



ugr

Universidad
de **Granada**

DEGREE THESIS

TELECOMMUNICATIONS ENGINEERING

LoRaWAN Gateway and IoT Low-Cost Mote Prototype

Author

Angel Guzman-Martinez

Supervisors

Jorge Navarro-Ortiz

Sandra Sendra-Compte



SCHOOL OF INFORMATICS AND TELECOMMUNICATIONS ENGINEERING

Granada, June of 2017

LoRaWAN Gateway and IoT Low-Cost Mote Prototype

Author

Angel Guzman-Martinez

Supervisors

Jorge Navarro-Ortiz

Sandra Sendra-Compte



DPT. OF SIGNAL THEORY, TELEMATICS AND COMMUNICATIONS

Granada, June of 2017

Prototipo de un gateway LoRaWAN y una mota IoT low-cost para comunicaciones LPWAN

Ángel Guzmán Martínez

Palabras clave: Internet de las cosas, baja potencia, LoRaWAN, mota de bajo coste, *gateway*.

Resumen

El Internet de las cosas es un tema candente hoy en día que va tomando cada vez más relevancia. Sin embargo, esto trae consigo una serie de problemas que las redes convencionales no solventan correctamente, como el alcance y la duración de las baterías de los dispositivos conectados a estas redes.

Como solución a estos problemas, surgen las redes de baja potencia. Dichas redes están orientadas a proporcionar un mayor alcance con un menor consumo, a cambio de una menor velocidad de transmisión. Por este motivo, las redes de baja potencia son el candidato ideal para los dispositivos del internet de las cosas. Una de estas redes de baja potencia es LoRaWAN, que es en la que se basará este proyecto.

El objetivo de este proyecto es el desarrollo de dos dispositivos relacionados con LoRaWAN:

- Una mota que envíe información a través de una red LoRaWAN. En particular una mota de bajo coste, con el objetivo de demostrar que LoRaWAN es una tecnología al alcance de cualquier usuario. Aunque esto solo será posible si los usuarios disponen de un *gateway* LoRaWAN ya desplegado cercano a ellos.
- Un *gateway* LoRaWAN que sea capaz de recibir los paquetes LoRaWAN enviados por las motas y transmitirlos, en tiempo a real, a las aplicaciones, de forma que los usuarios puedan ver de forma remota la información que sus motas están transmitiendo.

LoRaWAN Gateway and IoT Low-Cost Mote Prototype

Angel Guzman-Martinez

Keywords: Internet of Things, LPWAN, LoRaWAN, low cost mote, gateway.

Abstract

Nowadays, the Internet of Things (IoT) is a trending topic that is growing in relevance every day. However, it brings out some problems that are not properly solved by the conventional networks, such as the range and battery life of the devices connected these networks.

The Low Power Wide Area Networks (LPWAN) appear as a solution to these problems. These networks provide an improved range and less power consumption, at the expense of the data rate. Thus, these low power networks are the ideal candidate for the IoT devices. One of these low power networks standards is called LoRaWAN, which is the one this project will be based on.

The goal of this project is to develop two LoRaWAN related devices:

- A mote capable of transmitting information through LoRaWAN. Particularly, a low cost mote in order to prove that LoRaWAN is a technology that everyone can use. Although this is only possible as long as there is an already deployed LoRaWAN gateway nearby.
- A LoRaWAN gateway that is able to receive the LoRaWAN packets sent by the motes and forward them, in real time, to the applications owned by the users, so that they can remotely see the data their motes are transmitting.

Yo, **Angel Guzman-Martinez**, alumno de la titulación Grado en Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Angel Guzmán Martínez

Granada, 21 de junio de 2017.

D. **Jorge Navarro Ortiz**, Profesor del Área de Ingeniería Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. **Sandra Sendra Compte**, Profesora del Área de Ingeniería Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Prototipo de un gateway LoRaWAN y una mota IoT low-cost para comunicaciones LPWAN***, ha sido realizado bajo su supervisión por **Ángel Guzmán Martínez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 21 de junio de 2017.

Los directores:

Jorge Navarro Ortiz

Sandra Sendra Compte

Acknowledgements

First off, I would like to start by thanking Jorge, my supervisor, because despite being extremely busy with his own work and being the director of many other projects, he still helped me greatly and got very involved with this project. In addition, I understand I might have been quite an annoyance to him with all of my questions, yet he was always friendly and willing to help me.

Secondly, I want to thank my parents and friends for encouraging me when I was feeling overwhelmed by all of the work I had to do, not only because of this project but also because of the many classes I was taking at the same time.

Lastly, a special mention to Sandra, Miguel Angel and Pablo, for letting us place the gateway on their terrace and for acting as our "personal maintenance team".

Contents

Glossary	21
1 Introduction	23
1.1 Context and Motivation	23
1.2 Goals and Reach of the Project	23
1.3 LoRaWAN's Current Situation	24
1.4 Structure of the document	24
2 LoRaWAN Overview	27
2.1 LoRa Modulation	27
2.1.1 Direct Sequence Spread Spectrum (DSSS)	27
2.1.2 LoRa Spread Spectrum	28
2.1.3 Main Properties of LoRa Modulation	30
2.1.4 Link Budget Example	31
2.2 LoRaWAN	32
2.2.1 Device Classes	33
2.2.2 Regional Parameters	35
2.2.3 Security	35
2.3 The Things Network	36
3 State of the Art	39
3.1 Gateways	39
3.1.1 LoRaWAN Single-Channel Packet Forwarder	39
3.1.2 Multi-channel DIY LoRaWAN Gateway	42
3.1.3 The Things Gateway	44
3.1.4 Other Multi-Channel Already-Built Gateways	45
3.2 Motes	46
3.2.1 Mote Without LoRaWAN Stack Implemented	46
3.2.2 Mote With Already Implemented LoRaWAN Stack	48
4 Planning and Cost Estimate	51
4.1 Development Stages	51
4.1.1 State of the Art Revision	51
4.1.2 Requirement Specifications	53

4.1.3	Design	53
4.1.4	Implementation	53
4.1.5	Testing	53
4.1.6	Documentation	53
4.2	Resources and Cost Estimate	53
4.2.1	Human Resources	53
4.2.2	Material Resources	54
4.3	Total Budget	56
5	Requirement Specifications	57
5.1	Gateway	57
5.1.1	Functional Requirements	57
5.1.2	Non-Functional Requirements	57
5.2	Mote	58
5.2.1	Function Requirements	58
5.2.2	Non-Functional Requirements	58
5.3	Application	59
5.3.1	Functional Requirements	59
5.3.2	Non-Functional Requirements	59
6	Design	61
6.1	Mote	61
6.1.1	Development Board	61
6.1.2	Transceiver Module	63
6.1.3	Antenna	63
6.2	Gateway	64
6.2.1	Embedded Linux Board	64
6.2.2	Concentrator	65
6.2.3	Antenna	65
6.2.4	Enclosure	66
7	Implementation	67
7.1	Application	67
7.2	Mote	69
7.2.1	Wiring	69
7.2.2	Software	71
7.3	Gateway	73
7.3.1	Wiring	73
7.3.2	Software	74
7.3.3	Registration	75
7.3.4	Enclosure	76
7.3.5	Location	78

8	Testing and Results	79
8.1	Sending Data to the Application	79
8.2	Gateway Coverage	81
8.2.1	Indoors	82
8.2.2	Outdoors	88
8.3	Channel Hopping Histogram	89
8.4	Spectrum Analysis	91
8.5	Power Consumption	91
8.5.1	Worst Case Scenario	93
8.5.2	Realistic Scenario	94
9	Conclusions	97
9.1	LoRa	97
9.2	LoRaWAN and TTN	97
9.3	The Gateway	97
9.4	Low Cost Mote	98
	Appendices	101
A	Arduino Program for the Mote	103
B	Matlab Histogram Scripts	109
B.1	Floor 5 Histogram	109
B.2	Floor 4 Histogram	109
B.3	Floor 3 Histogram	109
B.4	Floor 2 Histogram	110
B.5	Floor 1 Histogram	110
B.6	Floor 0 Histogram	110
B.7	Floor -1 Histogram	110
B.8	Channel Hopping Histogram	111
C	Matlab Functions Used for the Spectrum Analysis	113
C.1	Function That Captures and Stores the Spectrum	113
C.2	Function That Displays the Spectrum	115

List of Figures

1.1	World market for Internet connected devices forecast [1]. . . .	24
1.2	LoRaWAN gateways in Spain	25
2.1	DSSS Example [5].	28
2.2	Chirp signal in the time domain.	29
2.3	LoRa modulation in the frequency domain [6].	29
2.4	LoRaWAN Topology [8].	33
2.5	LoRaWAN Classes [2].	34
2.6	Class A receive windows [2].	34
2.7	LoRaWAN EU863-870MHz ISM band parameters [7].	35
2.8	LoRaWAN's security [9].	36
2.9	The Things Network's architecture	37
3.1	Single channel LoRaWAN gateway example [13].	40
3.2	Multi-channel DIY LoRaWAN gateway [14].	42
3.3	The Things Gateway[10].	44
4.1	Gantt chart.	52
6.1	Wemos D1 Mini [16].	62
6.2	NodeMCU.	62
6.3	Some of the different LoRa modules available[17].	63
6.4	LoRa BEE by Dragino [18].	64
6.5	Antenna and pigtail for iC880A concentrator [19].	65
7.1	Application configuration example.	68
7.2	Device configuration example.	68
7.3	Wemos D1 Mini pinout [16].	69
7.4	LoRa BEE pinout [18].	70
7.5	Mote after the wiring process.	71
7.6	Example of Arduino parameters configuration.	73
7.7	Arduino pin mapping.	73
7.8	Gateway after the wiring process.	75
7.9	Example of gateway registration.	76

7.10 Gateway status: connected.	76
7.11 Gateway privacy settings.	77
7.12 Gateway information.	77
7.13 Gateway enclosed.	78
8.1 Packets received at the gateway.	80
8.2 Fragment of the packet details at the gateway.	80
8.3 Packets received at the application.	81
8.4 Fragment of the packet details at the application.	82
8.5 Fragment of the log file for floor 0.	83
8.6 Floor 5 RSSI histogram.	84
8.7 Floor 4 RSSI histogram.	84
8.8 Floor 3 RSSI histogram.	85
8.9 Floor 2 RSSI histogram.	86
8.10 Floor 1 RSSI histogram.	86
8.11 Floor 0 RSSI histogram.	87
8.12 Floor -1 RSSI histogram.	88
8.13 Results of the outdoor coverage testing	89
8.14 Zoomed in results of the outdoor coverage testing	90
8.15 Histogram of the different channels used.	91
8.16 Spectrum centered on 868MHz when nothing is transmitting.	92
8.17 Spectrum centered on 868MHz when packets are being transmitted.	92
8.18 USB Tester after measuring for 32 minutes during worst case scenario.	93
8.19 USB Tester after measuring for one hour during realistic scenario.	94

List of Tables

3.1	Total cost for the single channel gateway.	40
3.2	Total cost for the multi-channel DIY gateway.	43
3.3	Cost estimate for the ESP8266 chip based mote.	47
3.4	Cost estimate for the LoRaWAN certified chip based mote. .	48
4.1	Distribution of days for the different development stages. . .	52
4.2	Human resources cost.	54
4.3	General purpose hardware cost.	54
4.4	Mote hardware cost.	55
4.5	Gateway hardware cost.	56
4.6	Total cost of the project.	56
7.1	Pin connections for ground and power input.	70
7.2	Pin connections for SPI bus.	70
7.3	Pin connections for I2C bus.	70
7.4	Pin connections between the Raspberry Pi and the concentrator.	74

Glossary

ABP Activation By Personalization.

ADR Adaptive Data Rate.

AES Advanced Encryption Standard.

BW Bandwidth.

CPU Central Processing Unit.

DIY Do It Yourself.

DR Data Rate.

DSSS Direct Sequence Spread Spectrum.

ETSI European Telecommunications Standards Institute.

EUI Extended Unique Identifier.

FHSS Frequency-Hopping Spread Spectrum.

FSK Frequency-Shift Keying.

GPIO General-Purpose Input/Output.

GPS Global Positioning System.

I2C Inter-Integrated Circuit.

IoT Internet of Things.

ISM band The industrial, scientific, and medical radio band.

LoRa Long Range.

LoRaWAN Long Range Wide Area Network.

LPWAN Low Power Wide Area Network.

MAC Media Access Control.

MIC Message Integrity Check.

MISO Master Input Slave Output.

MOSI Master Output Slave Input.

MQTT Message Queue Telemetry Transport.

NF Noise Figure.

OTAA Over The Air Activation.

RSSI Received Signal Strength Indicator.

SCK Serial Clock.

SCL Serial Clock Line.

SDA Serial Data Line.

SDR Software-Defined Radio.

SF Spreading Factor.

SIR Signal to Interference Ratio.

SNR Signal to Noise Ratio.

SPI Serial Peripheral Interface.

SS/NSS Slave Select.

TTN The Things Network.

WAN Wide Area Network.

Chapter 1

Introduction

1.1 Context and Motivation

Nowadays, the Internet of Things is a trending topic as the number of devices that are connected to the Internet is growing at incredible pace, see Figure 1.1. We see the appearance of new devices with Internet connection capabilities everywhere, which range from household appliances that can be accessed remotely to clothes that change their appearance based on the current weather. Usually, these devices are mobile, wireless and depend on batteries, which can be a problem with the regular WiFi networks, since a WiFi connection can be very taxing to the battery life. In order to address this problem, we are in need of another type of Internet network.

This is where Low Power Wide Area Networks (LPWANs) come into play. They are designed to make these Internet connections possible while allowing the battery to last longer than it would by using WiFi.

However, these LPWANs are not suitable for all IoT devices, as they have very small bandwidth and do not allow for high data rates. As a consequence, an IoT device that needs to transmit at a high data rate does not benefit from using a LPWAN.

On the other hand, a device like a mote, which usually only needs to send a little bit of data every so often, will make great use of a LPWAN, allowing its battery to last much longer, thus reducing its maintenance cost.

One of these LPWANs is called LoRaWAN, which makes use of the physical layer protocol LoRa, and is the one this project will be based on.

1.2 Goals and Reach of the Project

The goal of this project consists of building two LoRaWAN related devices:

1. A fully functional LoRaWAN gateway.

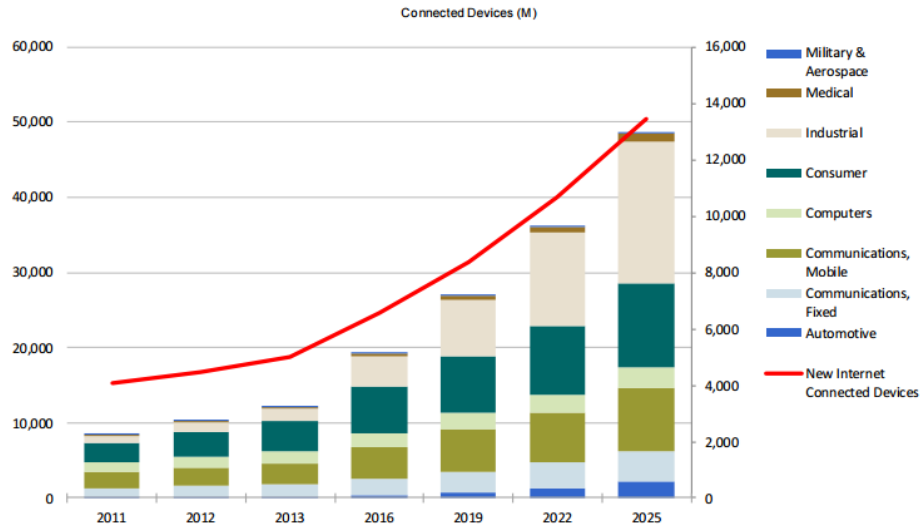


Figure 1.1: World market for Internet connected devices forecast [1].

2. A low cost mote capable of transmitting data to the gateway, as well as receiving data from it.

Regarding the mote, we want to make it cheap and affordable in order to prove that using a LoRaWAN network does not require any large economical effort, as long as there is an already deployed gateway in the area of interest.

With respect to the gateway, it will be made using a Raspberry Pi and a concentrator in order to provide a multi-channel service, thus complying with the LoRaWAN standard.

Lastly, when the project is completed, our intention is to set up the gateway on top of the school so it can be freely used by everyone around and be the one of the firsts LoRaWAN gateways in Granada, Spain.

1.3 LoRaWAN's Current Situation

LoRaWAN is a very recent technology, as LoRaWAN version 1.0 came up in January 2015 [2]. Due to this, there are very few LoRaWAN gateways deployed, especially in Spain, Figure 1.2.

As we will explain later, gateways play a crucial part in LoRaWAN. Accordingly, a lack of gateways shows a clear underdevelopment of this technology in Spain, which we hope will change in the coming years.

1.4 Structure of the document

This document consists of nine chapters which will now be briefly described:



Figure 1.2: LoRaWAN gateways registered by the TTN Mapper application [3] in Spain, June 2017.

1. **Introduction.** Brief explanation of the context and reasons that lead to the creation of this project.
2. **LoRaWAN Overview.** Overview of the main protocols and standards that take part in this project.
3. **State of the Art.** Analysis of alternative options which seek the same purpose as this project.
4. **Planning and Cost Estimate.** Rough estimate of the time required to carry out the project, as well as an approximation of the total cost.
5. **Requirement Specifications.** Quick summary of the functional and non functional requirements of the elements involved in the project.
6. **Design.** Explanation and reasoning behind the hardware and software used.
7. **Implementation.** Documentation of the process followed to set everything up for both hardware and software.
8. **Testing and Results.** Summary of the different tests performed once everything is working, as well as the results obtained from them.
9. **Conclusions.** Overview of what we have learned during the realization of this project.

In addition, there are three appendices at the end of the document:

1. **Appendix A.** Arduino program used for the mote.
2. **Appendix B.** Matlab scripts used to make the histograms.
3. **Appendix C.** Matlab functions used for the spectrum analysis.

Chapter 2

LoRaWAN Overview

In this chapter, we will introduce the protocols and technologies that make this project possible. Specifically:

1. LoRa. It is the physical layer protocol that LoRaWAN is based on, thus it is necessary to have a basic understand of it before moving onto LoRaWAN.
2. LoRaWAN. A network protocol and the core of this project. We will briefly describe what makes LoRaWAN unique and its properties.
3. The Things Network (TTN). A non-profit organization whose backend will be of great help for us.

2.1 LoRa Modulation

The following section is based on the Semtech's LoRa description [4].

LoRa stands for Long Range and is a Semtech proprietary spread spectrum modulation. Its main selling point consists of being able to gain range at the expense of data rate. What makes LoRa stand out from other modulation methods is its unique spread spectrum technique, which provides robustness against interferences and a very low minimum SNR for the receiver to be able to demodulate the signal.

