in Figure A.5. For the default method however, the SNR values are a bit more consistent compared to `raspb2`. Figure A.6 show that the largest density of the SNR values is at 6 dB.

6.5.2 Testbed 2

Starting with RSSI, Figures 6.15 and 6.16 show the RSSI results for `raspold`. It can be seen through both the violin and box plots that the alternative method is closer to zero than the default method is. The largest density of the values for the alternative method is above −50 dBm, whereas for the default method, they are below −50 dBm. This can be seen in both the violin plots and box plots.

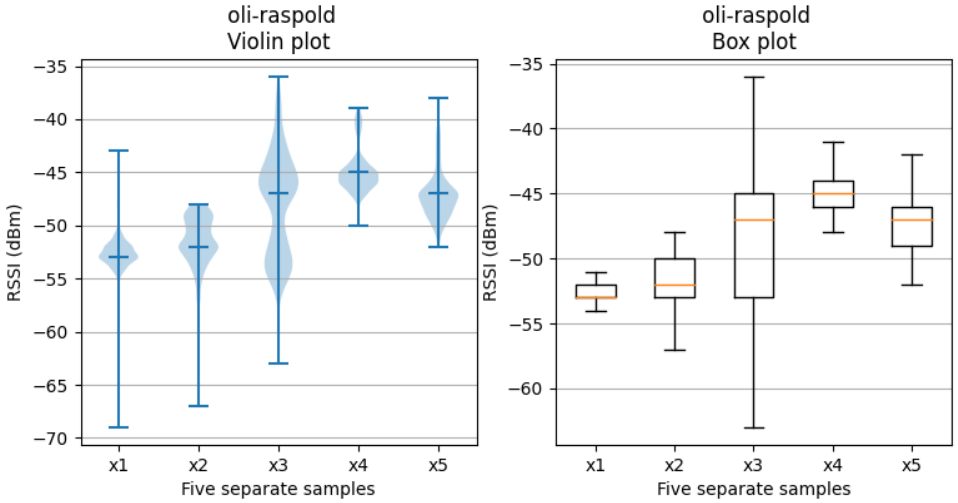


Figure 6.15: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

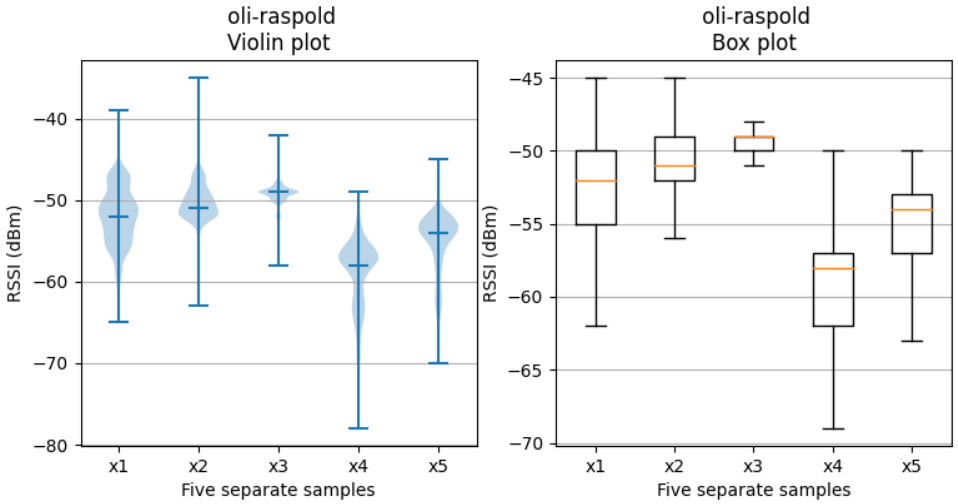


Figure 6.16: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

For the other end-device, `raspuni`, Figures B.3 and B.4 show that the default method generally have lower RSSI values of than the alternative method. For the

most part, the default method is around −43, whereas the alternative has the bulk of its values just below −45.

Moving on to the SNR results, Figures 6.17 and 6.18 show that both of the methods have almost all values around 6 dB. Neither method reaches a higher SNR than the other, but the alternative has the lowest SNR values. This is somewhat replicated for the other end-device `raspuni`, where both of them are centered around 6. This can be seen in Figures B.5 and B.6, where all of the medians are at 6. However, one thing that does differ is that the default method has a higher spread than the alternative method.

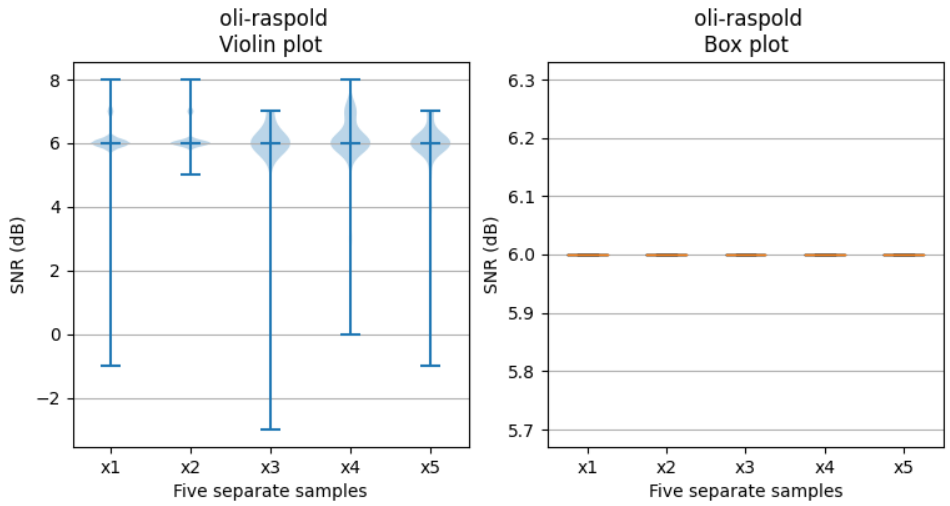


Figure 6.17: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.

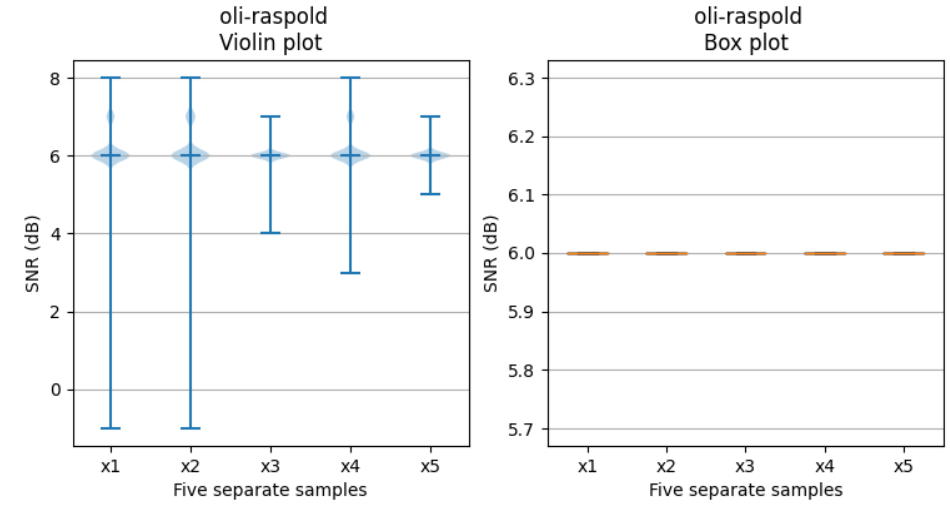


Figure 6.18: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.