2.1.1 Direct Sequence Spread Spectrum (DSSS)

This is the traditional way of applying the spread spectrum technique. An expanded signal is obtained by multiplying the original data signal with a spreading code (chip sequence). In the time domain, this spreading code has a much faster rate than the data signal, which translates into a wider bandwidth, beyond the original one, in the frequency domain.

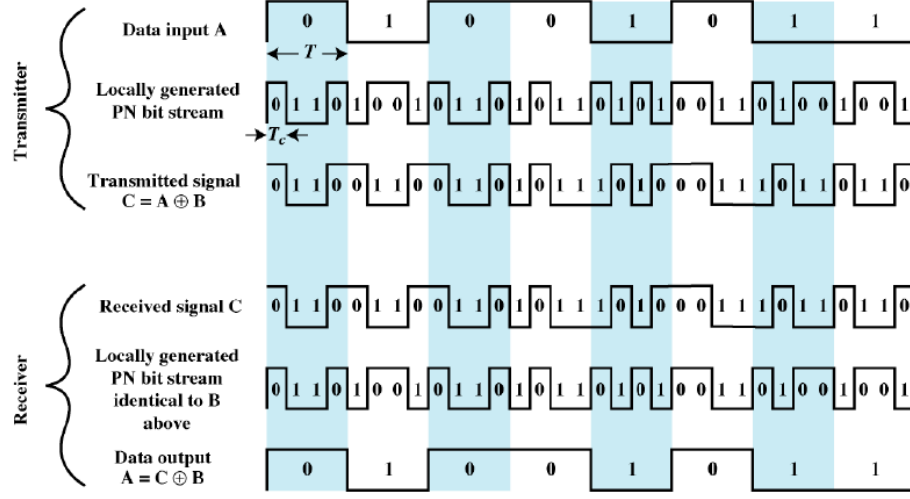


Figure 2.1: DSSS Example [5].

Then, said expanded signal can be recovered at the receiver by multiplying again with a replica of the spreading sequence generated at the receiver itself, Figure 2.1.

This method is widely used but it presents some problems for low-cost or power-constrained devices, as they require a highly accurate and expensive reference clock source. Furthermore, the longer the spreading code, the longer the time required by the receiver to perform a correlation over the entire length of the code sequence. This is problematic for devices that cannot always be on and need to be able to repeatedly and rapidly synchronize.

2.1.2 LoRa Spread Spectrum

Semtech's LoRa modulation provides a robust alternative to the traditional spread-spectrum techniques. Its strong points are its low-cost and low-power requirements.

In this case, the spreading of the spectrum is achieved by generating a chirp signal that continuously varies in frequency, Figure 2.2. This chirp signal is first sent several times without any data modulated onto it, as a synchronization mechanism, known as the preamble. The preamble also includes the transmission of the inverse chirp, which is used by the receiver to demodulate the signal. As a result, this method does not require a highly accurate source clock, unlike in DSSS, which helps reduce the cost of the receiver.

After the synchronization phase has been completed, the chirp signal is used to modulate the data signal, by shifting the phase of the chirp, Figure 2.3.

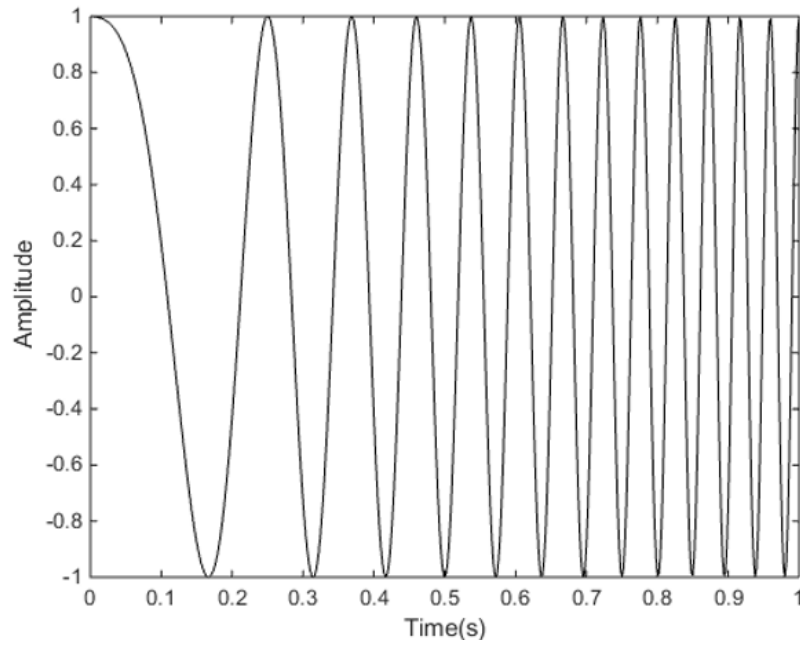


Figure 2.2: Chirp signal in the time domain.

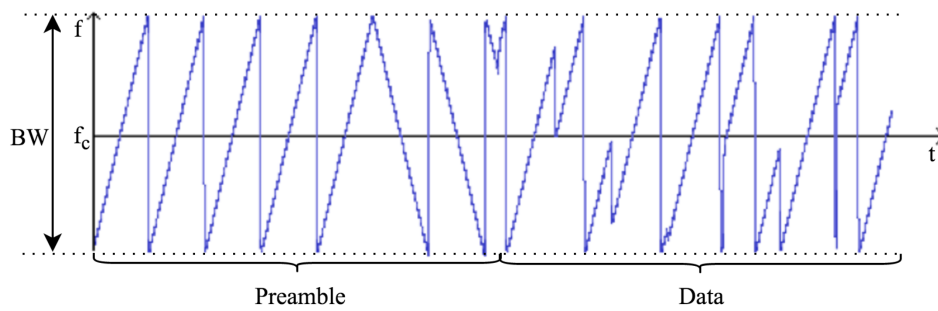


Figure 2.3: LoRa modulation in the frequency domain [6].

Furthermore, we can also adapt our data rate depending on what we need, as a result of the orthogonal spread spectrum factors that the LoRa modulation provides us. These spreading factors are what enable LoRa to perform a trade off between data rate and range. Likewise, the higher the spreading factor the more robust our signal is against interferences at the expense of lowering the data rate. For example, if we apply a lower spreading factor to our signal, it will be more vulnerable to interferences, thus lowering the range but also allowing us to transmit at a higher rate.

In addition, using a higher spreading factor also translates into a longer airtime for the signal, since the symbol period is defined by the following formula:

$$T_s = \frac{2^{SF}}{BW} \text{ seconds}$$

Lastly, the different spreading factors do not interfere with each other, as they are orthogonal with each other. As a result, we can transmit several signals over the same frequency range by using different spreading factors.

2.1.3 Main Properties of LoRa Modulation

Adaptability

We can adapt our data rate and range according to our needs by simply choosing the appropriate spreading factor. However, we need to understand that we cannot have both high data rate and long range at the same time.

Constant Envelope

Similarly to FSK, LoRa is a constant envelope modulation scheme. This means that the amplitude level of the signal is constant throughout its length, therefore the data contained within the signal is not related to its amplitude level, making it robust against possible nonlinear distortion caused by power amplifiers. As a result, we can use highly efficient non-linear power amplifiers. For reference, non constant envelope signals require a fully linear power amplifier, which is not ideal as this type of power amplifier tends to be very inefficient.

Robustness

Due to its asynchronous nature, a LoRa signal is very resistant to both in-band and out-of-band interference mechanisms. The LoRa symbol period is longer than the usual short-duration burst of FHSS systems, providing excellent immunity to interferences caused by Pulse Amplitude Modulation, which consists of several pulses of short duration that vary in amplitude.

Fading Resistant

Thanks to the chirp pulse being relatively broadband, the LoRa signal is resistant to multipath and fading.

Doppler Resistant

The Doppler effect introduces a relatively negligible shift in the time axis of the baseband signal, which makes LoRa ideal for mobile data communications.

Long Range Capability

For a fixed throughput and output power, the robustness to interference and fading mechanisms allows the LoRa signal to achieve a range about four times better than that of the conventional FSK.

2.1.4 Link Budget Example

Let us calculate a quick example to showcase the power of the LoRa modulation. In order to calculate the link budget, we have to account for the power transmitted and the sensitivity of the receiver, which gives us the maximum path loss allowed by the receiver:

$$PathLoss_{max}(dB) = Power_{Tx}(dB) - Sensitivity_{Rx}(dB) \quad (2.1)$$

Typically, we have to take into account the gain of the antennas and the loss caused by the connectors, but we are going to ignore these factors to simplify the example. It is also worth noting, when talking about LoRa modulation, the main limiting factor is the SNR rather than the SIR. This is caused by LoRa's robustness against interference but very low power transmission, making it vulnerable to noise.

Then we need to know the formula for the receiver sensitivity, which specifies the minimum signal power in order to be demodulated by the receiver:

$$Sensitivity(dBm) = -174 + 10 \log_{10}(BW) + NF + SNR \quad (2.2)$$

Where:

NF is the receiver noise figure (dB).

SNR is the minimum Signal-to-Noise ratio allowed by the receiver.

Next, if we assign some realistic values to these variables:

- Power Transmitted = 14dBm.

- SNR = -20.
- NF = 6dB.
- Bandwidth = 125kHz.

And apply these values to equations 2.2 and 2.1, we obtain:

$$Sensitivity = -137dBm \quad (2.3)$$

$$PathLoss_{max} = 14 + 137 = \mathbf{151dB} \quad (2.4)$$

This means we can lose about 150dB between the transmitted signal and the received signal and the receiver would still be able to demodulate it. In fact, if we take the free space path loss formula:

$$FSPL(dB) = 20 \log_{10}(d) + 20 \log_{10}(f) + 32.44$$

Where:

d is the distance of the receiver from the transmitter (km)

f is the signal frequency (MHz)

We obtain that the signal could travel almost **800km** and the receiver would still manage to demodulate it. Obviously, this is too good to be true, because in reality we do not work with free space and there are other effects like multipath, Fresnel zones, fading, etc., that also affect our signal. Nonetheless, this example showcases LoRa's capabilities and why it is a strong option for most of the low-power networks.

2.2 LoRaWAN

The following section is based on the LoRaWAN specification [2] and Regional Parameters [7] defined by the LoRa Alliance ¹.

LoRaWAN is a network protocol optimized for battery-powered end-devices that are either mobile or static.

LoRaWAN network topology usually consists of a star-of-stars involving end-devices, gateways and a central network server. On one hand, gateways are connected to the network server using standard IP connections. On the other hand, end-devices make use of single-hop LoRa or FSK modulation to one or multiple gateways, Figure 2.4.

¹LoRa Alliance is an open, non-profit association whose goal is to standardize Low Power Area Networks (LPWAN) and guarantee interoperability between operators in one open global standard.

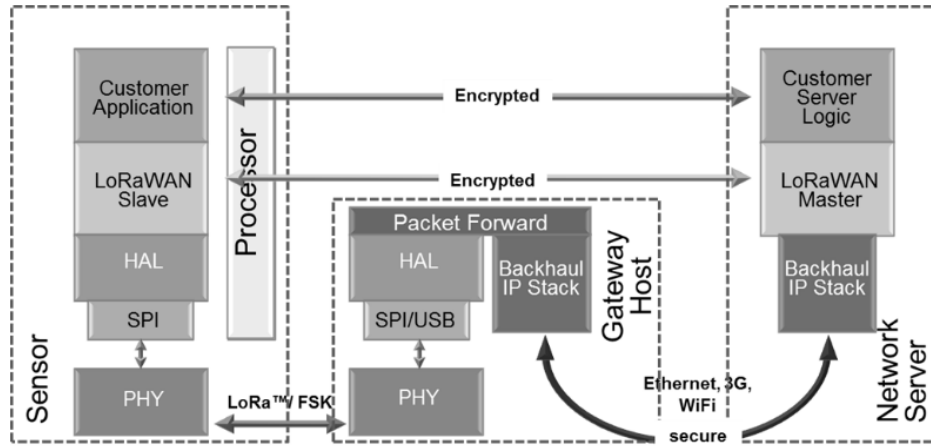


Figure 2.4: LoRaWAN Topology [8].

Communication between end-devices and gateways is spread out on different frequency channels and data rates. LoRa data rates range from 0.3kbps to 50kbps. In order to maximize both battery life of the end-devices and overall network capacity, the LoRa network is able to manage each end-device individually via an ADR scheme.

The end-devices must respect the following rules in order to transmit on any channel available at any time:

- The end-device changes channel in a pseudo-random way for every transmission.
- The end-device respects the maximum transmit duty cycle relative to the sub-band used and local regulations. For example, if a device just transmitted a 0.5 seconds long frame on one of the default channels, in order to comply with the 1% duty cycle, that device cannot transmit on that whole sub-band (868-868.6MHz) during 49.5 seconds.
- The end-device respects the maximum transmit duration relative to the sub-band used and local regulations. In our case, there is no dwell time limitation for the EU863-870 ISM band.

2.2.1 Device Classes

A LoRa network distinguishes between a basic LoRaWAN class (named class A) and optional features (class B and class C), Figure 2.5:

- Bi-directional end-devices (Class A): Each end-device's uplink transmission is followed by two short downlink receive windows, Figure 2.6. The end-device is able to transmit based on its own needs with a small

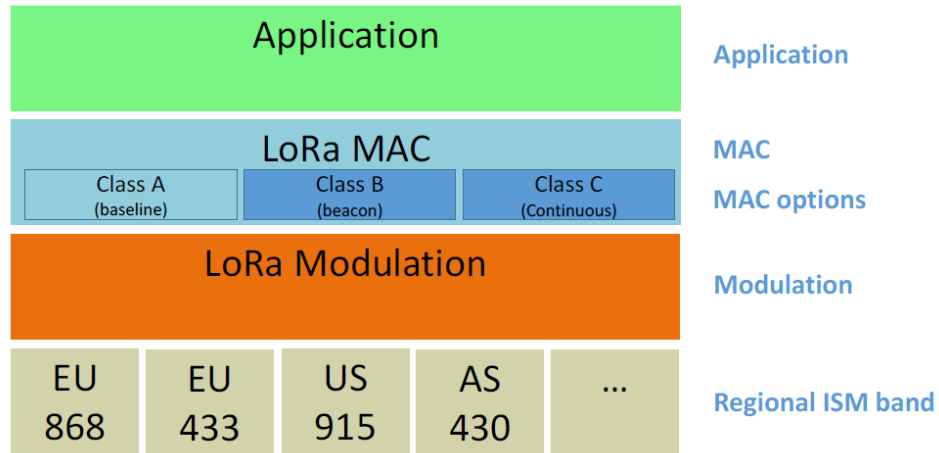


Figure 2.5: LoRaWAN Classes [2].

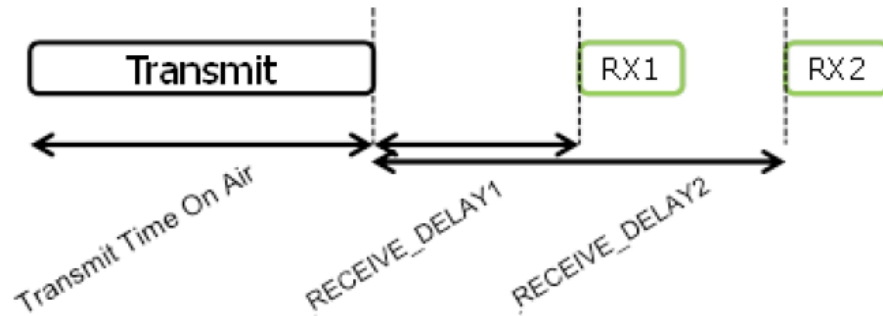


Figure 2.6: Class A receive windows [2].

random variation (ALOHA-type protocol). Downlink communications from the server at any other time will have to wait until the next scheduled uplink. This is the ideal class for applications that only require a response from the server shortly after transmitting.

- Bi-directional end-devices with scheduled receive slots (Class B): In addition to the Class A random receive windows, Class B devices open extra receive windows at scheduled times. To achieve this, the end-device has to be synchronized with the gateway, which is done by using a time synchronized beacon sent by the gateway. This way the server knows when the end-device is listening.
- Bi-directional end-devices with maximal receive slots (Class C): This class allows for nearly continuous open receive windows that are only

Modulation	Bandwidth [kHz]	Channel Frequency [MHz]	FSK Bitrate or LoRa DR / Bitrate	Nb Channels	Duty cycle
LoRa	125	868.10 868.30 868.50	DR0 to DR5 / 0.3-5 kbps	3	<1%

Figure 2.7: LoRaWAN EU863-870MHz ISM band parameters [7].

closed when transmitting. This class is the most power-demanding of the 3 classes, but it also offers the lowest latency for server to end-device communication.

2.2.2 Regional Parameters

To be able to ensure interoperability between different LoRaWAN networks, the LoRaWAN specification defines the parameters that should be followed for each region.

In our case, we will be using the European 863-870 MHz ISM band. In Figure 2.7 we can see the different data rates allowed for the European ISM band. These data rates translate into the different bitrates a transmitter can choose from. These data rates are directly related to the spreading factors seen previously in LoRa modulation.

In order to access the physical medium of these free-license bands, the European Telecommunications Standards Institute (ETSI) imposes some restrictions such as maximum time the transmitter can be on or the maximum time a transmitter can transmit per hour. Currently, the LoRaWAN specification uses duty-cycled limited transmissions to comply with the ETSI regulations.

2.2.3 Security

This section will briefly explain the security fundamentals of LoRaWAN.

The main principle is that LoRaWAN makes use of three different 128 bits keys:

- **Application key.** Generated using AES-128 and it is used to generate the other two keys.
- **Network session key.** Derived from the Application Key. This key is known by the network in order to prove and verify the integrity and authenticity of the packets.
- **Application session key.** Derived from the Application Key. Ensures encryption end to end for the application payload.

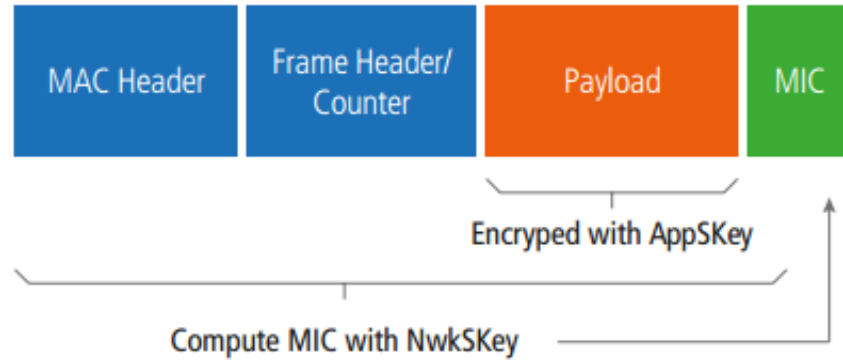


Figure 2.8: LoRaWAN's security [9].

In addition, these keys can be managed in two ways:

- **Activation By Personalization.** The end-device skips the join procedure, which means the session keys become static and do not change over different sessions.
- **Over The Air Activation.** The end-device performs a join procedure with the network. As a result, the session keys are generated dynamically for each session.

Moreover, a frame counter is also used to prevent replay attacks. If a packet is received with a frame counter lower than the expected frame counter, the packet is dropped.

Lastly, it is recommended that the session keys are generated by a separate entity from the network, in order to ensure the network cannot decrypt the application payloads.

In conclusion, from how the session keys are used and from Figure 2.8, we can see that the network session key is only used within the network while the application session key is used end to end. This means that the integrity of the payload is not guaranteed once it leaves the network to reach the application, but the confidentiality is.

2.3 The Things Network

In this project we are building a gateway and an end-device, yet a LoRaWAN network requires another element called network server.

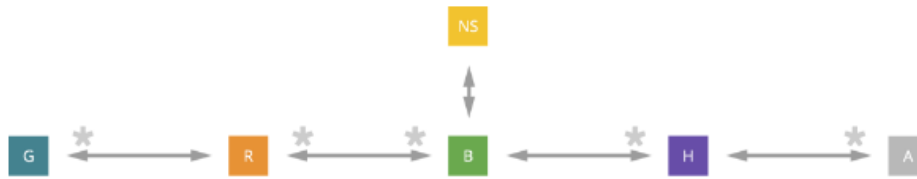


Figure 2.9: The Things Network’s architecture: Gateway, Router, Broker, Network Server, Handler and Application [10].

A network server is an element of LoRaWAN that performs several tasks such as integrity check, frame counter update, downlink template generation, etc.

Sine we do not dispose of a network server nor plan on implementing one, we will be using The Things Network backend, which provides not only the network server but all of the elements in between needed to connect to it, Figure 2.9.

Furthermore, TTN is a community driven project, so it provides its features for free as long as we provide a gateway to connect to it. However, connecting to TTN adds a limitation in the form of "Fair access policy" [11]. This policy states the following:

- Maximum of 30 seconds of uplink time on air, per day and per device.
- Maximum of 10 downlink messages per day, per device.
- Under 12 bytes payload.

For now, this policy is not enforced. In fact, we will most likely not respect it while testing our devices. Nevertheless, it should be respected after the testing phase is over.

Chapter 3

State of the Art

In this chapter we will present and analyze the different possibilities we have to build a LoRaWAN gateway and a mote. This analysis will involve several parameters such as hardware requirements, price and TTN support among others, which will help us choose the best option for the project.

3.1 Gateways

This section will go through the different LoRaWAN gateways available and compare their characteristics such as price, hardware requirements, software requirements, LoRaWAN compatibility, etc. To be precise, we will study the following gateway possibilities:

- Single-channel gateway.
- Multi-channel DIY gateway.
- The Things Gateway.
- Other multi-channel already-built gateways.

The Things gateway, although being a multi-channel already-built gateway, is the official gateway distributed by TTN. Therefore, we think it is worth a deeper analysis and its own section.

3.1.1 LoRaWAN Single-Channel Packet Forwarder

This is the simplest and cheapest version of a LoRaWAN gateway, Figure 3.1. However, it has very limited features and it is only able to listen on one channel [12].

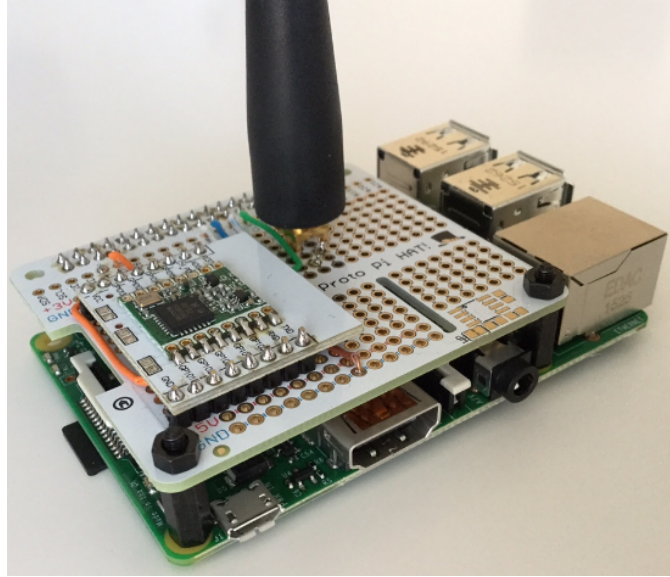


Figure 3.1: Single channel LoRaWAN gateway example [13].

Concept	Unit cost	Quantity	Total
Raspberry Pi	39.90 €	1	39.90 €
RF transceiver ¹	5.58 €	1	5.58 €
Antenna ²	5.58 €	1	5.58 €
Total			51.06 €

Table 3.1: Total cost for the single channel gateway.

Hardware Requirements

In order to build this gateway we need the following items:

- Raspberry Pi (any model).
- Radio frequency transceiver.
- Antenna.

From this list we can see why this is the cheapest option, as it requires very little hardware, at the cost of only being able to listen on a single channel.

Cost Estimate

Table 3.1 shows the estimated of the cost for each hardware component and the total cost they add up to.

For a total cost of about 50 €, this is a very affordable gateway.

¹SX1272 used as reference.

²3dBi and 868MHz antenna as reference.

Software

There is one repository with all the code needed for the gateway [12], which consists of a C++ program you have to run as root in the Raspberry Pi as well as some configurable parameters. Unfortunately, this repository is deprecated and no longer up to date to use on TTN.

Supported Features

The following list contains the most important features that are fully developed for this type of gateway.

- Able to listen on configurable frequency.
- Spreading factors from SF7 to SF12.
- Status updates.
- Can forward to two routers.

Not Supported Features

Here we will list the features that either never made it into the final product or were never intended to be implemented.

- FSK modulation.
- Downstream messages.

This gateway not being able to retrieve downstream messages in addition to being single channel makes it not fully LoRaWAN compatible, thus it is not officially supported by TTN.

Main Advantages

According to the previous points, these are the main advantages of the single channel LoRaWAN gateway.

- Very cheap components.
- Easy to build.
- Great for getting started with LoRaWAN.

Final Notes

Due to its single channel nature, it has less than 2% of the capacity of a real multi-channel gateway, in addition to not being supported by TTN and not fully LoRaWAN compatible.

In conclusion, this type of gateway is best used for testing and educational purposes.



Figure 3.2: Multi-channel DIY LoRaWAN gateway [14].

3.1.2 Multi-channel DIY LoRaWAN Gateway

The multi-channel DIY LoRaWAN gateway, Figure 3.2, is a fully LoRaWAN compatible gateway and officially supported by TTN. Its cost is greater than the previous alternative, but it offers a wider range of features [14].

Hardware Requirements

This is the hardware equipment needed to build the gateway:

- Embedded Linux board.
- iC880A concentrator board.
- Pigtail for antenna.
- Antenna.
- Raspberry Pi to iC880A interface.

The most expensive element is the iC880A concentrator board, which is the piece that enables the gateway to provide multi-channel features.

Cost Estimate

Table 3.2 shows the cost estimate for the multi-channel DIY gateway.

³Raspberry Pi 3 model B used as reference.

⁴iC880A antenna as reference.

⁵Double female jumper wire as the reference.

Concept	Unit cost	Quantity	Total
Embedded Linux board ³	39.90 €	1	39.90 €
iC880A concentrator	189 €	1	189 €
Antenna ⁴	6.50 €	1	6.50 €
Pigtail for antenna	6.50 €	1	6.50 €
RPi to iC880A interface ⁵	0.20 €	7	1.40 €
Total			243.30 €

Table 3.2: Total cost for the multi-channel DIY gateway.

As we can see, this type of gateway is several times more expensive than the single channel version, mainly due to the need of using a concentrator.

Software

As a result of this gateway being officially supported by TTN, they provide us with an installation script [14].

Therefore, assuming we already have our Raspberry Pi ready to work, as in having an operating system installed and updated, we only need to download and run the script as well as configure a few parameters such as our contact name, location, email, etc.

Main Advantages

These are the main advantages of the DIY multi-channel LoRaWAN gateway:

- Compatible and fully supported by TTN.
- Instructions on how to build it provided by TTN, both for the hardware and the software.

Final Notes

Since TTN is basically a crowfunded network that relies on the community to set up the gateways, they want to provide us with good and easy instructions on how to set them up. As consequence, building this type of gateway is fairly easy even for someone who is new to LoRaWAN.

Overall, a very solid option that directly contributes to TTN and the one we have chosen for this project, as it is the cheapest multi-channel gateway as well as being fully LoRaWAN compatible.



Figure 3.3: The Things Gateway[10].

3.1.3 The Things Gateway

This is the official gateway distributed by TTN, Figure 3.3. It is fully LoRaWAN compatible and especially designed to work with TTN's backend and it comes fully built and ready to use [10].

Hardware Requirements

This gateway comes fully built as a pack. All the necessary equipment needed for the set up is included in the package.

Cost Estimate

As mentioned before, the gateway is shipped as one pack, which costs **300 €**. Since we do not need anything else to set up the gateway, the price of the pack accounts for the total cost associated to this gateway.

Software

Same as the hardware, the software comes already set up, meaning that we do not have to run any script or implement any program in order for the gateway to start working.

In addition, the only process required is the registration of the gateway over TTN, unless we use a private LoRaWAN network which does not make

use of TTN's backend, thus not requiring any kind of registration.

Main Advantages

These are the main advantages of The Things Gateway.

- Easy to install.
- Especially designed to work with TTN.
- Includes a bluetooth modem for indoor IoT connections.
- Runs on open hardware and open software.
- Can serve up to 10,000 nodes.

Final Notes

It is a great option if you do not have the time to build your own gateway. On top of that, the registration process over TTN is very easy and intuitive even for someone not familiar with LoRaWAN. Furthermore, being especially designed for TTN ensures an smooth experience when using TTN's backend.

Overall, an interesting choice for an easy to install and use gateway. It is also the most expensive option, although not by a large margin.

3.1.4 Other Multi-Channel Already-Built Gateways

There are a few other gateways of this type apart from The Things Gateway. Some of these other gateways are:

- MultiTech Conduit.
- Lorrier LR2.
- Link Labs LL-BS-8.

Although they would be valid options for our project, we will soon see that their high price will be reason behind discarding them.

MultiTech Conduit

This gateway supports multiple protocols, so in order to use it for LoRa we also have to buy the appropriate LoRa cards. This results in quite a expensive LoRaWAN gateway. The price for the gateway itself is 442 €, while the price for one LoRa card is 159 €. This gives us a price of **601 €**, which is almost double the price of The Things Gateway.

Lorrier LR2

The Lorrier gateway is fairly similar to the DIY multi-channel gateway, as it is built using using an iC880A concentrator and a BeagleBone Green. Moreover, it is designed to be an outdoors gateway, thus it is enclosed in an IP66 metal case. Lastly, it has a price of **575 €**, which is quite high but it includes the price of the enclosure as well.

Link Labs LL-BS-8

This gateway from Link Labs is based on a Linux board and with a dual core CPU of 1GHz. Additionally, it is worth mentioning that it supports cellular connections. Price of **822 €**, the most expensive gateway presented in this chapter.

3.2 Motes

We will now go through the different possibilities for motes, and choose the one that best suits our project.

Most importantly, we have to remember that we are looking for a low-cost option, so price will be a very important factor when choosing the mote.

Furthermore, similarly to the gateways we will only discuss the two main types of LoRaWAN motes:

- Mote Without LoRaWAN Stack Implemented.
- Mote With Already Implemented LoRaWAN Stack.

3.2.1 Mote Without LoRaWAN Stack Implemented

This mote does not come with the MAC layer already implemented, which means we have to do it ourselves. However, there are some libraries with already implemented LoRaWAN MAC functions which will ease this task.

Hardware Requirements

These are the hardware pieces we need to build the mote:

- Board with chip ESP8266.
- Antenna.
- Micro-USB cable.
- Breadboard or shell.

Concept	Unit cost	Quantity	Total
Board with chip ESP8266 ⁶	2.72 €	1	2.72 €
Antenna ⁷	12.38 €	1	12.38 €
Micro-USB cable	4 €	1	4 €
Breadboard	6 €	1	6 €
Total			25.1 €

Table 3.3: Cost estimate for the ESP8266 chip based mote.

Cost Estimate

According to the hardware required, Table 4.4 shows the estimate of the price for this mote.

At the expense of having to program it ourselves, this mote comes at quite a low price.

Software

For this mote we have to implement the software ourselves, however it is easy to find already programmed LoRaWAN examples made by the community. These examples together with the LMIC-Arduino library, which contains all the necessary functions to set up the LoRaWAN MAC, make the software implementation much more manageable.

Main Advantages

Here we are going to list the strong points of this type of mote.

- Very affordable.
- Small size.

While it does not have many advantages, the very low price is what we are looking for, which outweighs its disadvantages.

Final Notes

This is the mote we have chosen for the project, the main reason being its price. Although we have to program it ourselves, the minimal hardware requirements makes this mote the ideal candidate for the low-cost mote we are looking for.

⁶Board Wemos D1 Mini used as reference for the price.

⁷Dragino LoRa BEE used as reference for the price.

Concept	Unit cost	Quantity	Total
Certified chip ⁸	11.95 €	1	11.95 €
Antenna ⁹	12.38 €	1	12.38 €
Micro-USB cable	4 €	1	4 €
Breadboard	6 €	1	6 €
Total			34.33 €

Table 3.4: Cost estimate for the LoRaWAN certified chip based mote.

3.2.2 Mote With Already Implemented LoRaWAN Stack

This mote comes with an already implemented LoRaWAN MAC layer, at the expense of a higher price.

The chip used in this type of mote has to pass the LoRa Alliance certification test, which ensures an easy integration into any LoRaWAN network.

Hardware Requirements

Here is the list of the different hardware elements required for this type of mote:

- LoRaWAN Chip.
- Antenna.
- Breadboard or shell.
- Micro USB cable.

Cost Estimate

Table 3.4 shows an estimate of the price based on the hardware requirements.

In summary, it is significantly more expensive than the previous option, which makes it the main disadvantage for this type of mote.

Software

This type of motes come with that a fully integrated LoRaWAN chip, meaning we do not have to implement any kind of program ourselves. Moreover, some chips may even include a text based interface for configuration purposes.

⁸RN2483 used as reference for the price.

⁹Dragino LoRa BEE used as reference for the price.

Main Advantages

The take away features of this type of mote.

- MAC layer already implemented.
- Less development time required.
- Easy LoRaWAN network integration.

Final Notes

In conclusion, the main strong point is the easy integration as counterpart for the higher price. Therefore, it is very reasonable choice to minimize development time. However, its higher price makes it unappealing for our project, since we are committed to building a mote with very low budget.

Chapter 4

Planning and Cost Estimate

In this chapter we will first describe the different stages involved in the development of this project, followed by a Gantt chart that shows the time required for each one of them.

In addition, this chapter will also contain an estimate for the total cost of the project, which be obtained by calculating four independent budgets:

1. **Human resources.** Rough estimate of the cost associated to the work hours by the people involved.
2. **General purpose hardware.** Element or elements needed throughout the whole project.
3. **Gateway budget.** We will estimate its cost based on the individual pieces that conform the gateway.
4. **Mote budget.** Similarly to the gateway, we will estimate its cost by adding up the prices for the individual pieces.

Lastly, we will add those four budgets together to obtain the total estimated cost for the project as a whole.

4.1 Development Stages

This section will contain the distribution of days for each development stage, Table 4.1 and Figure 4.1, followed by a brief description of them.

4.1.1 State of the Art Revision

In order to make an adequate and informed decision for the project, we must study the already existing solutions.

Development stage	Duration	Start	End
State of the art revision	25 days	10/11/2016	05/12/2016
Specifications	24 days	05/12/2016	30/12/2016
Design	45 days	30/12/2016	13/02/2017
Implementation	60 days	13/02/2017	15/04/2017
Testing and results	28 days	15/04/2017	13/05/2017
Documentation	183 days	10/11/2016	13/05/2017

Table 4.1: Distribution of days for the different development stages.

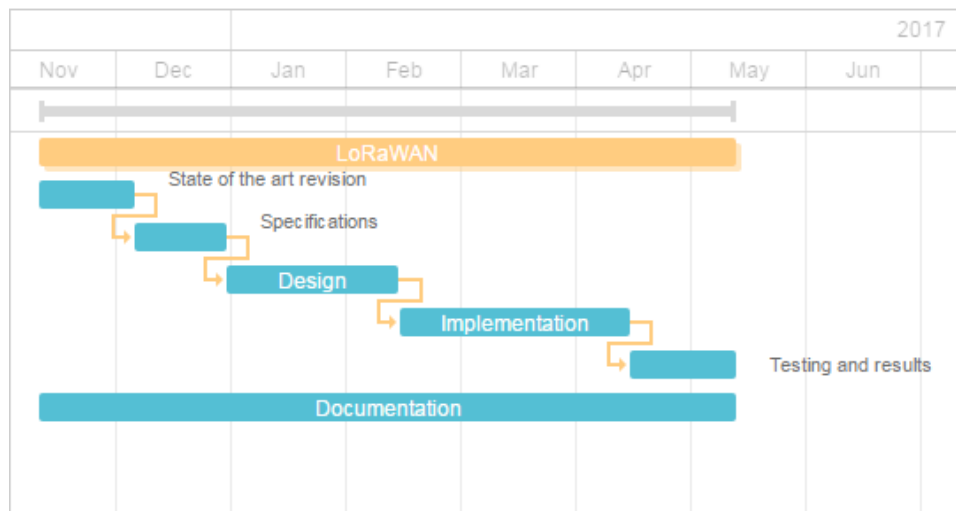


Figure 4.1: Gantt chart.

4.1.2 Requirement Specifications

We will define the different functional and non functional requirements that both the mote and the gateway should meet.

4.1.3 Design

Aiming to meet the previously defined specifications, we will work on choosing the different hardware and software elements.

4.1.4 Implementation

Once the design phase is completed, we will proceed with its implementation.

4.1.5 Testing

Lastly, we are now able to test our devices and ascertain that they work as we expected.

4.1.6 Documentation

This phase consists on the writing of this document and, although it is placed as the last phase, is carried out in parallel alongside the rest of the previous phases.

4.2 Resources and Cost Estimate

This section will study the resources employed for this project, which will be split in human, hardware and software resources. Additionally, an estimate of the cost associated to the elements in the different categories will also be included.

4.2.1 Human Resources

The following people have taken part in the accomplishment of this project:

- Jorge Navarro-Ortiz, associate professor of the Department of Signal Theory, Telematics and Communications of the University of Granada, as thesis supervisor.
- Sandra Sendra-Compte, associate professor of the Department of Signal Theory, Telematics and Communications of the University of Granada, as thesis supervisor.
- Angel Guzman-Martinez, student of the School of Informatics and Telecommunications Engineering of the University of Granada.