6.6 Comparison between Nessa *et al.'s* results and this thesis's results

In Nessa *et al.'s* paper [6], the authors use the Data Extraction Rate (DER) and the average delay per confirmed packet as their two evaluation metrics. Here, DER is another term for PDR. Worth noting again for their test setup is that they simulate upwards of 500 end-devices on a software simulator. On the physical testbeds in this thesis, the end-devices have a duty cycle of 98.43%. Assuming that an end-device should send with 1% duty cycle, this results in the two end-devices simulating a network of 196 end-devices. However, this number does not reflect reality because when the reliability modes kick in, the sending frequency is haltered. With this said, being conservative and estimating that the testbeds simulate 100 end-devices, Figure 6.19 can be used to compare this thesis's results with Nessa *et al.'s* results.

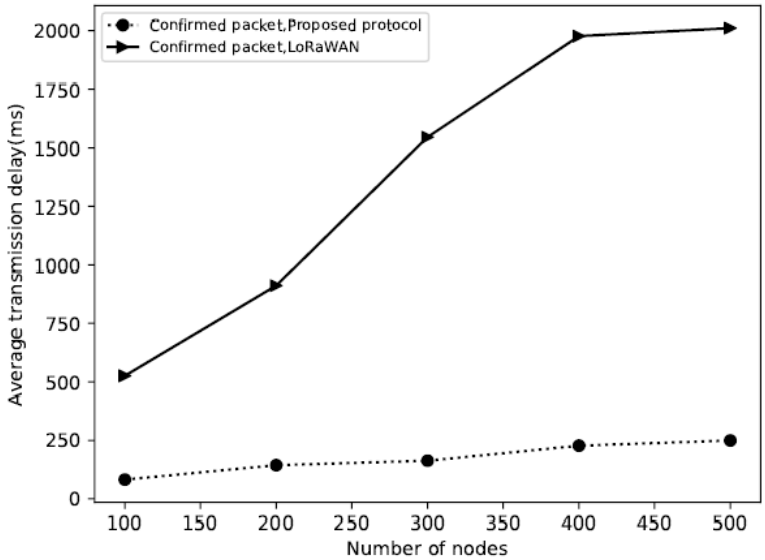


Figure 6.19: Nessa *et al.'s* average transmission delay versus the number of nodes for confirmed packets. Image from their paper [6].

Table 6.2: The latency results for the default and alternative method. It contains both the results of this thesis as well as Nessa *et al.'s* [6].

What result	**This thesis** Testbed 1/Testbed 2	**Nessa *et al.***
Number of nodes	100	100
Packet length (bytes)	52	10
Latency alternative	0.280s/0.214s	$\approx$ 0.180 s
Latency default	0.850s/0.727s	0.500 s

As can be seen in Table 6.2, Nessa *et al.'s* simulation resulted in lower latency values for both the default and the alternative reliability methods. However, they use a packet length of 10 bytes compared to this thesis, which impacts the latency due to the packets spending less Time-on-Air.

In Figure 6.20, Nessa *et al.'s* PDR results can be seen. This figure can be used as a rough comparison between this thesis and Nessa *et al.'s* results. However, they specify different kinds of traffic, making the comparison not fully doable. Nevertheless, choosing their most prioritized data, i.e., the emergency data, the PDR differs by around 4%. This comparison can be seen in Table 6.3.

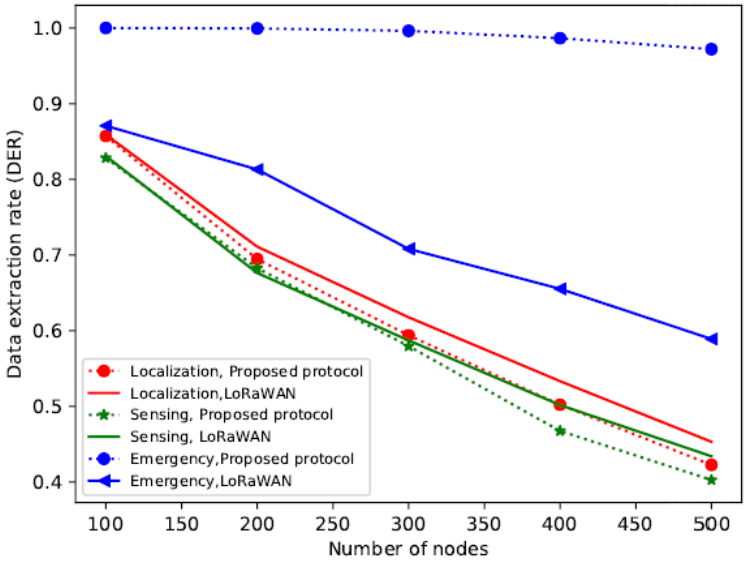


Figure 6.20: Nessa *et al.'s* DER (PDR) versus the number of nodes under different kinds of traffic. Image from their paper [6].

Table 6.3: The PDR results for the default and alternative method. It contains both the results of this thesis as well as Nessa *et al.'s* [6].

What result	**This thesis** Testbed 1/Testbed 2	**Nessa et al**
Number of nodes	100	100
Packet length (bytes)	52	10
PDR Alternative	96.6%/96.0%	100%
PDR default	95.1%/95.2%	87%

7
Discussion

This chapter discusses many aspects of this thesis. It will begin with an in-depth analysis of all of the results presented in Chapter 6. The second section will discuss the aspect of using LoRaWAN with IIoT devices. The third section contains comments on the custom-built LoRaWAN infrastructure. The chapter then concludes with a discussion on potential future work that can be performed on this thesis.

7.1 Discussion of the results

This section will analyze and discuss the obtained data results from Chapter 6. It will give more insight into the presented results and discuss what stands out from the results. It will also discuss what could have affected the results.

Starting with PDR, both testbeds had improved results with the Alternative Method (AM) compared to the Default Method (DM). This can be seen in Table 7.1. One thing that stands out with the averages is that the AM has a better average in all cases except for `raspuni`.

Table 7.1: All average PDR values for all end-devices for both methods and both testbeds.

End-device/testbed	Default method	Alternative method
raspb2	98, 7%	99, 3%
raspb4	91, 4%	93, 8%
Testbed 1	95, 1%	96, 6%
raspold	91, 1%	95, 8%
raspuni	99, 2%	96, 2%
Testbed 2	95, 2%	96, 0%
Both testbeds	95, 2%	96, 3%

This is thought to be the case because the end-devices compete for the gateways' attention. On testbed 1, `raspb2` is the device that gets the best PDR for both the AM and DM in all samples. For testbed 2 it is instead `raspuni` for the DM. It is, however, different from the AM on testbed 2. In Figure 6.8 for the AM, it looks like `raspuni` has established a good communication connection with the gateway

for samples [1, 2]. But in samples [4, 5], it is instead `raspold` with the good communication connection. Finally, in sample [3], it looks like both end-devices are sharing the connection. Thus, even if `raspuni` performs worse in the AM compared to the DM in some samples, it does so by giving increased PDR to `raspold`.

Concluding the PDR results, the AM generally performs better than the DM. From our understanding, this is probably because of the redundant retransmission with a pseudo-random value as a delay. Another factor could be the `ACK_TIMEOUT` based on the ToA value in the AM compared to the static value in DM. The `ACK_TIMEOUT` is of interest because two or more end-devices could start their retransmission at the same time. That would mean that they would be somewhat synchronized and that their retransmissions could cancel out each other. With a static `ACK_TIMEOUT`, they could stay synchronized like this until they reach `ACK_TIMEOUT_MAX` and give up on the transmission, or if one of them successfully retransmits. Because of this, a non-deterministic `ACK_TIMEOUT` could stop the synchronization, whereas, with the DM with static `ACK_TIMEOUT`, synchronized end-devices could take longer to become asynchronous.

An indicator that the redundant retransmission is one improvement in PDR for the AM is the PAR. The PAR metric indicates how many packets are acknowledged and how much data traffic a reliability method generates. The results for PAR in Section 6.4 showed that the PAR was lower for AM than it was for DM. Thus the AM has a higher PDR but lower PAR. This is one of the costs of using the AM compared to the DM. The results can be seen in Table 7.2.

Table 7.2: All average PAR values for all end-devices for both methods on both testbeds.

End-device/testbed	Default method	Alternative method
raspb4	88, 7%	84, 2%
raspb2	89, 6%	87, 1%
Testbed 1	89, 1%	85, 7%
raspold	88, 8%	88, 0%
raspuni	91, 7%	88, 4%
Testbed 2	90, 3%	88, 2%
Both testbeds	89, 7%	87, 0%

Since this value should be high, it can be concluded that the AM generates more data traffic than the DM on both testbeds. This applies to all end-devices both individually and their averages on both testbeds. Worth noting is that if both methods had the same PAR, the AM would still generate more data traffic. This has to do with the fact that the AM utilizes redundant retransmissions.

Moving on from the PAR results and looking at Table 7.3, it can be concluded that the AM has overall a better average latency. This reduction in latency is believed

to be due to two factors in the AM. The first one is redundant retransmission, as it occurs almost immediately after the first retransmission. Thus if the first one is not received, then the redundant retransmission will be received instead. The second factor is the `ACK_TIMEOUT`. As shown in Section 5.3, the `ACK_TIMEOUT` in the AM starts as a small value and then grows exponentially. This, compared to the static `ACK_TIMEOUT` in the DM, leads to the first retransmission stage being initiated quicker in the AM compared to the DM.

Table 7.3: Average latency in milliseconds for all end-devices for both methods and both testbeds.

End-device/testbed	Default method	Alternative method
raspb4	1063.2	305.6
raspb2	644.9	256.6
Testbed 1	854.1	281.1
raspold	892.9	218.3
raspuni	560.3	209.0
Testbed 2	726.6	213.7
Both testbeds	790.4	247.4

Following the latency results, the average RSSI values are displayed in Table 7.4. The average RSSI between DM and AM is generally really close except for the end-devices in testbed 2. Here, `raspuni` have a difference of 7.33 whereas `raspold` have a difference of 4.59. Every end-device in both testbeds was stationary when the tests were performed, along with the same SF and Tx-power settings. Therefore, there should not be any major differences between the methods. The only theory regarding these two end-devices is that their antenna position changed very slightly during the data collection or temporary signal interference occurred from nearby radio signals.

Table 7.4: Average RSSI in dBm for all end-devices for both methods and testbeds.

End-device/testbed	Default method	Alternative method
raspb4	-43.30	-43.30
raspb2	-39.74	-40.41
Testbed 1	-41.52	-41.85
raspold	-53.30	-48.71
raspuni	-40.85	-48.18
Testbed 2	-47.08	-48.45
Both testbeds	-44.30	-45.15

As for SNR, Table 7.5 shows the average SNR values. SNR for testbed 1 is higher with the AM, whereas on testbed 2, it is the other way around. Overall, the SNR

does not differ that much between devices and methods except for `raspb4` which seems to have had a better signal compared to the other devices. Although the `raspb2` improved with almost 1 dB from the DM to the AM, it is marginal. However, these small variations could be caused by the same reasons as for RSSI since both these metrics have to do with the signal quality.

Table 7.5: Average SNR in dB for all end-devices for both methods and testbeds.

End-device/testbed	Default method	Alternative method
raspb4	5.95	6.09
raspb2	4.81	5.75
Testbed 1	5.38	5.92
raspold	6.09	5.89
raspuni	5.80	5.93
Testbed 2	5.94	5.92
Both testbeds	5.66	5.92

Finally, to conclude the discussion of the results, the AM improves the PDR by a small amount, 1.1%. The latency, however, has an improvement of 69% in reduction between the DM and AM. Nonetheless, even though the PDR and the latency had improvements with the AM, these improvements come with a cost. Firstly, more traffic is generated with the AM because it uses redundant retransmissions. On top of this, the AM also has a lower PAR which means that it will make more unnecessary retransmissions than the DM. This results in the AM generating more network traffic and use more power to send than the DM. More power is because more traffic is being sent, and therefore the LoRa module is operational longer time.

As for signal quality, there is not a big enough difference between the two methods in terms of RSSI and SNR. In theory, both SNR and RSSI should remain the same between the two methods since every parameter and placement of the testbeds remained the same. However, since it is radio frequencies, outside interference can affect these results.

The results are summarized as follows:

- **PDR:** The DM has an average of $95,2\%$, whereas the AM has 96.3%. Thus the AM is an improvement of 1.1% increase in PDR.
- **Latency:** The DM has an average of 790.35 ms, whereas the AM has 247.4 ms. Thus the AM is an improvement with 69% reduced latency.
- **PAR:** The DM has an average of 89.7%, whereas the AM has 87%. Thus the AM is not an improvement and has a reduced PAR of 3%.
- **RSSI/SNR:** There is no significant difference between the two methods.

7.2 LoRaWAN for IIoT

One of the primary goals of this thesis was to provide numerical results on how reliable LoRaWAN is, to give manufacturers an indication of what LoRaWAN can achieve in terms of reliable communication. There are, however, things to consider when looking at this thesis's results.

The data collection was done on only one spreading factor, which was 7. Regarding how CSS modulation works, increasing the spreading factor will lead to a signal less susceptible to interference and, therefore, higher reliability. This seems very detrimental to the result of this thesis. However, the drawback of increasing the spreading factor is the increase in latency. This can be seen by looking at the data rate of the different spreading factors, shown in Table 2.1. Stepping from SF 7 (5470 bits/s) to SF 8 (3215 bits/s) leads to an increase in latency of 75%. Even if it would be optimal to do the data collection on several different spreading factors, doing it on the lowest spreading factor leads to a lower bound in latency and an indication of a lower bound PDR.