Concept	Cost/time	Quantity	Total
Project work	20 €/h	704 hours	14,080 €
Tutorship	50 €/h	16 hours	800 €
Total			14,880 €

Table 4.2: Human resources cost.

Concept	Unit cost	Average lifespan	Time used	Total
Personal laptop	700 €	4 years	183 days	87.74 €
Total				87.74 €

Table 4.3: General purpose hardware cost.

Currently in Spain, disclosing a salary/price of reference for any profession or service is forbidden by law [15]. Therefore, this category will be based on the assumption that a telecommunication engineer, usually earns no less than 20 € per hour and no more than 50 € per hour. From there, we have decided to assign 20 €/h to Angel Guzman-Martinez and 50 €/h to Jorge Navarro-Ortiz and Sandra Sendra-Compte.

In addition, we need to make a rough estimate of the total amount of hours that each of the participants will spend on the project:

- Angel Guzman-Martinez: 4 hours a day during 8 months excluding weekends. This gives us, approximately, 704 hours.
- Jorge Navarro-Ortiz: 8 hours in total of tutorship.
- Sandra Sendra-Compte: 8 hours in total of tutorship.

Gathering all of the previous data, we have presented the results in Table 4.2, where we obtain a total cost of **15,580 €**.

4.2.2 Material Resources

Here we will present the necessary physical elements for the execution of the project, which will be divided into three categories: hardware for the mote, for the gateway and a the general purpose hardware which will be used for the project as a whole.

General purpose

The items in this section will be used throughout the entirety of the project:

- Personal laptop.

The result from this category is shown in Table 4.3. From this table, **700 €** is the total cost obtained.

Concept	Unit cost	Quantity	Total
Wemos D1 Mini	2.72 €	1	2.72 €
Dragino LoRa BEE	12.38 €	1	12.38 €
Micro-USB cable	4 €	1	4 €
Breadboard	6 €	1	6 €
Wemos to LoRa BEE interface ¹	0.20 €	9	1.8 €
Total			26.9 €

Table 4.4: Mote hardware cost.

Mote

The following list presents us with the required materials to build the low-cost mote:

- Wemos D1 Mini.
- Dragino LoRa BEE.
- Micro-USB cable.
- Breadboard or shell.
- Wemos to LoRa BEE interface. In the case we do not use a shell.

As we mentioned before, the total price for the mote is an important result. From Table 4.4, we can conclude that our goal of building a low-cost mote is achievable, as its total cost is only of **25.1 €²**. We consider this a very affordable price for anyone interested in using LoRaWAN.

Gateway

Here we list what we need to build the LoRaWAN gateway:

- Embedded Linux board.
- iC880A concentrator.
- Antenna.
- Pigtail for antenna.
- Board to iC880A interface.
- Right angle USB connector.
- IP54 metal enclosure.

Concept	Unit cost	Quantity	Total
Embedded Linux board ³	39.90 €	1	39.90 €
iC880A concentrator	189 €	1	189 €
Antenna ⁴	6.50 €	1	6.50 €
Pigtail for antenna	6.50 €	1	6.50 €
Board to iC880A interface ⁵	0.20 €	7	1.40 €
Right angle USB connector	2.99 €	1	2.99 €
IP54 metal enclosure	6.36 €	1	3.36 €
Total			249.65 €

Table 4.5: Gateway hardware cost⁶.

Table 4.5 shows the result for the cost of the gateway, with the total cost being **249.65 €**.

4.3 Total Budget

To wrap up, we are now able to make an estimate of the total cost of the project based on the tables for the four different budgets defined previously.

Concept	Cost
Human	14,480 €
General purpose	87.74 €
Hardware mote	26.9 €
Hardware gateway	249.65 €
Total	14,844.29 €

Table 4.6: Total cost of the project.

Finally, from Table 4.6, the total cost estimated for this project is **14,844.29 €**, FOURTEEN THOUSAND EIGHT HUNDRED AND FORTY FOUR WITH TWENTY NINE EUROS.

¹Double male jumper wire used as reference.

²Assuming we already have a laptop available, which is needed for setting up the mote's software.

³Raspberry Pi 3 model B used as reference.

⁴iC880A antenna used as reference.

⁵Double female jumper wire used as reference.

⁶Cost for the initial set up of the Raspberry Pi (Ethernet/HDMI cable, external keyboard...) not included.

Chapter 5

Requirement Specifications

This chapter will talk about the functional and non-functional requirements the gateway and the mote have to fulfill in order for the project to be considered successful.

Furthermore, we have to remember that the final destination of the data, sent by the mote and forwarded by the gateway, is an application. Similarly to the mote and the gateway, the application also has to meet some requirements in order for us to recover useful information. Therefore, the application requirements will also be described in this chapter.

5.1 Gateway

Here we define the requirements we aim to meet with the gateway once it is working. Although some of the requirements are not fully necessary for the gateway to work, all of them should be met for a quality experience when using it.

5.1.1 Functional Requirements

- Able to receive packets over the LoRa physical layer.
- Able to forward the received packets over to TTN via either WiFi or Ethernet, in order to reach the application.

5.1.2 Non-Functional Requirements

- Fully LoRaWAN compatible.
- Supported by TTN, since we aim to forward the packets through it.
- Wide coverage.
- Reasonably cheap. Only the necessary features.

- Must provide security throughout the entire communication process.
- Availability. It should not have long periods of downtime.
- Easy to access and maintain.
- Efficient, as to make the power consumption as low as possible.
- Must support interoperability, as to be able to receive and forward packets from a wide range of different end-devices.
- Must support concurrency, the gateway should be able to handle several motes transmitting simultaneously.
- Scalable. The gateway must be able to handle the exponential growth of IoT devices.

5.2 Mote

In this section we will list the requirements our mote should meet after the implementation phase is done. Again, we do not have to meet all of them for the mote to work, in fact it would be fine for testing purposes not to meet some of them. However, we want to build a mote that closely resembles one that would be used in a real scenario, hence why all of the requirements should be met.

5.2.1 Function Requirements

- Able to transmit and receive packets over LoRa as the physical layer.
- Must be able to encrypt the packets before sending them.
- Will allow customizable payloads, such as a simple string or data from a sensor.
- Will have the option to change the spreading factor used, as to adapt to different distances and data rates
- Capable of changing the transmission frequency. In order to comply with TTN's fair access policy.

5.2.2 Non-Functional Requirements

- It must be cheap and affordable.
- Power efficient. A very important aspect, since in a real scenario the mote will be powered by a battery.

5.3 Application

Lastly, here we list the requirements that the application must meet. Moreover, it is important that the application meets these basics requirements, as that will help us perform the testing phase properly.

5.3.1 Functional Requirements

- Able to decrypt the payload and display the content in plain text.
- Allows to change the encryption keys in case we need it.
- Must not be restricted to one device. It needs to be capable of receiving packets from different end-devices. Therefore, it also must be able to distinguish packets from different end-devices.

5.3.2 Non-Functional Requirements

- Secure. We will need to provide our credentials before having access to the data.
- Availability. Similarly to the gateway, it must have as little downtime as possible.
- Readability. The data should be easy to interpret.
- Informative. It should display the information for different parameters, such as RSSI or SNR.

Chapter 6

Design

This chapter will explain the process followed to design both the gateway and the mote, describing the different hardware elements employed as well as the reasoning behind them.

In addition, we have to keep in mind that the goal of this design process is to build a mote and a gateway that meet the requirement specifications defined for them in the previous chapter.

6.1 Mote

First, we will start with the design of the mote. Note that the design does not include any type of sensor, as it is not needed to test that everything works together. In the case we wanted to add a sensor, we would have to perform the necessary wiring to the development board and adapt the code, so that the payload contains the information gathered by the sensor.

6.1.1 Development Board

As we mentioned in the state of the art chapter, we decided to build a mote that does not come with the LoRaWAN MAC layer already implemented in order to minimize the cost as much as possible. Therefore, when designing the mote, we mainly considered two boards:

- Wemos D1 Mini, Figure 6.1.
- NodeMCU, Figure 6.2.

These two boards are very cheap and they both come with the ESP12 chip, which contains the WiFi module ESP8266. However, this chip can be programmed to work with LoRa so we can use it to implement the MAC layer.

From this point and although both boards are very similar and cost about the same price, we decided to go with the Wemos D1 Mini because

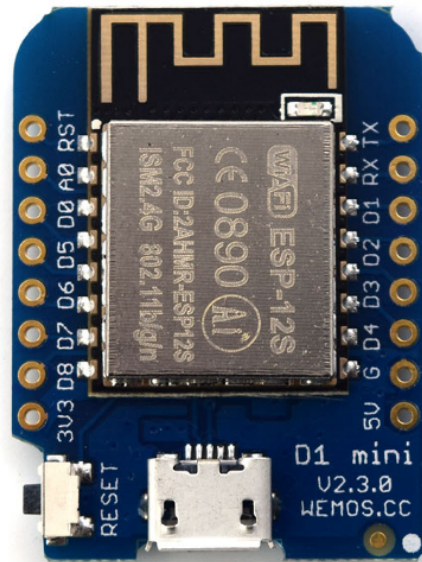


Figure 6.1: Wemos D1 Mini [16].

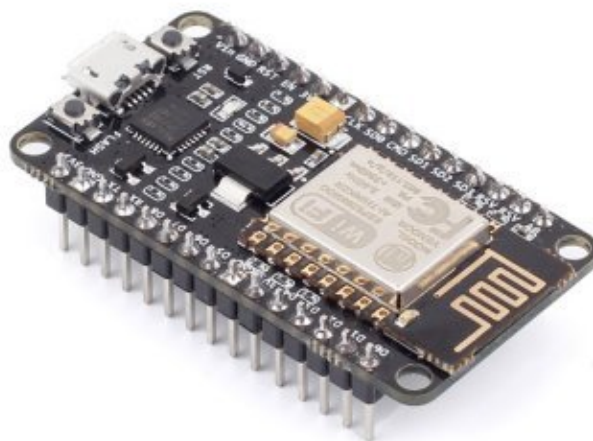


Figure 6.2: NodeMCU.

Part Number	Frequency Range	Spreading Factor	Bandwidth	Effective Bitrate	Est. Sensitivity
RFM95W	868/915 MHz	6 - 12	7.8 - 500 kHz	.018 - 37.5 kbps	-111 to -148 dBm
RFM97W	868/915 MHz	6 - 9	7.8 - 500 kHz	0.11 - 37.5 kbps	-111 to -139 dBm
RFM96W/RFM98W	433/470MHz	6- 12	7.8 - 500 kHz	.018 - 37.5 kbps	-111 to -148 dBm

Figure 6.3: Some of the different LoRa modules available[17].

of its fairly smaller size: 34.2mm long and 25.6mm wide as opposed to the 49mm long and 24.5mm wide NodeMCU.

On a side note, we could have minimized the price even further if we just chose to work with the ESP8266 chip by itself, but we felt the board was already cheap enough, thus working with the chip by itself was not worth the extra hassle.

6.1.2 Transceiver Module

This is the part in charge of taking the packets and modulate them using LoRa. We have a few different options for it, Figure 6.3.

Firstly, since we are in Europe, we need a module that supports the 868MHz frequency range, so we have to discard both RFM96W and RFM98W.

Secondly, the difference in price between the modules is almost neglectable, so we are looking for the one with the best performance. From Figure 6.3 we can see that the module RFM95W supports all five spreading factors and has the best estimated sensitivity, so it will be the one we choose.

Finally, now that we have decided that we will use the RFM95W module, we need to find a chip with it. In our case we have found the chip SX1276, by Semtech.

6.1.3 Antenna

An antenna is the last part we need to have a functional mote, whose functions consist of transmitting and receiving the signal over the 868MHz frequency range.

Since we need the chip SX1276, we have decided to use the transceiver module LoRa BEE, which is based on said chip and it comes with an 868MHz antenna, Figure 6.4.

With this final piece we now have all of the necessary hardware to build our mote. From this point, we only have to properly wire the pieces together, implement the MAC layer and we will have a working mote.



Figure 6.4: LoRa BEE by Dragino [18].

6.2 Gateway

In the State of the Art chapter, we decided to build the DIY multi-channel Raspberry Pi gateway and therefore we will need the following:

- Embedded Linux board.
- Concentrator.
- Antenna.
- Pigtail for the antenna.
- Board to concentrator interface.
- Enclosure. Since we plan on placing it outside.

6.2.1 Embedded Linux Board

There are several embedded Linux boards to choose from such as Raspberry Pi, Beagle Bone and Banana Pi among others. However, we are more familiar with the Raspberry Pi and it will be the embedded Linux board we will be using.

Now, there are several different Raspberry Pi models, and almost all of them should work except for the original versions [14]. In addition, we are not aiming for a low-cost gateway, so we can afford to use the latest versions.



Figure 6.5: Antenna and pigtail for iC880A concentrator [19].

In our case we will use the Raspberry Pi 3 model B, which is well over the minimum requirements but it will help improve the overall performance of the gateway.

6.2.2 Concentrator

As we have mentioned before in previous chapters, the concentrator is the piece that allows our gateway to be multi-channel.

Regarding the specifications, the concentrator needs to be LoRaWAN compatible and be able to handle packets with different spreading factors and data rates. We have found that the iC880A concentrator meets all of those requirements and so it will be the one we will use.

Lastly, this concentrator allows for female jumper wire as interface between it and the board. This means that we will use double female jumper wires to connect the concentrator and the Raspberry Pi.

6.2.3 Antenna

Similarly to the mote, an antenna is a crucial part of the gateway, in charge of transmitting and receiving the LoRa packets over the ISM band.

Luckily for us, the manufacturer that sells the concentrator also sells the antenna and pigtail for it. Therefore and mainly for convenience, we will be using those ones, Figure 6.5. However, any antenna that supports the adequate frequency range, 868MHz in our case, should work too.

6.2.4 Enclosure

Since we want the gateway to be outside for maximum coverage, we need to make sure it does not get damaged by the weather, birds, insects or dust. However, Granada is not a very rainy city, thus we do not need an extreme water proof enclosure. Nevertheless, it is important that we choose the adequate level of protection according to our zone, as it is a big factor in our gateway's useful life and performance. Therefore and taking all of those factors into account, we have chosen an IP54 enclosure:

1. The first number in IP54, 5, stands for the level of protection against small objects. In this case, level 5 means complete protection against contact and dust deposit.
2. The second number in IP54, 4, stands for the level of waterproofing. In this case, level 4 means protection against splashed water, which is enough for us.

Finally, we feel that this level of protection is enough for our zone. However, we should monitor the state of the gateway and make sure the enclosure is working as intended.

Chapter 7

Implementation

This chapter will describe the steps followed to set up the mote, the gateway and the application. Thus, it will include the process between having the different hardware elements by themselves and everything working together and fully operational.

Also, we have to set up the application before the mote, since we will need some parameters from the application to implement the mote.

7.1 Application

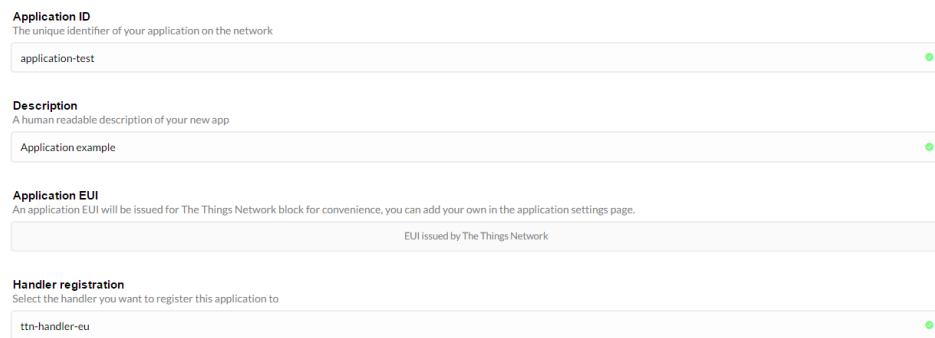
Here we will go through the process of creating our application, which will be linked to our mote in order to receive and decipher the packets. Since we are making use of TTN, we have to create the application through their website.

To start off, we need to register an account before we can create the application. Then, after logging in with said account, we navigate to the console of TTN: <https://console.thethingsnetwork.org/>, where we have the option to either register a gateway or create an application. Right now we want to create an application, so we click on "Applications". This will take us to the application manager, but we do not have any applications yet so we click on "add application".

When creating the application, we have to fill in some fields which will help describe and identify our application. An example configuration is shown in Figure 7.1.

At this point our application is ready, however we are not done yet because we have to link at least one device to it. In order to link a device we navigate to the "Devices" tab and click on "register device". Here we only have to fill in the "Device ID" field, as the rest of the fields will be generated automatically. Example configuration shown in Figure 7.2.

To wrap up, the last thing we have to do is enter the "Settings" tab in our device and change the "Activation Method" from OTAA to ABP, as it



Application ID
The unique identifier of your application on the network

application-test

Description
A human readable description of your new app

Application example

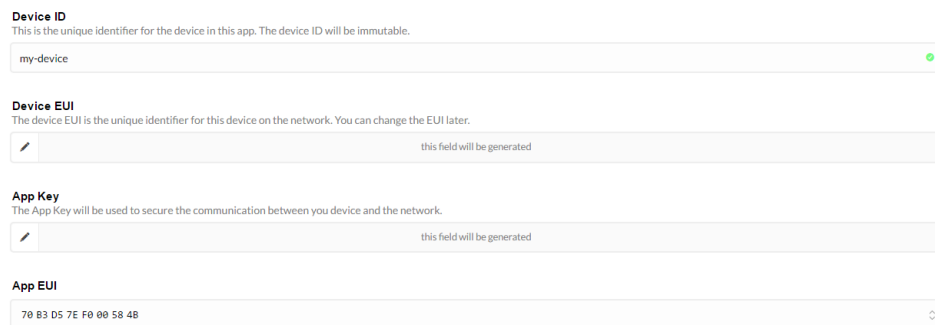
Application EUI
An application EUI will be issued for The Things Network block for convenience, you can add your own in the application settings page.

EUI issued by The Things Network

Handler registration
Select the handler you want to register this application to

ttn-handler-eu


Figure 7.1: Application configuration example.




Device ID
This is the unique identifier for the device in this app. The device ID will be immutable.

my-device

Device EUI
The device EUI is the unique identifier for this device on the network. You can change the EUI later.

 this field will be generated

App Key
The App Key will be used to secure the communication between you device and the network.

 this field will be generated

App EUI

70 B3 D5 7E F0 00 58 48

Figure 7.2: Device configuration example.