Because of this, IIoT manufacturers can look at this thesis and look at the PDR as a lower bound. This could potentially be improved by increasing the spreading factor. Likewise, the latency can also be looked at as a lower bound, as long as it is taken into account that it will change with distance, packet size, and reliability method.

As for what reliability method to recommend, it depends on network size and density. For denser networks, the recommendation would be the default method with a non-deterministic but increasing `ACK_TIMEOUT`. This is because the network would benefit from the lower latency from the AM and the decreased network traffic of the DM. On the other hand, the AM would be recommended for less dense networks as this increases the PDR and decreases the latency but with the cost of higher power usage and increased network traffic.

7.3 The custom LoRaWAN infrastructure

The custom-built infrastructure for this thesis was the best that could be achieved with the budget available and the project's time frame. It was also a reasonable choice to go with a custom solution to perform the necessary changes to make the hardware work. Although it has succeeded in delivering the required functionality for this thesis, it is still not optimal. As mentioned, the LoRa RFM9x modules have their limits and problems like only being one channel and the problems with getting the SF to work correctly. This is not ideal since the gateway can not handle more than one device simultaneously and therefore does not comply with the LoRaWAN specifications. Furthermore, it makes a potential ADR implementation difficult since ADR adjusts SF, Tx-power, and sub frequencies for individual end-devices.

With this hardware limitation, the adjustment of SF for retransmission could not be made in either reliability method. This may have affected at least the comparison with Nessa *et al.* since their simulator does use variable SF in the contention stages. It is, however, not clear from their results whether or not any end-devices entered contention stages since no data on increased SF is provided in their paper. Nevertheless, between the two methods evaluated on the physical testbed in this thesis, the results are comparable since neither method has dynamic SF.

Since the implemented protocol is a custom one, it does not entirely follow what is described in the LoRa specification. The architecture, as shown in Figure 2.3, is not the architecture for this thesis's setup. Instead, this thesis opted to move most of the functionality into the gateway. This is non-ideal, and a better solution would be to implement it more according to the specification. With that said, the measurements in this thesis occurred between gateway and end-device. The focus was not between the JS or the NS. If one opted to look at the latency between the end-device and AS, then the NS and the JS would need to be implemented.

7.4 Future Work

This section will go through and explain parts of this thesis work that can be extended and improved. It will cover additional parts that were chosen not to be implemented in this thesis but can further benefit the evaluation of reliability methods.

7.4.1 Metrics

This thesis decided that PDR, PAR, and latency would be the primary metrics used for evaluation. PDR was selected to measure the reliability methods' effectiveness and latency for seeing what use cases LoRaWAN enables. PAR was selected to be an indicator of how much traffic was acknowledged and unnecessary generated. However, PAR does not give exact numbers on unnecessary traffic. An improvement to this thesis and its results would be to see how many total transmissions were done. That is, if the application layer sent 1000 data points, how many transmissions were done in total on the MAC layer level.

With this metric, it would be possible to conclude the difference between methods in both actual numbers of transmission and percentages. Thus, this would be a more precise way of seeing how costly a method is in terms of network traffic. Furthermore, one could argue more clearly about the power usage since the exact number of transmissions is available.

7.4.2 Latency with JS and NS

This thesis had to limit itself to making sure that the necessary parts for evaluating reliability were implemented. One addition could be implementing a JS and an NS

to see how the join procedure affects the transmissions. Since this process involves security, it could be interesting to see how encryption affects the latency, the PDR, and PAR. The work done by Mårlind *et al.* [40] could be the first step of integration since the hardware setup is similar to this thesis.

7.4.3 Using a concentrator board for the gateway

As discussed in Section 7.3, the LoRa modules are limited to what can be achieved for a LoRaWAN network in terms of functionality. If instead a concentrator board with multi-channel support was used for the gateway device, it would provide the necessary functionality of supporting multiple end-devices simultaneously. Furthermore, this would mean that it could handle multiple SFs and sub-frequencies for individual end-devices.

As a result, this would provide the ability to support changing the SF for retransmission in both the AM and the DM. It would further then make sense to implement Adaptive Data Rate (ADR) to fully take advantage of the concentrator board since the dynamic transmission settings of ADR would then be a possibility. This would be an interesting addition to see how the two reliability methods fair with these dynamic transmission settings.

7.4.4 Dynamic testing environment

It is not uncommon that IoT and IIoT devices exist in scenarios where they are moving. As a result, these devices are at different distances from the closest gateway. To improve this thesis work and make it more realistic for the industry, conducting mobile tests could be one improvement. This would require first modifying the infrastructure a bit with portable power banks. It would also require automating the programs to automatically start when the Raspberry Pis boot, alternatively making sure that end-devices listen for commands on LoRaWAN to start data collection. Azure could still be used as long as the gateways have an internet connection, either Wi-Fi or cellular. If Azure is not wanted, then The Things Network could be a potential candidate. Finally, if neither Azure nor The Things Network is possible or wanted, local data logging on the gateways is also possible.

With this mobility, it would then be possible to make long-distance tests and see how the reliability methods cope metrics-wise. Additionally, if a concentrator board is used for the gateway, it opens up the ability to change parameters dynamically. Especially the SF, since at a longer distance, the SF needs to be increased for the gateways to have a chance of receiving data from the end-devices.

7.4.5 Evaluate more methods

As mentioned in Section 4.1, there were a few methods that were up for potential evaluation in this thesis. Most of them did not meet the time requirement, however. This made sense for this thesis since the underlying protocol had to be implemented

as well. However, with that now in place, more focus could be put on evaluating other methods. Those mentioned in related work in Chapter 3 are only a few of those researched for this thesis. Reliability methods that are not centered around `ACKNOWLEDGED` traffic are something that could be of interest since the DM and the AM both contain `ACKNOWLEDGED` traffic.

7.4.6 Implement simulators

At the start of this thesis, the initial idea was to evaluate the reliability methods both on the physical testbed and simulators. LoRaSim[1], as used by Nessa *et al.* in their work, or NS-3[2]as used by Reynders *et al.*, were two of the network simulators that was considered to be used. This evaluation had to be skipped for this thesis due to time constraints. Nevertheless, this would have been an interesting complement to the physical testbed since the AM originates from the implementation done on LoRaSim by Nessa *et al.*

[1]https://www.lancaster.ac.uk/scc/sites/lora/lorasim.html
[2]https://github.com/networkedsystems/lora-ns3

8
Conclusion

The purpose of this thesis was to evaluate the reliability in LoRaWAN by investigating the default method in LoRaWAN and see if there exists an alternative method that can be used as an improvement. This thesis has evaluated both the default reliability method and an alternative method found in the literature. This evaluation was performed in terms of the primary metrics latency, PDR, and PAR. The goal was to provide numerical results for manufacturers and companies to determine if LoRaWAN is suitable for IIoT devices.

The evaluated alternative method alters the default method's uplink and downlink windows when sending and receiving data by introducing a redundant retransmission scheme that uses the ToA value of the previous transmission. The primary evaluation came from running tests that sent 1000 messages containing telemetry data. These tests were performed multiple times for each end-device on both testbeds. This made it possible to measure the latency, PDR, PAR, RSSI, and SNR values to see how the two methods perform in terms of these metrics.

The results presented in this thesis show that the alternative method improves in increased PDR and reduced latency. To have a high PDR, while at the same time low latency for every message, makes it possible for LoRaWAN to be used for more mission-critical tasks. The numerical results show that the default method has an average PDR of 95.2% and the alternative method improves this with 1.1%. Furthermore, latency has an average of 790.35 ms in the default method but is reduced by 69% in the alternative method. The PDR and latency improvements of the alternative method come at a cost, however, namely the cost of decreased PAR. The average PAR for the default method is $89,7\%$ but is decreased by 3% with the alternative method. This means that the alternative method can achieve better results in terms of PDR and latency but at the cost of increased network traffic and, therefore, power usage.

This thesis has discussed that both methods come with their respective pros and cons. It has also shown that there can be improvements made to the default method. It is ultimately up to the manufactures to determine whether the provided numerical results for LoRaWAN fit with their specific device requirements and environmental conditions.

Bibliography

[1] Charith Perera, Chi Harold Liu, and Srimal Jayawardena. "The emerging internet of things marketplace from an industrial perspective: A survey". In: *IEEE Transactions on Emerging Topics in Computing* 3.4 (2015), pp. 585–598.

[2] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. "Industrial internet of things: Challenges, opportunities, and directions". In: *IEEE Transactions on Industrial Informatics* 14.11 (2018), pp. 4724–4734.

[3] John Michael Spinelli. "Reliable communication on data links". In: (1988).

[4] Manoj Wadekar. *Handbook of Fiber Optic Data Communication: Chapter 11. InfiniBand, iWARP, and RoCE*. Elsevier Inc. Chapters, 2013.

[5] *LoRaWAN and NB-IoT : competitors or complementary*. [Online]. Accessed 2021-03-15. URL: https://lora-alliance.org/resource_hub/lorawan-and-nb-iot-competitors-or-complementary/.

[6] Ahasanun Nessa, Fatima Hussain, and Xavier Fernando. "Adaptive Latency Reduction in LoRa for Mission Critical Communications in Mines". In: *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2020, pp. 1–7.

[7] *Radio interference*. [Online]. Accessed 2021-05-05. URL: https://www.rsm.govt.nz/business-individuals/interference/radio-interference/.

[8] *What is Azure?* [Online]. Accessed 2021-03-08. URL: https://azure.microsoft.com/sv-se/overview/what-is-azure/.

[9] *Get started guide for Azure developers*. [Online]. Accessed 2021-03-08. URL: https://docs.microsoft.com/sv-se/azure/guides/developer/azure-developer-guide.

[10] *What is Azure IoT Central?* [Online]. Accessed 2021-03-08. URL: https://docs.microsoft.com/sv-se/azure/iot-central/core/overview-iot-central.

[11] *Take a tour of the Azure IoT Central UI*. [Online]. Accessed 2021-03-08. URL: https://docs.microsoft.com/sv-se/azure/iot-central/core/overview-iot-central-tour.

[12] *Build the IoT Central device bridge to connect other IoT clouds to IoT Central*. [Online]. Accessed 2021-03-10. URL: https://docs.microsoft.com/sv-se/azure/iot-central/core/howto-build-iotc-device-bridge.

[13] Eric B. *LoRa*. [Online]. Accessed 2021-03-04. URL: https://lora.readthedocs.io/en/latest/.

[14] *How Spreading Factor affects LoRaWAN device battery life*. [Online]. Accessed 2021-03-08. URL: https://www.thethingsnetwork.org/article/how-spreading-factor-affects-lorawan-device-battery-life.

[15] LoRa Alliance Technical Marketing Workgroup. *A technical overview of LoRa and LoRaWAN ™ What is it?"* [Online]. Accessed December 6 2020. URL: https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf.

[16] *RP2-1.0.2 LoRaWAN® Regional Parameters*. [Online]. Accessed 2021-03-08. URL: https://lora-alliance.org/resource_hub/rp2-102-lorawan-regional-parameters/.

[17] B. Reynders and S. Pollin. "Chirp spread spectrum as a modulation technique for long range communication". In: *2016 Symposium on Communications and Vehicular Technologies (SCVT)*. 2016, pp. 1–5. DOI: 10.1109/SCVT.2016.7797659.

[18] Lou Frenzel. *Fundamentals of Communications Access Technologies: FDMA, TDMA, CDMA, OFDMA, AND SDMA*. [Online]. Accessed 2021-04-06. URL: https://www.electronicdesign.com/technologies/communications/article/21802209/fundamentals-of-co%20mmunications-access-technologies-fdma-tdma-cdma-ofdma-and-sdma.

[19] *LOS vs NLOS | Difference between LOS and NLOS wireless channels*. [Online]. Accessed 2021-04-07. URL: https://www.rfwireless-world.com/Terminology/LOS-vs-NLOS-wireless-channel.html.

[20] *Get started with LoRaWAN*. [Online]. Accessed 2021-03-05. URL: https://www.thethingsnetwork.org/docs/lorawan/index.html.

[21] *LoRaWAN® Specification v1.1*. [Online]. Accessed 2021-03-15. URL: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/.

[22] *LoRaWAN Architecture*. [Online]. Accessed 2021-03-15. URL: https://www.thethingsnetwork.org/docs/lorawan/architecture/.

[23] *Adaptive Data Rate*. [Online]. Accessed 2021-03-15. URL: https://www.thethingsnetwork.org/docs/lorawan/adaptive-data-rate/.

[24] *Difference between Star and Mesh Topology*. [Online]. Accessed 2021-03-15. URL: https://www.geeksforgeeks.org/difference-between-star-and-mesh-topology/.

[25] Ismail Butun, Nuno Pereira, and Mikael Gidlund. "Security Risk Analysis of LoRaWAN and Future Directions". In: *Future Internet* 11.1 (2019). ISSN: 1999-5903. DOI: 10.3390/fi11010003. URL: https://www.mdpi.com/1999-5903/11/1/3.

[26] *Device Classes*. [Online]. Accessed 2021-03-15. URL: https://www.thethingsnetwork.org/docs/lorawan/classes/.

[27] *A complete guide to understanding, monitoring and fixing network packet loss*. [Online]. Accessed 2021-04-07. URL: https://www.ir.com/guides/what-is-network-packet-loss.

[28] *LoRa – Device Activation Call Flow (Join Procedure) using OTAA and ABP*. [Online]. Accessed 2021-04-22. URL: https://www.techplayon.com/lora-device-activation-call-flow-join-procedure-using-otaa-and-abp/.

[29] Mohamed Eldefrawy, Ismail Butun, Nuno Pereira, and Mikael Gidlund. "Formal security analysis of LoRaWAN". In: *Computer Networks* 148 (2019), pp. 328–339.

[30] *LoRaWAN SECURITY: FULL END–TO–END ENCRYPTION*. [Online]. Accessed 2021-04-22. URL: https://lora-alliance.org/wp-content/uploads/2020/11/lorawan_security_whitepaper.pdf.

[31] *Raspberry PI models comparison*. [Online]. Accessed 2021-03-03. URL: https://socialcompare.com/en/comparison/raspberrypi-models-comparison.