Pin	Function	ESP-8266 Pin
TX	TXD	TXD
RX	RXD	RXD
A0	Analog input, max 3.3V input	A0
D0	IO	GPIO16
D1	IO, SCL	GPIO5
D2	IO, SDA	GPIO4
D3	IO, 10k Pull-up	GPIO0
D4	IO, 10k Pull-up, BUILTIN_LED	GPIO2
D5	IO, SCK	GPIO14
D6	IO, MISO	GPIO12
D7	IO, MOSI	GPIO13
D8	IO, 10k Pull-down, SS	GPIO15
G	Ground	GND
5V	5V	-
3V3	3.3V	3.3V
RST	Reset	RST

Figure 7.3: Wemos D1 Mini pinout [16].

will be the activation method that we will use on the mote. After this last step, we have generated all the keys and parameters we will later need to set up the mote.

Finally, as a suggestion and only during testing, we should disable the "Frame Counter Checks" option, so that our application picks up all the packets after we reset the transmission.

And that is it, we are done. Our application is ready to receive packets from our mote through TTN.

7.2 Mote

In this section we will describe how to set up both the hardware and the software for the mote. The goal here is to end up with a fully functional transceiver capable of sending a customizable payload over the LoRa physical layer.

7.2.1 Wiring

Although we could start with the software first, we are going to begin building the mote by wiring the different pieces together.

First thing we have to do is identify the pins for both the Wemos and LoRa BEE, Figures 7.3 and 7.4 respectively.

Secondly, now that we know the pins layout, we will connect the ground and power pins, Table 7.1.

Then we will connect the serial peripheral interface bus, by connecting the MISO, MOSI, SCK and SS pins together, Table 7.2.

Finally, we will connect the I2C bus, Table 7.3.

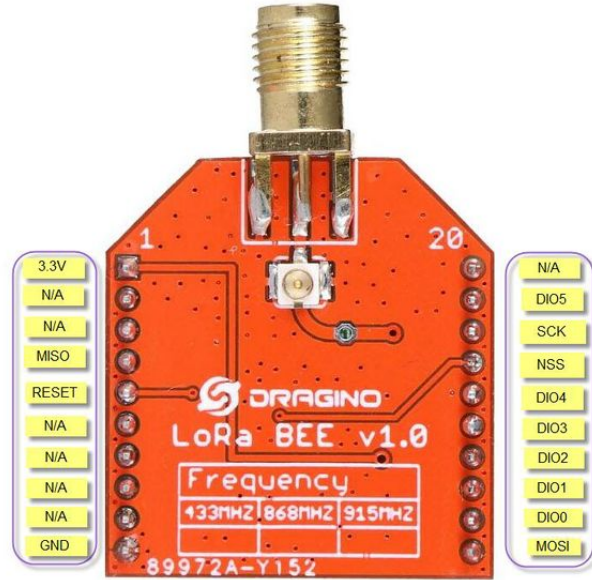


Figure 7.4: LoRa BEE pinout [18].

Function	Wemos pin	LoRa BEE pin
3.3V Power input	3V3	1
Ground	G	10

Table 7.1: Pin connections for ground and power input.

Function	Wemos pin	LoRa BEE pin
MISO	D6	4
MOSI	D7	11
SCK	D5	18
SS	D8	17

Table 7.2: Pin connections for SPI bus.

Function	Wemos pin	LoRa BEE pin
SCL	D1	12
SDA	D2	13

Table 7.3: Pin connections for I2C bus.

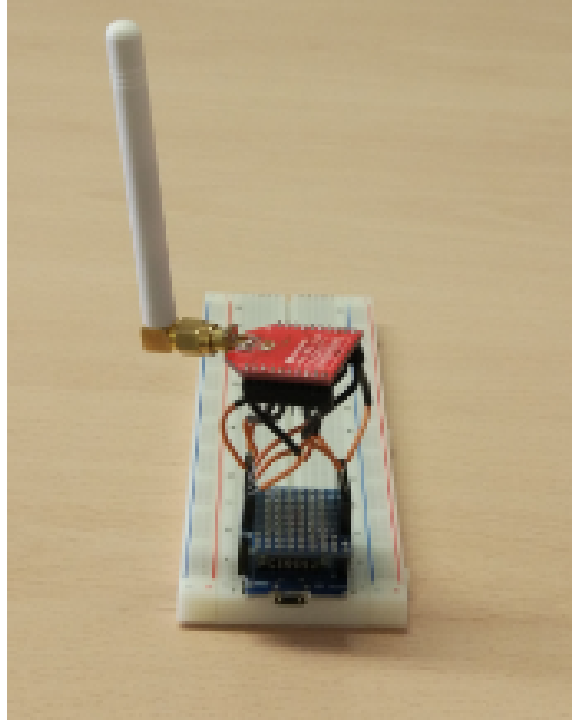


Figure 7.5: Mote after the wiring process.

Now, we should have everything properly connected and waiting for the software to be implemented. However, we encountered a problem during the wiring process. The distance between the LoRa BEE pins is different from the distance between the pins in the breadboard. Thus, we had to figure out a different way to connect the wires. As a result, we ended up welding the wires between the Wemos board and the LoRa BEE, Figure 7.5.

7.2.2 Software

First of all, we need to know what language we are going to use. The Wemos D1 Mini development board supports Arduino, NodeMCU and MicroPython. We are familiar with the Arduino language, thus it will be the one we will be using.

Secondly, it is not viable and out of the reach of this project to implement the LoRaWAN MAC layer from scratch, therefore we will use the Arduino-LMIC library [20] version 1.5, which is based on the LMIC library developed by IBM for C language.

Once we have the library downloaded and added to Arduino we can start our program but, again, we do not have to start from scratch as the library also includes some LoRaWAN demo programs that we can use as a starting point. To be specific, it comes with two LoRaWAN examples:

- "Hello World" program class A node using ABP.
- "Hello World" program class A node using OTAA.

Either of the examples can be used as a valid starting point, but ABP is slightly simpler to implement and the one we will use. Then, with our Arduino sketch open, we have to configure some parameters:

- Network session key, "NWSKEY".
- Application session key, "APPSKEY".
- Device address, "DEVADDR".
- Pin map, "lmic_pins".
- Transmission interval, "TX_INTERVAL". We should set this interval over 1 minute in order to comply with TTN's fair access policy. However, we are going to set it to 6 seconds during our testing process and then, revert it back to 1 minute once we are done testing.
- Payload, "mydata". This one is optional, we will leave the default "Hello, world!" because it is enough for our testing purposes. However, if we had a sensor we would have to adjust the payload so it contains the information gathered by the sensor.
- Spreading Factor, "DR_SF". An optional parameter as well, we will leave it as 7 but we can increase it up to 12 if we need longer range. Also, by leaving it as 7 it will decrease the airtime and thus, we can transmit packets a higher frequency. Additionally, note that forcing an specific spreading factor in the program is considered "hard-coding an spreading factor", and TTN strongly advises not to use motes with hard-coded SF11 or SF12.

Most of the parameters are located at the start of the sketch. The values we need to use for the network session key, the application session key and the device address were generated by the device we previously registered for the application. Therefore, we have to go back to the TTN console and copy those values into the Arduino sketch using the right hexadecimal format.

We can see an example configuration in Figure 7.6. Also, the entire Arduino sketch is available in Appendix A at the end of this document.

As for the pin map, Figure 7.7 shows the Arduino code that corresponds to the wiring configuration we did earlier. Also, when configuring the pin map, we have to take a few things into consideration:

- The names on the left correspond to pins on the transceiver, while the numbers refer to GPIO number on the Wemos side. For example, 15 means GPIO15 which corresponds to D8.

```

// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const PROGMEM u1_t NWKSKEY[16] = { 0xD4, 0x74, 0x3F, 0xB0, 0x0E, 0xD4, 0x02, 0x56, 0x88, 0xE9, 0xDD, 0x87, 0xE7, 0xC9, 0x44, 0xF7 };

// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the early prototype TTN
// network.
static const u1_t PROGMEM APPSKEY[16] = { 0xD9, 0x7D, 0x39, 0x84, 0xA2, 0x71, 0x0D, 0x67, 0xE3, 0xE0, 0x85, 0xC9, 0x08, 0x73, 0xB1, 0x65 };

// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x26011304 ; // <-- Change this address for every node!

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[] = "Hello, world!";

```

Figure 7.6: Example of Arduino parameters configuration.

```

// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = 15,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = LMIC_UNUSED_PIN,
    .dio = {4, 5, LMIC_UNUSED_PIN},
};

```

Figure 7.7: Arduino pin mapping.

- We do not need to specify the pins for the SPI bus, as they are always the same.
- Pins not used should be specified as "LMIC_UNUSED_PIN".
- Do not change the name of the struct, as it is a special name recognized by the library.

Finally, once we are done configuring all the parameters, we verify the sketch to make sure we have not made any mistakes, and then upload it to the Wemos board. Once this is done, we will now have both end points ready, the mote and the application, and at this point we are only missing the gateway for a fully operational LoRaWAN.

7.3 Gateway

7.3.1 Wiring

Wiring the Raspberry to the concentrator is very similar to the wiring we performed for the mote: we have to connect the 5V power input, ground and the SPI bus. Therefore, in this case we will list all the connections directly in Table 7.4.

Function	Concentrator pin	Raspberry Pi pin
5V Power input	21	2
Ground	22	6
Reset	13	22
SPI Clock	14	23
MISO	15	21
MOSI	16	19
NSS	17	23

Table 7.4: Pin connections between the Raspberry Pi and the concentrator.

And that is it for the wiring between the Raspberry Pi and the concentrator, Figure 7.8. The next step will be to set up the software.

7.3.2 Software

The first thing we have to do is install an operative system in the Raspberry Pi using the SD card. For this step we have chosen the Raspbian Jessie Lite operative System.

Following the installation of the operative system, we need to enter the following command:

```
1 $ sudo raspi-config
```

Here we can enable the SPI ("Interfacing options") and expand the file system ("Advanced options").

Now, we should make sure our system is up to date with the commands:

```
1 $ sudo apt-get update
2 $ sudo apt-get upgrade
```

Then we install "git":

```
1 $ sudo apt-get install git
```

At this point, we can now install the software for the gateway. Thankfully, TTN provides us with a script that installs everything for us [14]. We can make use of this script by typing these commands:

```
1 $ git clone -b spi https://github.com/ttn-zh/ic880a-gateway.git ~/ic880a-gateway
2 $ cd ~/ic880a-gateway
3 $ sudo ./install.sh spi
```

When running the script, we will be asked if we want to use remote configuration. Remote configuration is a handy option that will allow us



Figure 7.8: Gateway after the wiring process.

to change the settings of our gateway without actually having to connect our gateway, since it can be cumbersome once the gateway is properly set up. However, in our case we do not feel like we need this feature, so we will answer "no" and proceed to fill in the parameters manually. The positioning parameters we will be using are the following:

- Latitude: 37.19706.
- Longitude: -3.62453.
- Altitude: 750.

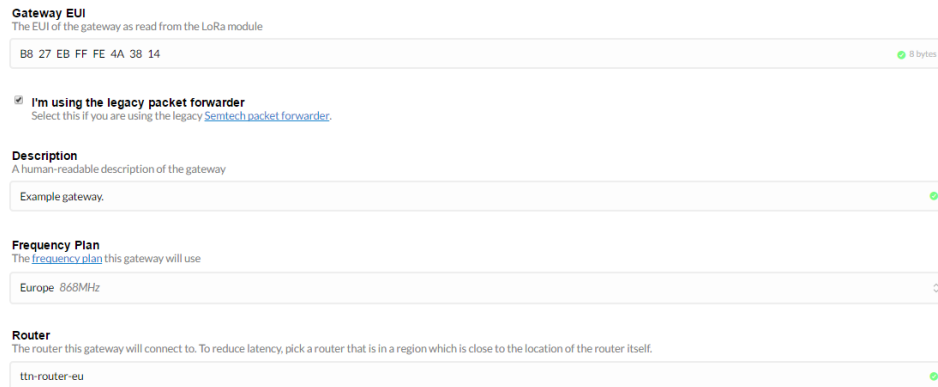
We obtained the latitude and longitude using "Google maps", while the altitude is a rough estimate based on the altitude of our city, Granada.

In addition, during the installation we will be prompted with the EUI of our gateway and we should write it down somewhere, as we will need it soon for the registration of the gateway. In our case our EUI is "B827EBFFFE4A3814".

At this point, our gateway is capable of receiving packets, but it cannot forward them to our application yet.

7.3.3 Registration

Next, we have to register our gateway on TTN before it is able to forward packets to TTN. For this, we have to go again to the TTN console, click



Gateway EUI
The EUI of the gateway as read from the LoRa module

B8 27 EB FF FE 4A 38 14 8 bytes

☒ **I'm using the legacy packet forwarder**
Select this if you are using the legacy [Semtech packet forwarder](#).

Description
A human-readable description of the gateway

Example gateway.

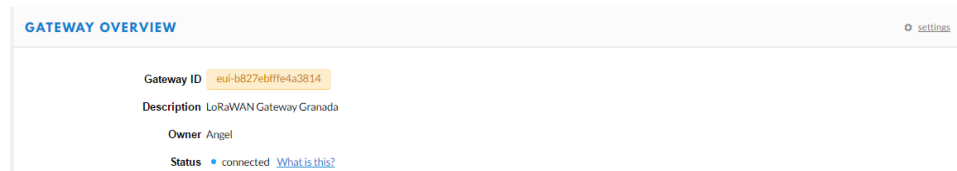
Frequency Plan
The [frequency plan](#) this gateway will use

Europe 868MHz

Router
The router this gateway will connect to. To reduce latency, pick a router that is in a region which is close to the location of the router itself.

ttn-router-eu

Figure 7.9: Example of gateway registration.



GATEWAY OVERVIEW settings

Gateway ID eui-b827ebffe4a3814

Description LoRaWAN Gateway Granada

Owner Angel

Status connected [What is this?](#)

Figure 7.10: Gateway status: connected.

on "Gateways" and then on "register gateway". Here we will have to fill in some fields, like our gateway's EUI and our frequency plan. An example using our parameters is shown in Figure 7.9.

Since the script we used to install the software for the gateway was based on the "Semtech packet forwarder", we have to check the box that says "I'm using the legacy packet forwarder".

As for the frequency plan, our gateway is located in Europe so we have to choose the European ISM band, which is the 868MHz band.

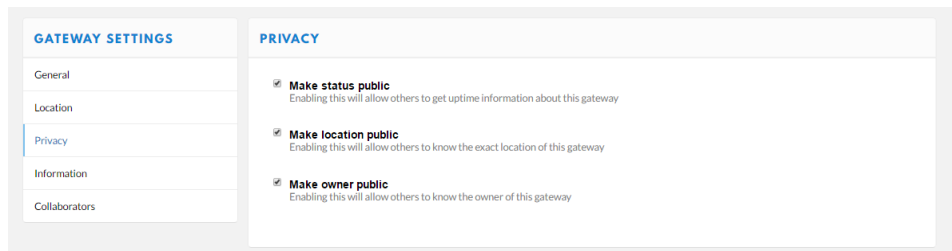
If everything has been set up correctly and our gateway is plugged in, we should now see the status of the gateway as "connected", Figure 7.10.

Lastly, we are going to set the privacy settings of the gateway as public, Figure 7.11, and add all of the information possible, Figure 7.12.

7.3.4 Enclosure

To wrap up, we are going to mount the gateway in the enclosure in order to keep it safe from insects, birds, rain, etc. But before doing so, we have to make two holes:

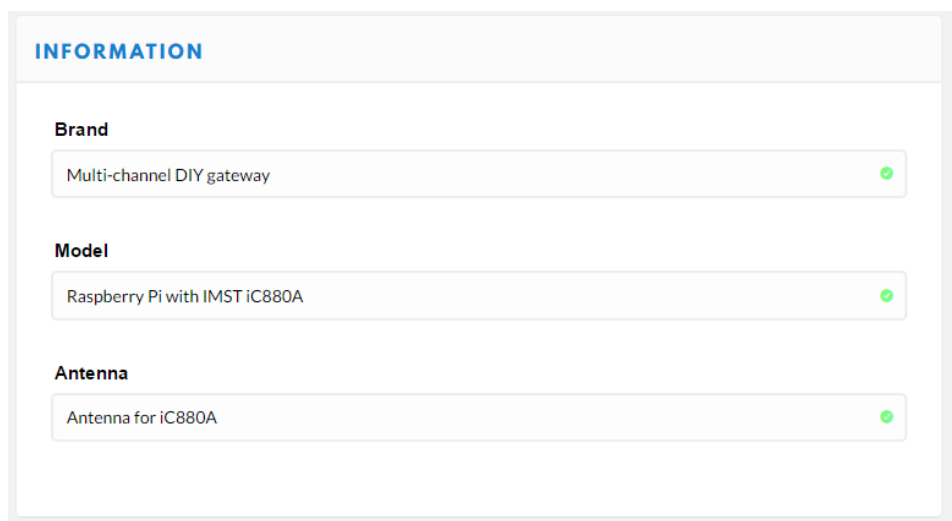
- One for the power supply. The gateway would not be able to do much without a power supply.



The screenshot shows a web interface for Gateway Settings. On the left is a sidebar with a 'GATEWAY SETTINGS' header and a list of tabs: General, Location, Privacy (which is selected), Information, and Collaborators. The main content area is titled 'PRIVACY' and contains three settings, each with a checked checkbox and a description:

- Make status public**
Enabling this will allow others to get uptime information about this gateway
- Make location public**
Enabling this will allow others to know the exact location of this gateway
- Make owner public**
Enabling this will allow others to know the owner of this gateway

Figure 7.11: Gateway privacy settings.



The screenshot shows a web interface for Gateway Information. It has a header 'INFORMATION' and three sections, each with a label and a text input field followed by a green checkmark icon:

- Brand**
Multi-channel DIY gateway
- Model**
Raspberry Pi with IMST iC880A
- Antenna**
Antenna for iC880A

Figure 7.12: Gateway information.



Figure 7.13: Gateway enclosed.

- Another one for the Ethernet cable. This one is optional, but in case we need to reconfigure our gateway for whatever reason we will not have to open the enclosure. Alternatively, we could set up the "wpa_supplicant" on the Raspberry Pi so we can access it via WiFi, however the enclosure might weaken the WiFi signal to the point where it barely reaches outside the enclosure.

As seen in Figure 7.8, the Ethernet cable barely fits in the enclosure and thus, cannot be turned in order to connect it to the Raspberry Pi. Due to this, we bought a right angle USB connector.

7.3.5 Location

Finally, we are going to place the gateway on high place so that it has better reach. For this purpose, we have placed the gateway on the terrace of the fifth floor of the school. Although we would like it to be even higher, the gateway will stay there during all of the testing process.