[32] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. "Low power wide area networks: An overview". In: *IEEE Communications Surveys & Tutorials* 19.2 (2017), pp. 855–873.

[33] Luiz Oliveira, Joel JPC Rodrigues, Sergei A Kozlov, Ricardo AL Rabêlo, and Victor Hugo C de Albuquerque. "MAC layer protocols for Internet of Things: A survey". In: *Future Internet* 11.1 (2019), p. 16.

[34] Siddhartha Borkotoky, Christian Bettstetter, Udo Schilcher, and Christian Raffelsberger. "Allocation of repetition redundancy in LoRa". In: *European Wireless 2019; 25th European Wireless Conference*. VDE. 2019, pp. 1–6.

[35] Ulysse Coutaud, Martin Heusse, and Bernard Tourancheau. “High reliability in lorawan”. In: *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE. 2020, pp. 1–7.

[36] Brecht Reynders, Qing Wang, Pere Tuset-Peiro, Xavier Vilajosana, and Sofie Pollin. “Improving reliability and scalability of lorawans through lightweight scheduling”. In: *IEEE Internet of Things Journal* 5.3 (2018), pp. 1830–1842.

[37] Paul Marcelis, Nikolaos Kouvelas, Vijay S Rao, and Venkatesha Prasad. “DaRe: Data recovery through application layer coding for LoRaWAN”. In: *IEEE Transactions on Mobile Computing* (2020).

[38] Lady Ada. *Assembly*. [Online]. Accessed 2021-04-30. URL: https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts/assembly.

[39] *Matplotlib: Visualization with Python*. [Online]. Accessed 2021-05-13. URL: https://matplotlib.org/.

[40] Fredrik Mårlind and Ismail Butun. “Activation of LoRaWAN End Devices by Using Public Key Cryptography”. In: *2020 4th Cyber Security in Networking Conference (CSNet)*. IEEE. 2020, pp. 1–8.

A

Testbed 1

This Appendix contains the latency results, RSSI results, and the SNR results from both reliability methods for the second end-device `raspb4` on testbed 1 .