Chapter 8

Testing and Results

In this chapter, now that we have everything up and running, we will go through the results obtained by the different tests we have performed. These tests are the following:

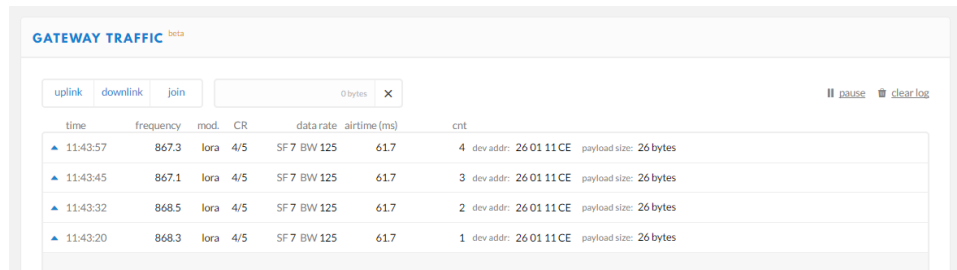
- Sending data to our application using the mote.
- Gateway coverage measurement, both indoors and outdoors.
- Channel hopping histogram.
- Spectrum analysis of an uplink transmission.
- Mote power consumption measurement.

8.1 Sending Data to the Application

This is the most important test. We are going to start transmitting with our mote and see if we receive the packets in the application. If this test fails it means something has gone wrong during the implementation and we would have to redo it again.


The test itself is really simple, first of all we have to make sure the gateway is connected and running. Once that is done, we plug in the mote and start transmitting. Then, if everything is set up properly, we should be able to see our packets using the TTN website.

First, we are going to check if the gateway is receiving the packets. In order to check the traffic of our gateway, we have to log on our TTN account, go to the console, navigate to our gateway and click on the "Traffic" tab. In Figure 8.1 we can see that the gateway is indeed receiving the packets. We can recognize the packets are from our mote by looking at the device address. Moreover, if we click on a packet we can see more detailed information, Figure 8.2. From the packet details we can obtain some parameters like the encrypted payload, RSSI, airtime, channel, etc.



time	frequency	mod.	CR	data rate	airtime (ms)	cnt
11:43:57	867.3	lora	4/5	SF 7 BW 125	61.7	4 dev addr: 26 01 11 CE payload size: 26 bytes
11:43:45	867.1	lora	4/5	SF 7 BW 125	61.7	3 dev addr: 26 01 11 CE payload size: 26 bytes
11:43:32	868.5	lora	4/5	SF 7 BW 125	61.7	2 dev addr: 26 01 11 CE payload size: 26 bytes
11:43:20	868.3	lora	4/5	SF 7 BW 125	61.7	1 dev addr: 26 01 11 CE payload size: 26 bytes

Figure 8.1: Packets received at the gateway.



11:45:11 868.5 lora 4/5 SF 7 BW 125 61.7 10 dev addr: 26 01 11 CE payload size: 26 bytes

Uplink

Dev Address

26 01 11 CE

Physical Payload

40 CE 11 01 26 00 0A 00 01 BE 3A E3 AD DF DC 93 B7 D9 71 3D 51 05 49 BE CB 73

Event Data

```

1 {
2   "gw_id": "eui-b827ebfffe4a3814",
3   "payload": "QW4RASaACgABvjnrjrd/ck7ZcT1RBUmwy3W=",
4   "f_cnt": 10,
5   "lorawan": {
6     "spreading_factor": 7,
7     "bandwidth": 125,
8     "air_time": 61696000
9   },
10  "coding_rate": "4/5",
11  "timestamp": "2017-06-07T09:45:11.028Z",
12  "rssi": -89,

```

Figure 8.2: Fragment of the packet details at the gateway.

APPLICATION DATA			
<input type="button" value="uplink"/> <input type="button" value="downlink"/> <input type="button" value="activation"/> <input type="button" value="ack"/> <input type="button" value="error"/>			
time	counter	port	
▲ 11:48:22	26	1	payload: 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
▲ 11:48:10	25	1	payload: 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
▲ 11:47:58	24	1	payload: 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
▲ 11:47:46	23	1	payload: 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
▲ 11:47:34	22	1	payload: 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21

Figure 8.3: Packets received at the application.

Now that we know the gateway is receiving the packets correctly, it is time to see if the application does too. This process is very similar to one we did for the gateway. We start by logging on our TTN account, then go to the console, navigate to the applications, select the device we are using and click on "Data".

It looks like everything is working as intended, as the packets are being received at the application, Figure 8.3. Similarly to the gateway, we can see the details of the packets by clicking on then, Figure 8.4. These details are very similar to the ones shown by the gateway, except for the payload which is decrypted.

Finally, to make sure that the payload is being decrypted correctly we are going to convert it to text, since the payload shown by the application is hexadecimal.

Hexadecimal string:

```
1 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
```

Converting it to text we obtain:

```
1 Hello, world!
```

So everything has been a success, the payload is correctly received and decrypted at the application.

8.2 Gateway Coverage

An important part of our gateway is its range, therefore the goal of this test is to measure how far away we can get from the gateway while it is still able to receive the packets with sufficient signal power.

However, these measurements vary greatly depending if we are measuring indoors or outdoors. Accordingly, we are going to perform the test in both situations.

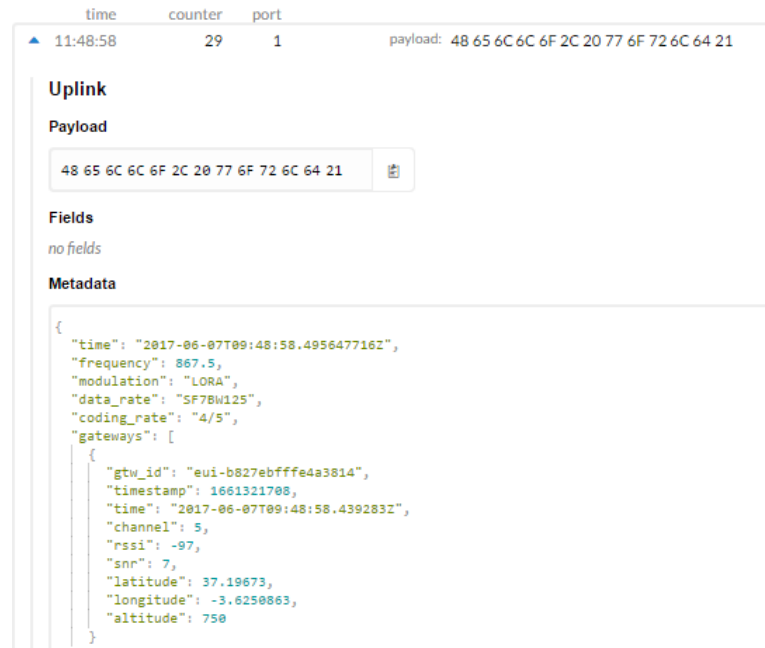


Figure 8.4: Fragment of the packet details at the application.

8.2.1 Indoors

The indoors test will consist on transmitting packets with the mote during 10 minutes on each floor of the school, except for floor 2 which will be explained on its own section. We will start on floor 5 and end on floor -1, for a total of seven floors. Moreover, we will store the RSSI level of each packet received during those 10 minutes. Lastly, these values will be used to make an RSSI histogram for each floor.

However, we cannot work directly with the data received on the application, unless we manually write down the values. Therefore, we use an MQTT client, Mosquitto version 3.1 in our case, to store the data so that we can work with it. By looking at the API provided by TTN, we use the following command to store the data:

```
1 $ mosquitto_sub -h eu.thethings.network -u mota01 -P ttn-account-v2.
    gtalAJgsBw4IEtL-jsZml_WF77dMLdnF-a5tYuXJU4Y -t "+/devices/+/up" >
    lora_mqtt_floorX.log)
```

Where:

- h is the host to connect.
- u is the username, in this case it corresponds to the Application ID.
- P is the Access Key of the application.

```

1 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":0,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":
2 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":1,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":
3 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":2,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":
4 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":3,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":
5 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":4,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":
6 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":5,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":
7 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":6,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":
8 {"app_id":"mota01","dev_id":"0000000000000001","hardware_serial":"00844245A921SDE2","port":1,"counter":7,"payload_raw":"SGVabG9eIHdvcmaKIQ==","metadata":

```

Figure 8.5: Fragment of the log file for floor 0.

-t indicates the topic we want to subscribe to.

> indicates the file where the data will be stored.

Once we have the data stored in a log file, we need to, somehow, store the RSSI values in an array in Matlab. We also have to keep in mind that we have stored much more data aside from the RSSI, such as SNR, airtime, channel, etc. Example of log file in Figure 8.5.

The procedure to get the RSSI array in Matlab involves opening the log file with excel in order to separate the different fields in different columns and, once that is done, we can use Matlab to read a specific column of the excel file to store it as an array. The code for the Matlab scripts can be found in Appendix B at the end of this document.

Lastly, remember the gateway is located on the fifth floor, as we mentioned in the implementation chapter.

Floor 5

We start our test on floor 5, Figure 8.6. We can see that the most common RSSI values are -57 and -60dBm with 11 and 9 appearances respectively.

These RSSI values are not that good considering we are almost next to the gateway. This might be due to the location of the gateway, which is placed right in front of the metallic grid that covers the entrance to the terrace. This grid might be deteriorating the received signal and thus, lowering the RSSI level.

Floor 4

On floor 4, Figure 8.7, the most common RSSI values are -51 and -52dBm with 14 and 13 appearances respectively, which are better values than on floor 5.

These measurements were taken slightly closer to the gateway (on the horizontal plane) than the ones taken on floor 5. That small deviation plus the metallic grid not being directly in the way is what might be causing the RSSI levels to be better here than on floor 5.

Floor 3

On floor 3, Figure 8.8, the most common RSSI values are -81 and -79dBm with 12 and 7 appearances respectively.

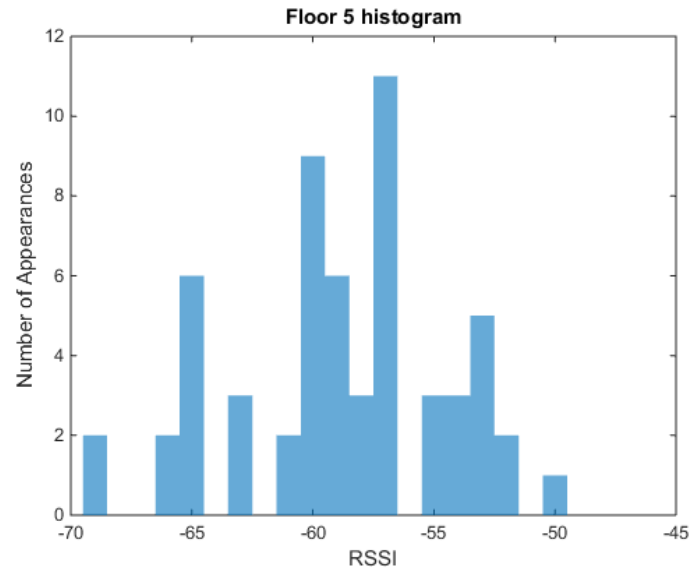


Figure 8.6: Floor 5 RSSI histogram.

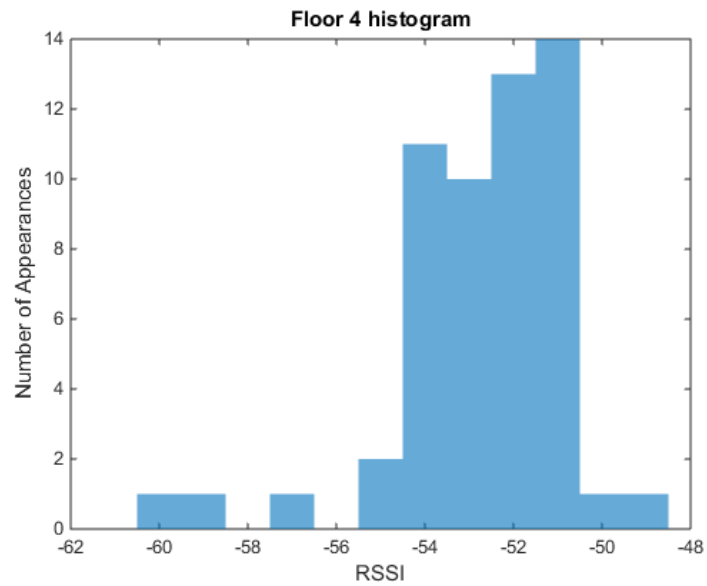


Figure 8.7: Floor 4 RSSI histogram.

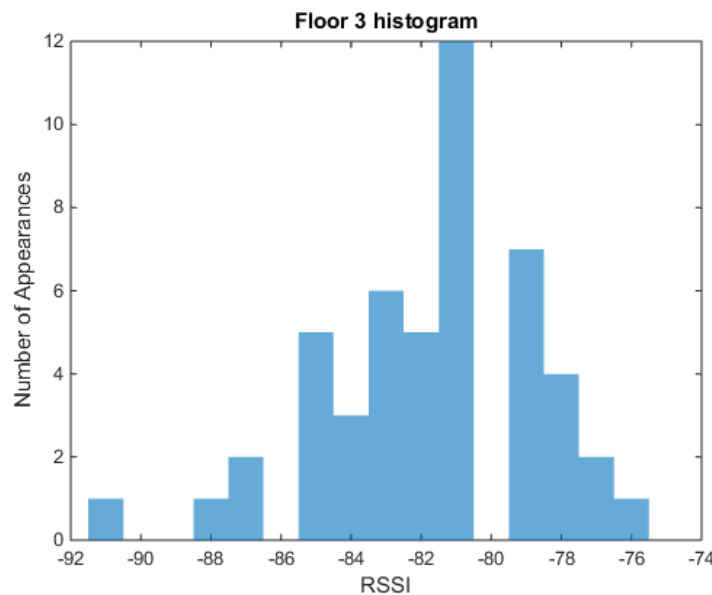


Figure 8.8: Floor 3 RSSI histogram.

It is on this floor where we started seeing a noticeable degradation of the RSSI levels.

Floor 2

On floor 2, Figure 8.9, the most common RSSI values are -90, -91, -97 and -99dBm with 69, 58, 58 and 58 appearances respectively.

We have a larger sample size on this floor because one of my supervisors has his office on this floor, so he was able to leave the mote sending packets for about 2 hours. Additionally, the test for this floor was performed on a different day than the rest of the floors.

We expected the histogram to be more similar to a Gaussian distribution with this large sample size. However, that is not what happened as there are several RSSI levels evenly distributed. We think this is caused by the channel hopping that occurs when using LoRaWAN, leading to different RSSI levels for each channel.

Floor 1

On floor 1, Figure 8.10, the most common RSSI values are -81 and -87dBm with 11 and 10 appearances respectively.

Although there are now four floors between us and the gateway, the packets are still being received with acceptable RSSI levels. In fact, these packets have a slightly better RSSI than the ones received from floor 2. We

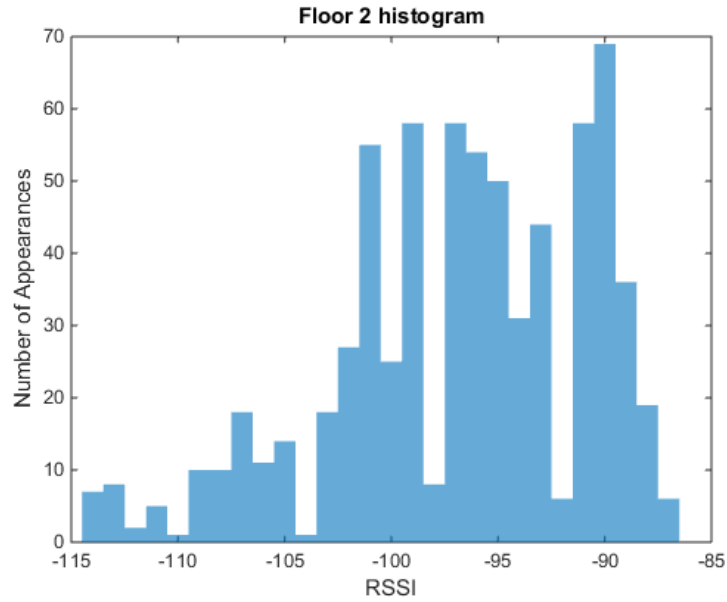


Figure 8.9: Floor 2 RSSI histogram.

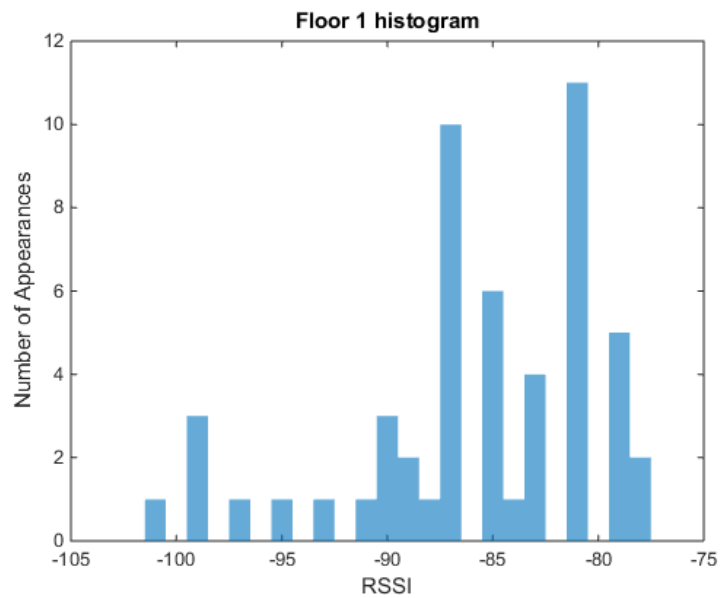


Figure 8.10: Floor 1 RSSI histogram.

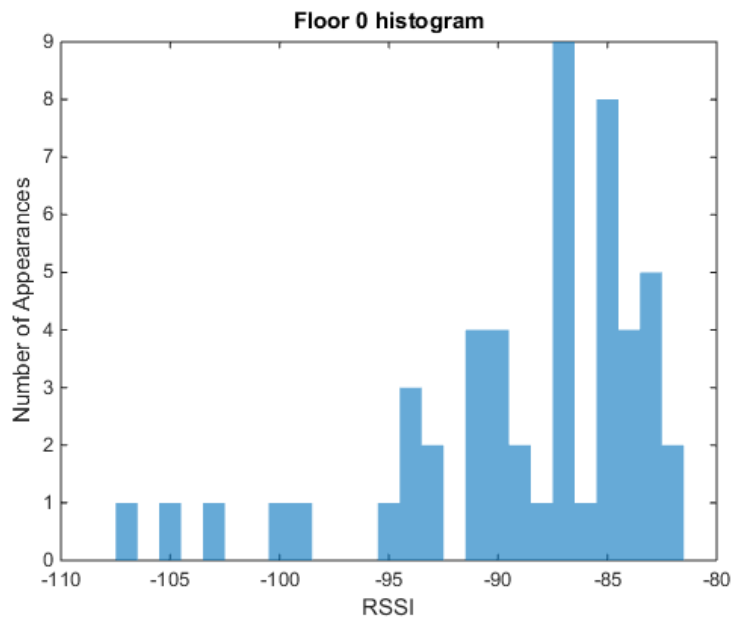


Figure 8.11: Floor 0 RSSI histogram.

think this might be caused by the small sample size and the fact that the measurements on floor 2 were taken on a different day.