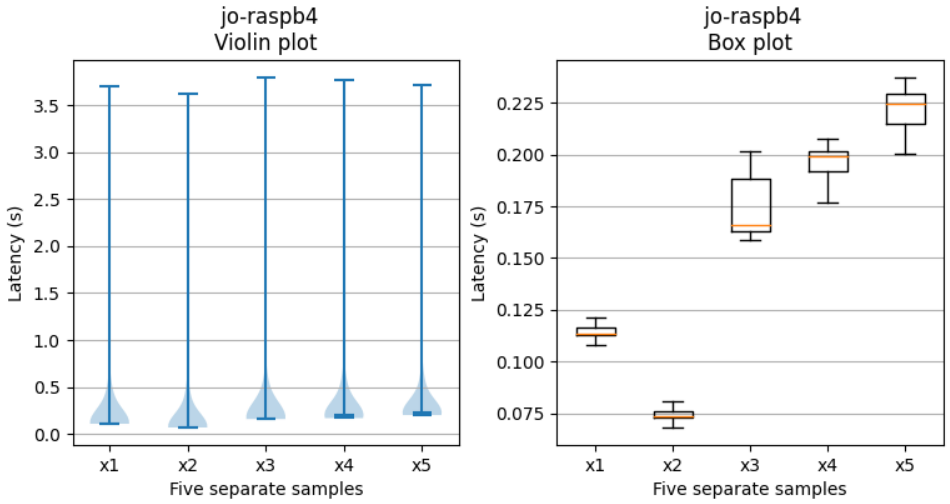


Figure A.1: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

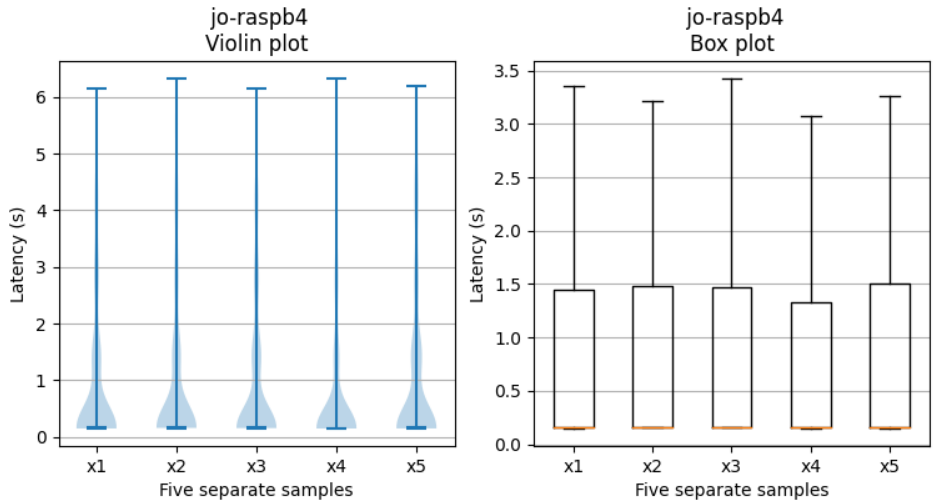


Figure A.2: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

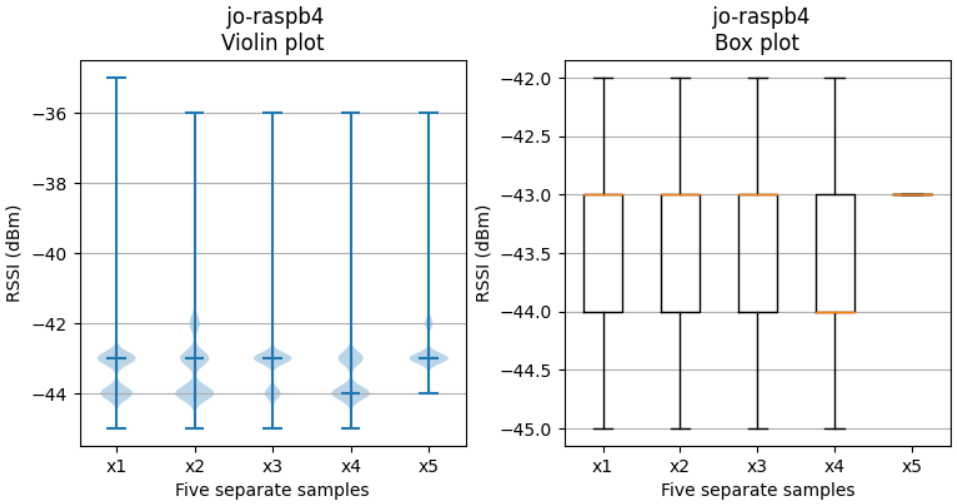


Figure A.3: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

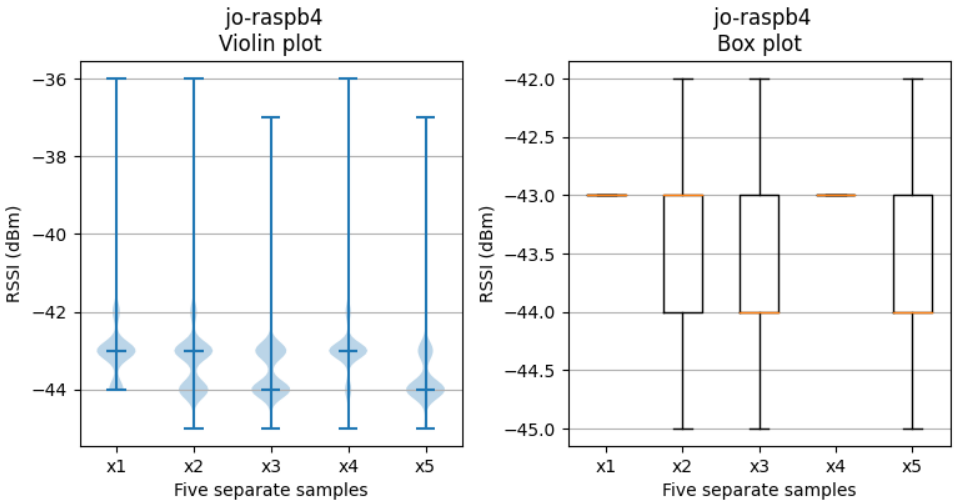


Figure A.4: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

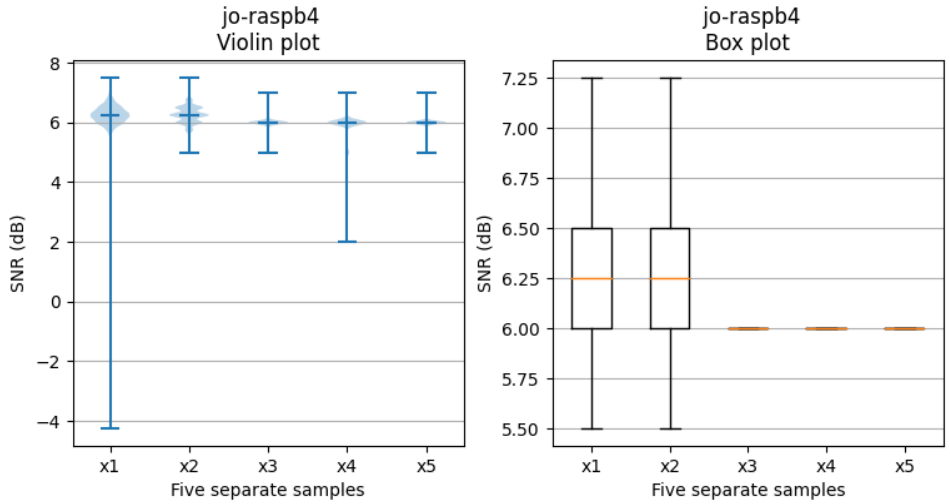


Figure A.5: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.

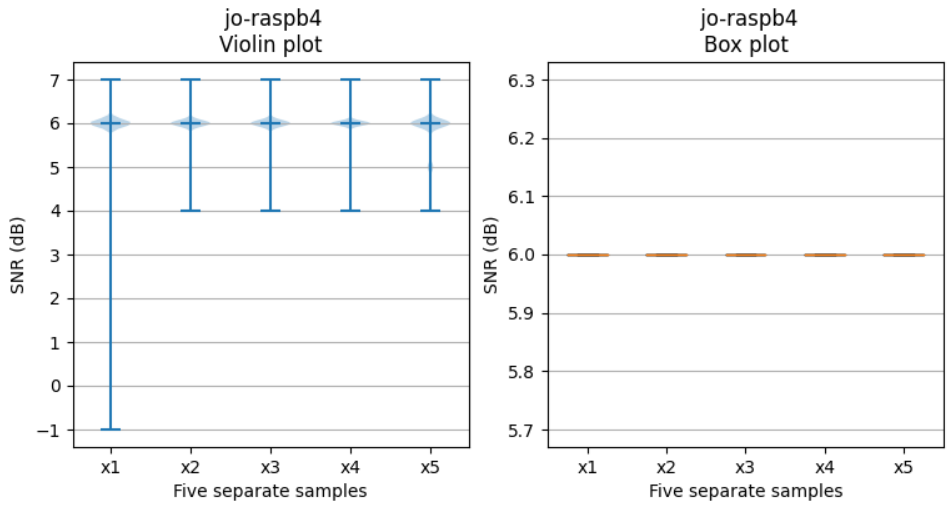


Figure A.6: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.

B

Testbed 2

This Appendix contains the latency results, RSSI results, and the SNR results from both reliability methods for the second end-device `raspuni` on testbed 2.

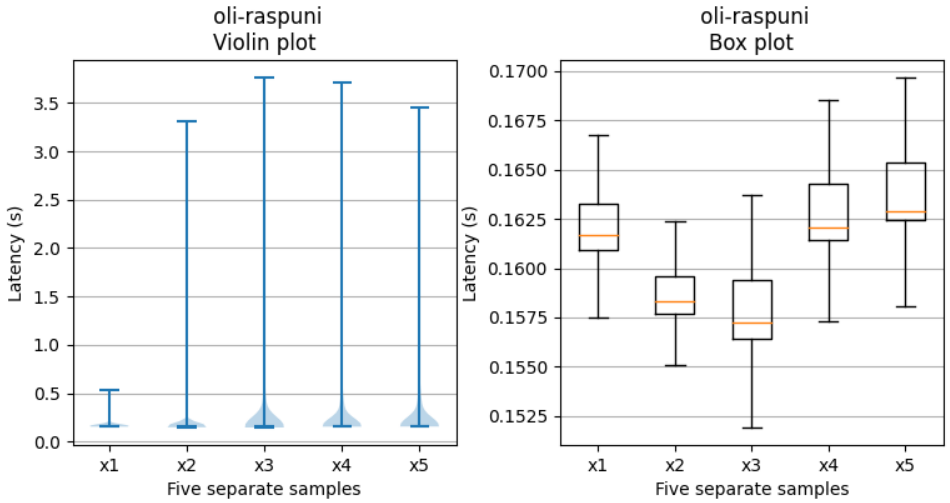


Figure B.1: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

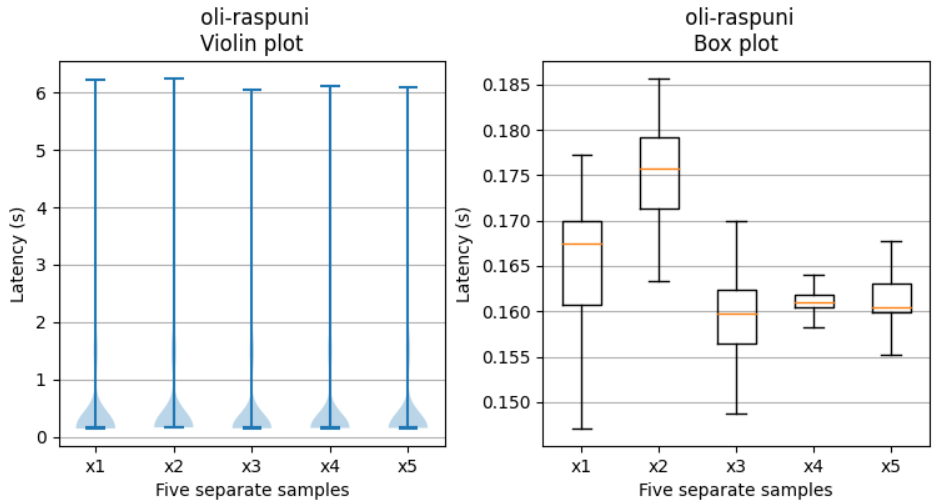


Figure B.2: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **latency** in seconds. Outliers have been removed from the box plot.