Floor 0

On floor 0, Figure 8.11, the most common RSSI values are -87 and -84dBm with 9 and 8 appearances respectively.

Despite being one floor below, we have about the same RSSI levels as on floor 4. This might be due to sending the testing packets from a different point on the floor, as I did not want to obstruct the stairs to the library on floor 1. Therefore, I ended up taking the measurements on a different point that was closer to the gateway on the horizontal plane.

Floor -1

On floor -1, Figure 8.12, the most common RSSI values are -101 and -97dBm with 9 and 7 appearances respectively.

Finally, on the last floor, there are 6 floors between us and the gateway and the packets are still being received just fine. We also have to keep in mind that there is concrete between each floor.

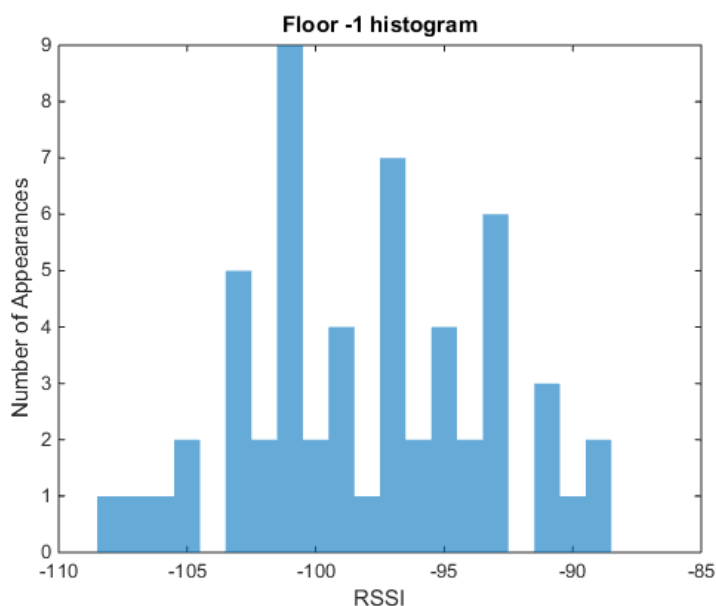


Figure 8.12: Floor -1 RSSI histogram.

8.2.2 Outdoors

As the second part of this coverage test, we are going to perform it outside. For this, we could use the same procedure we used for the indoors part, however we found an application called "TTN Mapper", which will yield us more elegant results.

For this part, all we need is a phone with and GPS receiver and Android operative system (version 4.03 or higher), as well as a LoRaWAN transmitter. Of course, the gateway has to be running too. To be more specific, we used:

- Sony Xperia phone with Android version 4.3.
- Our low-cost LoRaWAN mote.
- TTN Mapper application version 27 and build date 01/04/2017.

The procedure for this part is fairly simple:

1. We open the application and link our mote to it, either manually or by logging on our TTN account and selecting the device.
2. We plug in the mote and start transmitting, while keeping the phone close to it.
3. On the application, we click on "Start mapping".



Figure 8.13: Results of the outdoor coverage testing on the TTN Mapper website [3].

Now, we can start moving around with the mote and the TTN Mapper will display the position of our mote, by using the GPS of our phone, and the RSSI level obtained from that position. Also, during this test we used SF7 for several reason:

- It is the recommended spreading factor by the TTN Mapper application.
- It is the worst case scenario, as SF7 has the lowest range.
- It allows us to transmit packets mote often due to the duty cycle limitations, thus improving the resolution of the coverage map.

Once we are done mapping, we can see the results on Figures 8.13 and 8.14, obtained from the TTN Mapper website [3]. In these figures, the point with the TTN symbol represents the position of our gateway.

The furthest point measured is about 550m away from the gateway and was still received within the greater than -100dBm range.

8.3 Channel Hopping Histogram

Taking advantage of the log files we stored for the coverage measurements, we used the data in them to plot a histogram of the different channels used, Figure 8.15. This histogram will be based on the data collected from floor 2 because of the larger sample size. Furthermore, the script used can be found in Appendix B at the end of this document.

As a result from the histogram, we can clearly see how the channels are practically evenly distributed and there is no preferred channel.



Figure 8.14: Zoomed in results of the outdoor coverage testing on the TTN Mapper website [3].

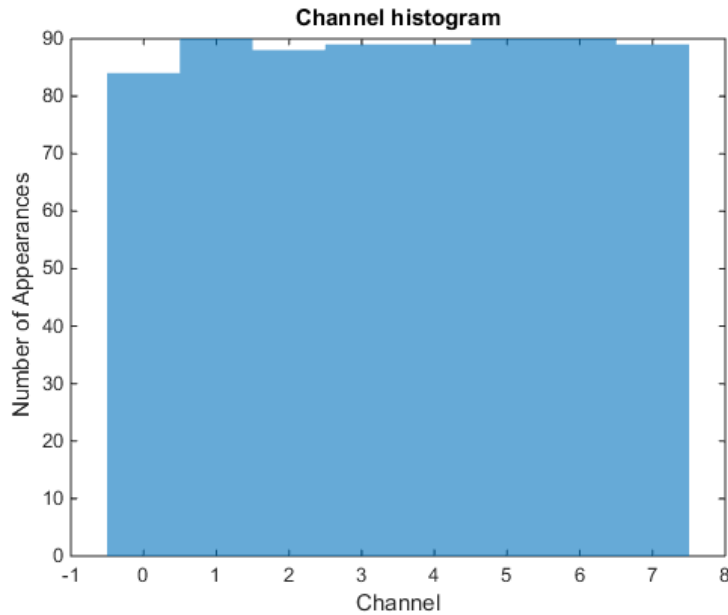


Figure 8.15: Histogram of the different channels used.

8.4 Spectrum Analysis

This test consists on using Matlab and an SDR to capture the spectrum of the ISM band for a few seconds. The Matlab script can be found in Appendix C at the end of this document.

Firstly, we will capture the spectrum when no packets are being transmitted, Figure 8.16. In this case, the small peak on the center of the image is caused by the hardware of the SDR.

Secondly, we will capture the spectrum once our mote has started transmitting packets, Figure 8.17. Also, note that for this test we are using SF7.

As a result of this test, we can clearly see the wideband nature of LoRa, as the signal is spread across the available bandwidth.

8.5 Power Consumption

One of the main advantages of LoRaWAN is its low power consumption. In fact, it is one of the reasons to use LoRaWAN over WiFi or cellular networks, as to minimize battery usage. Knowing this, we want to test how much power our mote consumes when transmitting packets every six seconds.

To perform this test, we will use a little device, called USB tester, that is connected between the power source and the mote. When running, this

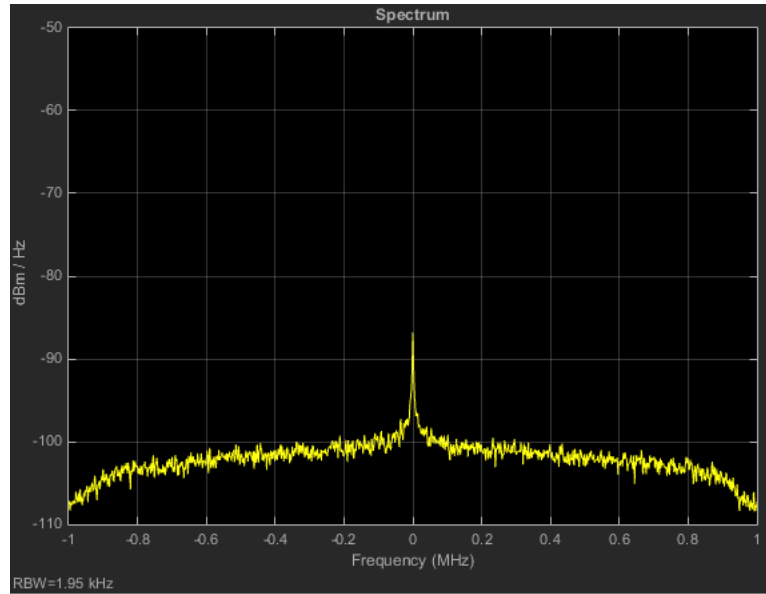


Figure 8.16: Spectrum centered on 868MHz when nothing is transmitting.

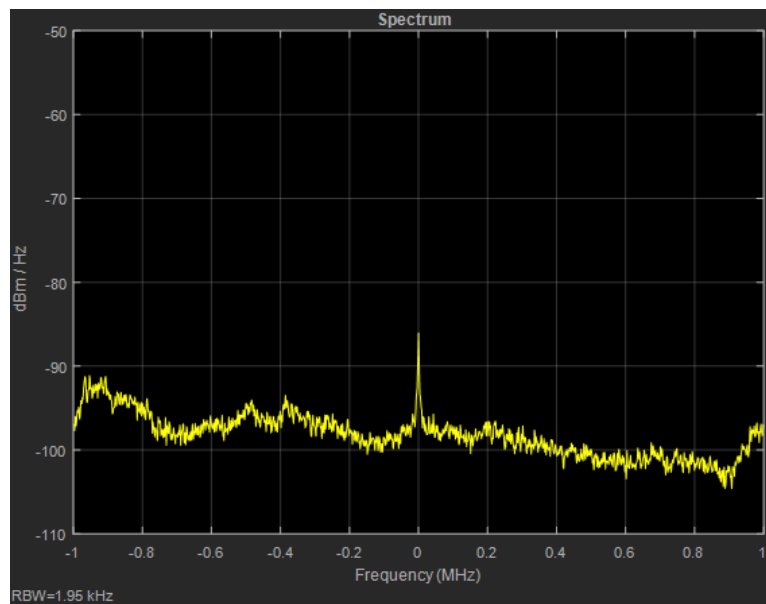


Figure 8.17: Spectrum centered on 868MHz when packets are being transmitted.



Figure 8.18: USB Tester after measuring for 32 minutes during worst case scenario.

devices shows the following parameters:

- Time expired. Resolution of one minute.
- Voltage, shown in Volts. Resolution of 0.01V.
- Current, shown in Amperes. Resolution of 0.01A.
- Accumulated electric charge, shown as milliamperes hour. Resolution of 1mAh.

Furthermore, this test will be split into two different tests:

1. First test will measure the mote during the worst case scenario. This is, when the mote is transmitting every few seconds, limited only by the duty cycle and using SF7.
2. Second test will repeat these measurements but for a more realistic scenario. This is, when the mote transmits a packet every five minutes in addition to disabling WiFi on the ESP8266, since we are not making use of it.

8.5.1 Worst Case Scenario

In this case, the mote is transmitting packets every 12 seconds approximately and the WiFi module, which we are not using, is still enabled. Then, after measuring for 32 minutes and transmitting 160 packets, we obtained 29mAh at 5.09V, Figure 8.18.

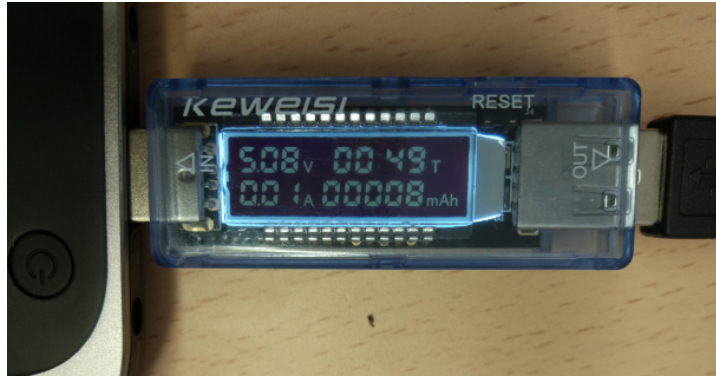


Figure 8.19: USB Tester after measuring for one hour during realistic scenario.

$$Power_{WorstCase} = 29 * 10^{-3} * 5.09 * \frac{60}{32} = \mathbf{0.2768W} \quad (8.1)$$

From this result we can draw the conclusion that this mode is not recommended for a battery dependent mote, as it is slightly power intensive. For reference, a 10,000mAh battery would only last for a little over a week.

8.5.2 Realistic Scenario

During this test, the mote will be transmitting packets every five minutes and the WiFi module will be disabled.

In order to disable the WiFi module we have to slightly modify our Arduino program. Firstly, we include library for the ESP8266 Module at the start of the sketch.

```
1 #include <ESP8266WiFi.h>
```

Then, we add in the "setup" these two commands:

```
1 WiFi.forceSleepBegin(0); //This function turns on modem sleep mode (
   turns off RF but not CPU)
2
3 delay(1); //For some reason the modem won't go to sleep unless you do
   a delay(non-zero-number) -- no delay, no sleep and delay(0), no
   sleep
```

Now that the WiFi module is turned off, we proceed to measure the power consumed for one hour, Figure 8.19. For some reason the timer shows 49 minutes, but the actual measuring time was one hour.

$$Power_{Realistic} = 8 * 10^{-3} * 5.08 * \frac{60}{60} = \mathbf{0.0406W} \quad (8.2)$$

From these results, it is clear that in a realistic scenario and with some optimizations, the power consumption is extremely low. Using the same reference we used for the worst case scenario, a 10,000mAh battery would last for about 52 days. Also, it is worth noting that we could reduce the power consumption even further by doing the following:

- Working with the ESP12 chip by itself without the Wemos board.
- By using the Deep Sleep mode available on the ESP8266 chip, which consumes about 60 μ A [21], and activating it in between transmissions.

Chapter 9

Conclusions

In this final chapter, we will draw our conclusions based on what we have learned throughout the whole process of carrying out this project.

9.1 LoRa

LoRa works really well despite the density of buildings surrounding the gateway. Accordingly, we would expect LoRa to work even better on open environments, so it is better suited for agricultural purposes.

Furthermore, it surpasses WiFi range without a doubt. For reference, and average WiFi antenna might reach about 30 meters.

9.2 LoRaWAN and TTN

LoRaWAN is still an up and coming technology with a lot of potential, however it relies heavily on the community as its only way of deploying LoRaWAN gateways.

So far this method has been working out, but if the interest of the community dies out in the next few years and not enough gateways remain operative, then LoRaWAN might fall out of relevance.

Moreover, TTN is also a great factor in LoRaWAN's success. TTN allows the users to make use of their network structure, so that only a gateway and an end-device are needed in order to have a fully operational LoRaWAN.

9.3 The Gateway

Our gateway has been a great success so far. It is able to receive packets from a large distance, from different motes, with different spreading factors and data rates.

Yet, we have encountered a problem we did not think of when we started: heat. Granada is a city that can get extremely hot during summer and, on

top of that, the enclosure of our gateway is black. Right now the gateway is not directly exposed to the sun most of the day and it reaches about 70°C (measurement obtained from the Raspberry Pi). Therefore, if we want to place it somewhere else, we have to be careful with the sun exposure.

All in all and despite that temperature issue, the gateway is working extremely well while also being fairly simple to build.

9.4 Low Cost Mote

The mote has proven to work properly while meeting the requirement of being cheap and affordable. In contrast, it has been slightly cumbersome to build since the LoRa BEE has a distance between its pins that is not compatible with a breadboard, so we had to come up with an inventive way of connecting the LoRa BEE and the Wemos board. We could also have made the process more convenient if we increased the price slightly and bought a shell that connects the Wemos and the LoRa BEE.

On one hand, this is an affordable mote that, if properly optimized, can last for a long time without having to replace the batteries.

On the other hand, the fact that we have to include the LoRaWAN MAC layer ourselves means that the average user might need some technical support, thus being less practical.

Overall, the mote is an excellent choice if the user is comfortable including the library that implements the MAC layer. Otherwise, it might be worth it to pay about 10 extra euros and buy a full LoRaWAN stack chip.