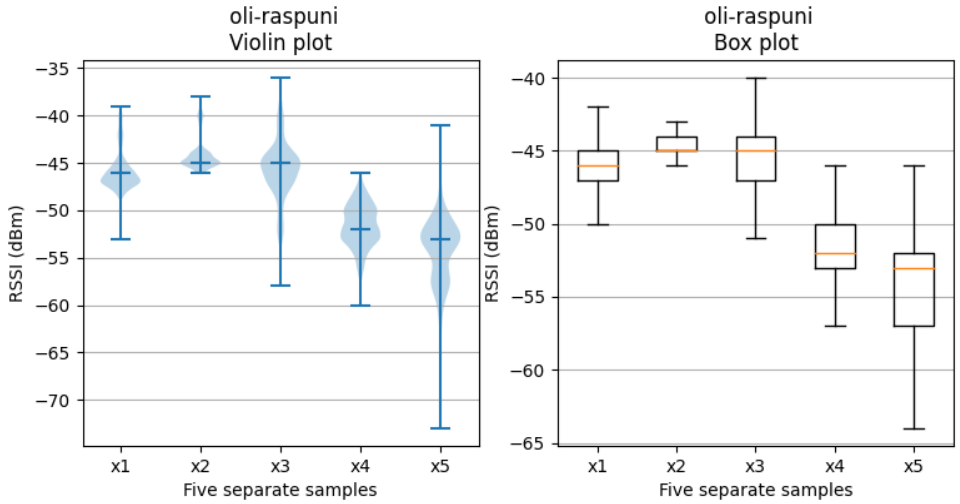


Figure B.3: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

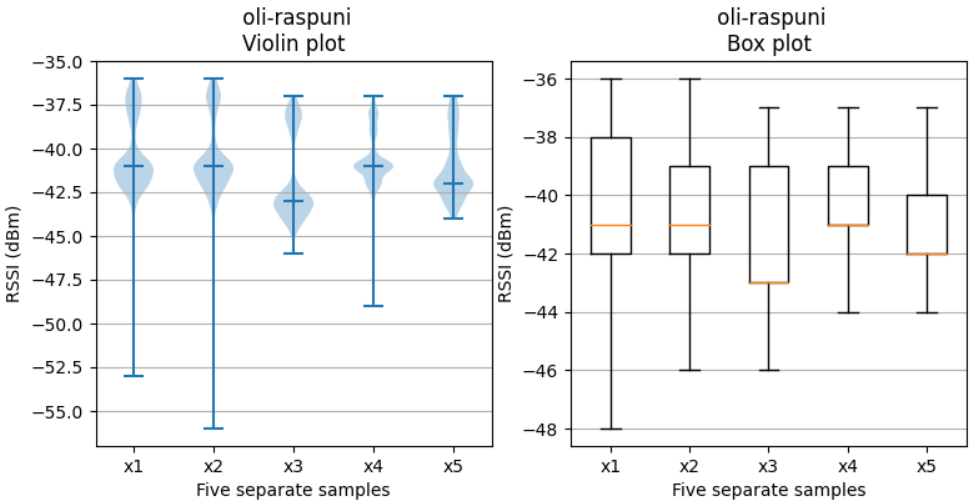


Figure B.4: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **RSSI** in dBm. Outliers have been removed from the box plot.

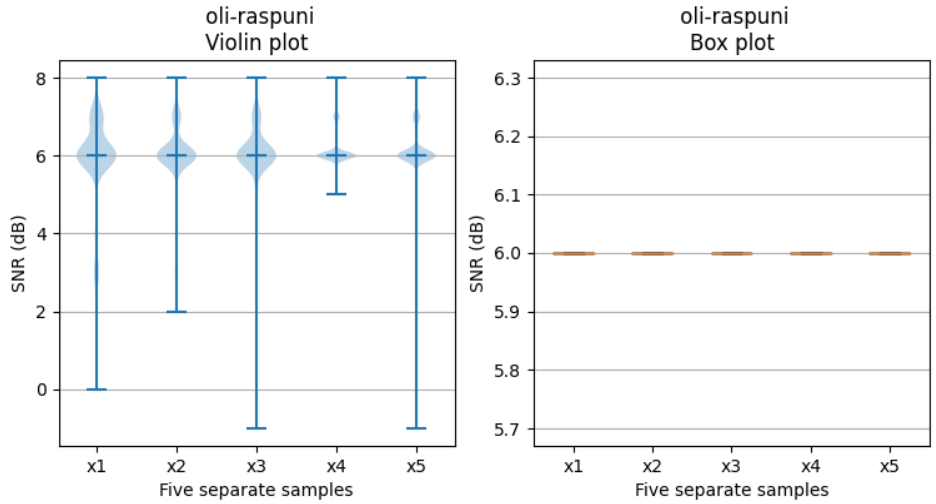


Figure B.5: A violin plot and a box plot of the five samples when the **alternative** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.

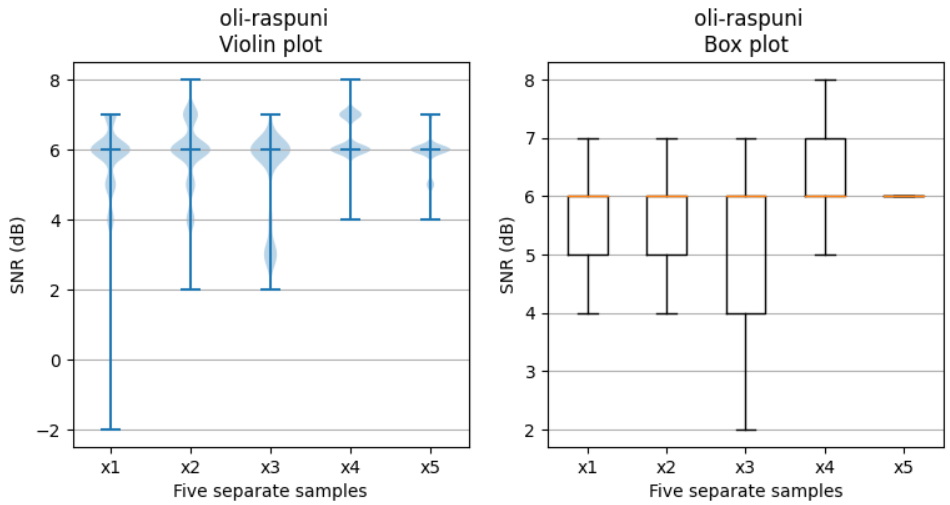


Figure B.6: A violin plot and a box plot of the five samples when the **default** reliability method was used. The metric measured and displayed is **SNR** in dB. Outliers have been removed from the box plot.