Bibliography

- [1] Bill Morelli, "Internet Connected Devices: Evolving from the "Internet of Things" to the "Internet of Everything"," 2013, accessed on 20/06/2017. [Online]. Available: https://www.ihs.com/pdf/IHS-IOT-Evolution_161384110915583632.pdf
- [2] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O.Hersent, "LoRaWAN Specification," Jan. 2015, accessed on 20/06/2017. [Online]. Available: <https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf>
- [3] JP Meijers, "TTN Mapper," accessed on 20/06/2017. [Online]. Available: <http://ttnmapper.org/>
- [4] Semtech Corporation, "LoRa Modulation Basics," May 2015, accessed on 20/06/2017. [Online]. Available: <http://www.semtech.com/images/datasheet/an1200.22.pdf>
- [5] Mostafa Hassan Dahshan, "Spread Spectrum."
- [6] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things," *Sensors*, vol. 16, no. 9, p. 1466, Sep. 2016, accessed on 20/06/2017. [Online]. Available: <http://www.mdpi.com/1424-8220/16/9/1466>
- [7] LoRa Alliance Technical committee, "LoRaWAN Regional Parameters," Jul. 2016.
- [8] LoRa Alliance, "LoRa Technology," accessed on 20/06/2017. [Online]. Available: <https://www.lora-alliance.org/What-Is-LoRa/Technology>
- [9] Gemalto, Actility, and Semtech, "LoRaWAN Security. Full end-to-end encryption for IoT application providers." Feb. 2017, accessed on 20/06/2017.
- [10] The Things Network, "The Things Gateway," accessed on 20/06/2017. [Online]. Available: <https://shop.thethingsnetwork.com/index.php/product/the-things-gateway/>

- [11] Wienke Giezeman, “Fair Access Policy,” Jan. 2016, accessed on 20/06/2017. [Online]. Available: <https://speakerdeck.com/wienke/the-things-network-2016-update#28>
- [12] T. Telkamp, “Single Channel LoRaWAN Gateway,” May 2017, accessed on 20/06/2017. [Online]. Available: https://github.com/tftelkamp/single_chan_pkt_fwd
- [13] Telkamp, “Single Channel Gateway Part 1,” accessed on 20/06/2017. [Online]. Available: <http://www.thethingsnetwork.org/forum/t/single-channel-gateway-part-1/798>
- [14] The Things Network, “The Things Network: iC880a-based gateway,” 2015, accessed on 20/06/2017. [Online]. Available: <https://github.com/ttn-zh/ic880a-gateway>
- [15] Spanish Government, “Defensa de la Competencia,” Jul. 2007, accessed on 20/06/2017. [Online]. Available: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2007-12946
- [16] Wemos Electronics, “Wemos D1 Mini,” accessed on 20/06/2017. [Online]. Available: https://wiki.wemos.cc/products:d1:d1_mini
- [17] HopeRF Electronic, “RFM Datasheet,” accessed on 20/06/2017. [Online]. Available: http://www.hoperf.com/upload/rf/RFM95_96_97_98W.pdf
- [18] Dragino, “Lora BEE,” accessed on 20/06/2017. [Online]. Available: http://wiki.dragino.com/index.php?title=Lora_BEE
- [19] IMST, “Antenna and Pigtail for iC880a-SPI,” accessed on 20/06/2017. [Online]. Available: <http://webshop.imst.de/pigtail-for-ic880a-spi-and-ic880a-usb.html>
- [20] M. Kooijman, “Arduino-LMIC,” Aug. 2016, accessed on 20/06/2017. [Online]. Available: <https://github.com/matthijskooijman/arduino-lmic>
- [21] Espressif Systems IOT Team, “ESP8266ex Datasheet,” 2015, accessed on 20/06/2017. [Online]. Available: <http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf>

Appendices

Appendix A

Arduino Program for the Mote

This is the Arduino program we have used to implement LoRaWAN MAC layer on the mote. This program is based on an example provided by the Arduino-LMIC library, version 1.5.

```
1  /*****
2   * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
3   *
4   * Permission is hereby granted, free of charge, to anyone
5   * obtaining a copy of this document and accompanying files,
6   * to do whatever they want with them without any restriction,
7   * including, but not limited to, copying, modification and
8   * redistribution.
9   * NO WARRANTY OF ANY KIND IS PROVIDED.
10  *
11  * This example uses ABP (Activation-by-personalisation), where a
12  *   DevAddr and
13  * Session keys are preconfigured (unlike OTAA, where a DevEUI and
14  * application key is configured, while the DevAddr and session keys
15  * are
16  * assigned/generated in the over-the-air-activation procedure).
17  *
18  * Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1% in
19  * g1, 0.1% in g2), but not the TTN fair usage policy.
20  *****/
21
22 #include <lmic.h>
23 #include <hal/hal.h>
24 #include <SPI.h>
25
26 // LoRaWAN NwkSKey, network session key
27 // This is the default Semtech key, which is used by the early
28 // prototype TTN
29 // network.
30 static const PROGMEM u1_t NWKSKEY[16] = { 0x55, 0xE5, 0xF4, 0xFE, 0xDF
31     , 0xB4, 0xD7, 0x43, 0xFD, 0xD6, 0x65, 0xB3, 0xD9, 0xAE, 0xD1, 0xAD
32     };
33
34 // LoRaWAN AppSKey, application session key
35 // This is the default Semtech key, which is used by the early
```

```

    prototype TTN
30 // network.
31 static const u1_t PROGMEM APPSKEY[16] = { 0xD9, 0x44, 0x61, 0x6B, 0x0D
    , 0xC1, 0x42, 0x8A, 0x97, 0x55, 0xF9, 0x7F, 0x80, 0xFE, 0x24, 0x70
    };
32
33 // LoRaWAN end-device address (DevAddr)
34 static const u4_t DEVADDR = 0x260111CE;
35
36 // These callbacks are only used in over-the-air activation, so they
    are
37 // left empty here (we cannot leave them out completely unless
38 // DISABLE_JOIN is set in config.h, otherwise the linker will complain
    ).
39 void os_getArtEui (u1_t* buf) { }
40 void os_getDevEui (u1_t* buf) { }
41 void os_getDevKey (u1_t* buf) { }
42
43 static uint8_t mydata[] = "Hello, world!";
44 static osjob_t sendjob;
45
46 // Schedule TX every this many seconds (might become longer due to
    duty
47 // cycle limitations).
48 const unsigned TX_INTERVAL = 6;
49
50 // Pin mapping
51 const lmic_pinmap lmic_pins = {
52     .nss = 15,
53     .rxtx = LMIC_UNUSED_PIN,
54     .rst = LMIC_UNUSED_PIN,
55     .dio = {4, 5, LMIC_UNUSED_PIN},
56 };
57
58 void onEvent (ev_t ev) {
59     Serial.print(os_getTime());
60     Serial.print(": ");
61     switch(ev) {
62         case EV_SCAN_TIMEOUT:
63             Serial.println(F("EV_SCAN_TIMEOUT"));
64             break;
65         case EV_BEACON_FOUND:
66             Serial.println(F("EV_BEACON_FOUND"));
67             break;
68         case EV_BEACON_MISSED:
69             Serial.println(F("EV_BEACON_MISSED"));
70             break;
71         case EV_BEACON_TRACKED:
72             Serial.println(F("EV_BEACON_TRACKED"));
73             break;
74         case EV_JOINING:
75             Serial.println(F("EV_JOINING"));
76             break;
77         case EV_JOINED:
78             Serial.println(F("EV_JOINED"));
79             break;
80         case EV_RFU1:
81             Serial.println(F("EV_RFU1"));
82             break;
83         case EV_JOIN_FAILED:
84             Serial.println(F("EV_JOIN_FAILED"));
85             break;

```

```

86     case EV_REJOIN_FAILED:
87         Serial.println(F("EV_REJOIN_FAILED"));
88         break;
89     case EV_TXCOMPLETE:
90         Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
          windows)"));
91         if (LMIC.txrxFlags & TXRX_ACK)
92             Serial.println(F("Received ack"));
93         if (LMIC.dataLen) {
94             Serial.println(F("Received "));
95             Serial.println(LMIC.dataLen);
96             Serial.println(F(" bytes of payload"));
97         }
98         // Schedule next transmission
99         os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(
          TX_INTERVAL), do_send);
100         break;
101     case EV_LOST_TSYNC:
102         Serial.println(F("EV_LOST_TSYNC"));
103         break;
104     case EV_RESET:
105         Serial.println(F("EV_RESET"));
106         break;
107     case EV_RXCOMPLETE:
108         // data received in ping slot
109         Serial.println(F("EV_RXCOMPLETE"));
110         break;
111     case EV_LINK_DEAD:
112         Serial.println(F("EV_LINK_DEAD"));
113         break;
114     case EV_LINK_ALIVE:
115         Serial.println(F("EV_LINK_ALIVE"));
116         break;
117     default:
118         Serial.println(F("Unknown event"));
119         break;
120 }
121 }
122
123 void do_send(osjob_t* j){
124     // Check if there is not a current TX/RX job running
125     if (LMIC.opmode & OP_TXRXPEND) {
126         Serial.println(F("OP_TXRXPEND, not sending"));
127     } else {
128         // Prepare upstream data transmission at the next possible
          time.
129         LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
130         Serial.println(F("Packet queued"));
131     }
132     // Next TX is scheduled after TX_COMPLETE event.
133 }
134
135 void setup() {
136     Serial.begin(115200);
137     Serial.println(F("Starting"));
138
139     #ifdef VCC_ENABLE
140     // For Pinoccio Scout boards
141     pinMode(VCC_ENABLE, OUTPUT);
142     digitalWrite(VCC_ENABLE, HIGH);
143     delay(1000);
144     #endif

```

```

145
146 // LMIC init
147 os_init();
148 // Reset the MAC state. Session and pending data transfers will be
    discarded.
149 LMIC_reset();
150
151 // Set static session parameters. Instead of dynamically
    establishing a session
152 // by joining the network, precomputed session parameters are be
    provided.
153 #ifdef PROGMEM
154 // On AVR, these values are stored in flash and only copied to RAM
155 // once. Copy them to a temporary buffer here, LMIC_setSession
    will
156 // copy them into a buffer of its own again.
157 uint8_t appskey[sizeof(APPSKEY)];
158 uint8_t nwkskey[sizeof(NWKSKEY)];
159 memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
160 memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
161 LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
162 #else
163 // If not running an AVR with PROGMEM, just use the arrays
    directly
164 LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
165 #endif
166
167 #if defined(CFG_eu868)
168 // Set up the channels used by the Things Network, which
    corresponds
169 // to the defaults of most gateways. Without this, only three base
170 // channels from the LoRaWAN specification are used, which
    certainly
171 // works, so it is good for debugging, but can overload those
172 // frequencies, so be sure to configure the full frequency range
    of
173 // your network here (unless your network autoconfigures them).
174 // Setting up channels should happen after LMIC_setSession, as
    that
175 // configures the minimal channel set.
176 // NA-US channels 0-71 are configured automatically
177 LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
178 LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B),
    BAND_CENTI); // g-band
179 LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
180 LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
181 LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
182 LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
183 LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
184 LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
185 LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK),
    BAND_MILLI); // g2-band
186 // TTN defines an additional channel at 869.525Mhz using SF9 for
    class B
187 // devices' ping slots. LMIC does not have an easy way to define

```

```

188         set this
189         // frequency and support for class B is spotty and untested, so
190         this
189         // frequency is not configured here.
190         #elif defined(CFG_us915)
191         // NA-US channels 0-71 are configured automatically
192         // but only one group of 8 should (a subband) should be active
193         // TTN recommends the second sub band, 1 in a zero based count.
194         // https://github.com/TheThingsNetwork/gateway-conf/blob/master/US
195         -global_conf.json
195         LMIC_selectSubBand(1);
196         #endif
197
198         // Disable link check validation
199         LMIC_setLinkCheckMode(0);
200
201         // TTN uses SF9 for its RX2 window.
202         LMIC.dn2Dr = DR_SF9;
203
204         // Set data rate and transmit power for uplink (note: txpow seems
205         to be ignored by the library)
205         LMIC_setDrTxpow(DR_SF7,14);
206
207         // Start job
208         do_send(&sendjob);
209     }
210
211 void loop() {
212     os_runloop_once();
213 }
    
```


Appendix B

Matlab Histogram Scripts

These are the scripts used to make the histograms shown on the "Testing and Results" chapter. Note that for these scripts to work the excel file must be located in the same folder as the script.

B.1 Floor 5 Histogram

```
1 RSSI = xlsread('Planta5.xlsx','AH1:AH70');  
2 %RSSI levels are located in the AH column.  
3  
4 histogram(RSSI)  
5 title('Floor 5 histogram')  
6 xlabel('RSSI')  
7 ylabel('Number of Appearances')
```

B.2 Floor 4 Histogram

```
1 RSSI = xlsread('Planta4.xlsx','AH1:AH70');  
2 %RSSI levels are located in the AH column.  
3  
4 histogram(RSSI)  
5 title('Floor 4 histogram')  
6 xlabel('RSSI')  
7 ylabel('Number of Appearances')
```

B.3 Floor 3 Histogram

```
1 RSSI = xlsread('Planta3.xlsx','AH1:AH70');  
2 %RSSI levels are located in the AH column.  
3  
4 histogram(RSSI)
```

```
5 title('Floor 3 histogram')
6 xlabel('RSSI')
7 ylabel('Number of Appearances')
```

B.4 Floor 2 Histogram

```
1 RSSI = xlsread('Planta2.xlsx','AH1:AH712');
2 %RSSI levels are located in the AH column.
3 %We have up to 712 RSSI values for this floor.
4
5 histogram(RSSI)
6 title('Floor 2 histogram')
7 xlabel('RSSI')
8 ylabel('Number of Appearances')
```

B.5 Floor 1 Histogram

```
1 RSSI = xlsread('Planta1.xlsx','AH1:AH70');
2 %RSSI levels are located in the AH column.
3
4 histogram(RSSI)
5 title('Floor 1 histogram')
6 xlabel('RSSI')
7 ylabel('Number of Appearances')
```

B.6 Floor 0 Histogram

```
1 RSSI = xlsread('Planta0.xlsx','AH1:AH70');
2 %RSSI levels are located in the AH column.
3
4 histogram(RSSI)
5 title('Floor 0 histogram')
6 xlabel('RSSI')
7 ylabel('Number of Appearances')
```

B.7 Floor -1 Histogram

```
1 RSSI = xlsread('PlantaSotano.xlsx','AH1:AH70');
2 %RSSI levels are located in the AH column.
3
4 histogram(RSSI)
5 title('Floor -1 histogram')
6 xlabel('RSSI')
7 ylabel('Number of Appearances')
```

B.8 Channel Hopping Histogram

```
1 Channels = xlsread('Planta2.xlsx','AF1:AF712')';  
2 %Channels are located in the AF column.  
3  
4 histogram(Channels)  
5 title('Channel histogram')  
6 xlabel('Channel')  
7 ylabel('Number of Appearances')
```


Appendix C

Matlab Functions Used for the Spectrum Analysis

These are the Matlab functions used to perform the Spectrum Analysis test in the "Testing and Results" chapter.

C.1 Function That Captures and Stores the Spectrum

This first function will capture the spectrum during a specified time, given by the sampling rate and the amount of frames we want to capture, and it will store the results on a file.

```
1 % Signal reception from a USRP (tested with NI 2901, i.e. B210)
2 % Syntax: [data,len]=receiveDataFromUSRP(<centerFrequency>, <
    samplingRate>, <noSamples>[, <bSpectrumAnalyzer>][, <gain>])
3 % Example: [data,len]=receiveDataFromUSRP(1.85e9, 200e3, 20e6)
4 function[data,len]=receiveDataFromUSRP(centerFrequency, samplingRate,
    noSamples, varargin)
5     debugPeriod = 100; % Show a message every 'debugPeriod' frames
6
7     if nargin < 3
8         error('receiveDataFromUSRP:TooFewInputs', ...
9             'requires at least 3 inputs: centerFrequency, samplingRate
10                , noSamples');
11     end
12
13     % Optional arguments: bSpectrumAnalyzer, gain, ...
14     numvarargs = length(varargin);
15     if numvarargs > 2
16         error('receiveDataFromUSRP:TooManyInputs', ...
17             'requires at most 2 optional inputs');
18     else
19         % set defaults for optional inputs
20         optargs = {0, 30}; % optargs = {defVal1, defVal2, ...};
21         [bSpectrumAnalyzer, gain] = optargs{:}; % [param1, param2,
22             ...] = optargs{:};
```

```

22     if (numvarargs == 1)
23         bSpectrumAnalyzer = varargin{1};
24     elseif (numvarargs == 2)
25         bSpectrumAnalyzer = varargin{1};
26         gain = varargin{2};
27     end
28 end
29
30 disp(['centerFrequency: ' num2str(centerFrequency) ...
31      ', samplingRate: ' num2str(samplingRate) ...
32      ', noSamples: ' num2str(noSamples) ...
33      ', bSpectrumAnalyzer: ' num2str(bSpectrumAnalyzer) ...
34      ', gain: ' num2str(gain)]);
35
36 %connectedRadios = findsdru;
37 %if (connectedRadios(1).Status == 'Success')
38
39     masterClockRate = 20e6; % Default value
40     samplesPerFrame = 10000;%4096; % Default value
41
42     %if (connectedRadios(1).Platform == 'B210')
43     %     masterClockRate = 20e6; % For B210
44     %end
45     decimationFactor = masterClockRate/samplingRate;
46
47 %     radio=comm.SDRuReceiver(...
48 %         'Platform', connectedRadios(1).Platform, ...
49 %         'SerialNum', connectedRadios(1).SerialNum, ...
50 %         'CenterFrequency', centerFrequency, ...
51 %         'MasterClockRate', masterClockRate, ...
52 %         'DecimationFactor', decimationFactor, ...
53 %         'SamplesPerFrame', samplesPerFrame, ...
54 %         'OutputDataType', 'double', ...
55 %         'Gain', gain);
56
57     radio=comm.SDRuReceiver(...
58         'Platform', 'B210', ...
59         'SerialNum', '30B56E1', ...
60         'CenterFrequency', centerFrequency, ...
61         'MasterClockRate', masterClockRate, ...
62         'DecimationFactor', decimationFactor, ...
63         'SamplesPerFrame', samplesPerFrame, ...
64         'OutputDataType', 'double', ...
65         'Gain', gain);
66
67 rxLog = dsp.SignalSink;
68 if (bSpectrumAnalyzer)
69     hSpectrumAnalyzer = dsp.SpectrumAnalyzer(...
70         'Name', 'Spectrum',...
71         'Title', ['Spectrum centered
72                   at ' num2str(floor(centerFrequency/1e5)/10) ' MHz'
73                   ], ...
74         'SpectrumType', 'Power density',...
75         'FrequencySpan', 'Full', ...
76         'SampleRate', samplingRate, ...
77         'YLimits', [-110,-50],...
78         'SpectralAverages', 50, ...
79         'FrequencySpan', 'Start and stop
80         frequencies', ...
81         'StartFrequency', -samplingRate/2, ...
82         'StopFrequency', samplingRate/2,...
83         'Position', figposition([50 30

```

```

30 40]]);
81     end
82
83     totalFrames = ceil(noSamples / samplesPerFrame);
84     for counter = 1:totalFrames
85         [rxSig, len] = step(radio);
86         if (len > 0)
87             rxLog(rxSig);
88             if (bSpectrumAnalyzer)
89                 % Display received frequency spectrum.
90                 hSpectrumAnalyzer(rxSig);
91             end
92         end
93
94         if(mod(counter,debugPeriod) == 0)
95             disp(['captured ', num2str(counter) ' frames (out of ',
96                 num2str(totalFrames) ')...']);
97         end
98     end
99
100     data = rxLog.Buffer;
101     len = length(data);
102
103     % Release all System objects
104     release(radio);
105     clear radio;

```

C.2 Function That Displays the Spectrum

The second function takes the output of the first function as input and displays the spectrum.

```

1 % Spectrum from data saved from a USPR (tested with NI 2901, i.e. B210
  )
2 % Syntax: spectrumFromData(<samplingRate>)
3 % Example: spectrumFromData('lora868.mat', 2e6)
4 function spectrumFromData(filename, samplingRate)
5     debugPeriod = 100; % Show a message every 'debugPeriod' frames
6
7     load(filename);
8
9     samplesPerFrame = 4096;
10    datalength = length(data);
11    totalFrames = floor(datalength / samplesPerFrame);
12
13    hSpectrumAnalyzer = dsp.SpectrumAnalyzer(...
14        'Name', 'Spectrum',...
15        'Title', 'Spectrum',...
16        'SpectrumType', 'Power density',
17        ...,
18        'FrequencySpan', 'Full',...
19        'SampleRate', samplingRate,
20        ...,
21        'YLimits', [-110,-50],...
22        'SpectralAverages', 50, ...
23        'FrequencySpan', 'Start and stop
    frequencies', ...

```

```
22         'StartFrequency',          -samplingRate/2,
23         ...
24         'StopFrequency',          samplingRate
25         /2,...
26         'Position',              figposition([50
27             30 30 40]));
28
29     for counter = 1:totalFrames
30         rxSig = data(1+(counter-1)*samplesPerFrame : counter*
31             samplesPerFrame);
32         % Display received frequency spectrum.
33         hSpectrumAnalyzer(rxSig);
34
35         if(mod(counter,20) == 0)
36             keydown = waitforbuttonpress;
37
38             if(mod(counter,debugPeriod) == 0)
39                 disp(['shown ' num2str(counter) ' frames...']);
40             end
41         end
42     end
43 end
```