# TTK4550 - Use of LoRaWAN for IoT communication, Including Investigation of Connectivity Properties.

Tobias Ulfsnes Rasmussen
759533

December 2018

**NTNU – Trondheim**
Norwegian University of
Science and Technology

# Task Description

**NTNU**
**Norwegian University of**
**Science and Technology**

**Faculty of Information Technology**
**and Electrical Engineering**
**Department of Engineering Cybernetics**

## Specialization Project Report

| | |
|---|---|
| **Candidate:** | **Tobias Rasmussen** |
| **Course:** | **TTK4550 Engineering Cybernetics, Specialization Project** |
| **Thesis title (Norwegian)** | **X** |
| **Thesis title (English):** | **Use of LoRaWan for IoT communication and investigation of the connectivity properties** |

-

**Thesis desctiption:** Communication is an important issue when using IoT devices. Us of existing communication infrastructures (e.g. mobile data network) ease the communication and by that the setup of an IoT network.

Although the LoRaWAN is not that common as an existing communication infrastructure, it is very interesting as an extension to the public mobile data network, as the LoRaWAN gateways are relative costless and should be easy to maintain and connect to the public mobile data network.

In this specialization project we want to investigate the use of LoRaWAN for IoT-communication and the connectivity for the LoRa network.

**The tasks will be:**

1. Conduct a survey concerning the LoRaWAN system (communication, devices and components) and possible connections to other communication systems.
2. Suggest a design of a simple LoRa End-Device included needed SW and make an implementation of the proposed solution.
3. Investigate the connectivity property of LoRaWan.

| | |
|---|---|
| **Start date:** | August 18th, 2018 |
| **Due date:** | December 18th, 2018 |
| **Thesis performed at**: | Department of Engineering Cybernetics |
| **Supervisor:** | Professor Geir Mathisen |

# Summary

All over the world embedded-electronics are being modernized through interconnection and internet-access. This phenomenon is called IoT, today there are a lot of options in terms of how to do this. Wireless protocols have been around for a long time, however utilizing these in embedded-electronics put restrictions in terms of especially power. Thus a whole new generation of low-power wireless protocols surfaces. One of these is called LoRaWAN, a fairly young protocol finalized in 2015. As of today, the protocol are in the process of implementation here in Trondheim.

This project aims to investigate how this protocol works in an embedded system. So an embedded system utilizing LoRaWAN as a protocol has been designed, implemented and tested. The hardware of which is covered by another project: TTK8. Furthermore, an investigation into the connectivity properties has been conducted.

# Preface

This report is the final product of a specialization-project at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology in Trondheim. The supervisor of this project has been Geir Mathisen which has provided guidance and advice throughout the entire project. However, Wireless Trondheim has also offered assistance, providing hardware for development and provided details about the situation for IoT in Trondheim today.

# Abbreviations

| | | |
|---|---|---|
| LoRa | = | Abbreviated from Long Range, modulation-standard. |
| LoRWAN | = | A Wide Area Network MAC-layer protocol utilizing LoRa-modulation. |
| IoT | = | Internet of Things. |
| PHY-Layer | = | Physical layer, the hardware-layer of a protocol. |
| MAC | = | Medium Access Control. |
| SDK | = | Software development kit, a literal toolbox for development. |
| API | = | Application programming interface. |
| MQTT | = | Message Queuing Telemetry Transport, a publish-subscribe architecture. |
| GUI | = | Graphical User Interface |
| End-Device | = | A device which utilizes LoRa to communicate externally. |
| Back-End | = | The application which utilizes the gathered information. |
| Uplink | = | An transmission from an end-device to a server/back-end. |
| Downlink | = | A transmission from server or back-end to a end-device. |
| DR | = | Data-rate. |
| GW | = | Gateway. |
| ack | = | acknowledged. |
| nack | = | Not acknowledged. |
| OTAA | = | Over the air activation, a routine a end-device runs to join a LoRa Network. |
| AppEUI | = | Application identifier. |
| DevAddr | = | End-device address, also known as device network identifier. |
| FSK | = | Frequency Shift Keying modulation technique. |
| ABP | = | Activation by Personalization. |
| OTAA | = | Over the air activation. |
| UML | = | Unified Modeling Language. |
| SysML | = | System Modeling Language. |
| PCB | = | Printable Circuit Board. |

# Table of Contents

# List of Tables

# List of Figures

Chapter 1

# Introduction

## 1.1 Background

As of 2018 the terms IoT and smart grids are becoming more central and relevant by the day. The progression in IT has propelled the society to rethink how we perceive the world of electronics. Controlling electrical devices have traditionally been controlled either by a hands-on approach(buttons and dials) or since the 1950s by remote control(restricted locally). When thinking of information it has almost always either been hard-wired or (since the 1990s) connected to some kind of local network which again may be connected to the internet. The terms Local and Network are quite common, but in later years the terms IoT and WAN have made an entrance into the world of IT and electronics.
One of the most well-developed protocols which facilitates these is LoRa-WAN. The world of IoT doesn't only revolutionize the way we think about controlling devices, but also how and where we can collect information. These are necessary features in the implementation of new technologies like smart-grids, where real-time information is required to realize the concept.
All this makes LoRa-WAN a very relevant tool which deserves a closer look, which may uncover new scopes/context's where this protocol can be used.

## 1.2 Motivation

LoRa-WAN is a wireless wide area network protocol which is well developed, and partly implemented in several cities. Since this protocol is wireless, it opens up for applications beyond concepts like smart-grids. An intriguing idea is Remote-coverage.
To elaborate. There are some places geographically where an internet-connection is unobtainable with the current network-coverage(both ground and in air), such locations may also have no infrastructure to provide power. These places may contain interesting assets such as important climate variations, seismic-activity etc. All of which contain information which demands little to no bandwidth to share. What if a LoRa-Gateway was placed in a position which accommodates an internet connection and covers the area in interest? This facilitates for remote location coverage, such that information from an otherwise unreachable area may be reachable.
A more concrete example is an alarm system on a collection of cabins. These cabins are located in a valley which shields from conventional internet-connections such as 4G/LTE. In addition to the location, they have no infrastructures providing power. However, vantagepoints surrounding the valley may provide infrastructure such as power and internet connection. A LoRa-gateway is installed on one of these vantagepoints. This enables us to create a battery-driven alarm system on these utilizing LoRa, this system may detect measure moisture, smoke, and intruders. Then notify the owner when an anomaly is detected via the internet. This is one of many possible applications of LoRa-WAN, and more specifically in terms of remote coverage.
This means that extensive tests have to be done to check how the LoRa-protocol copes with varying terrain and distance. This is one of many reasons to put LoRa under the scope.

## 1.3 Document Structure

This document is divided into 9 chapters. Chapter 2 encompasses a literature review and the discoveries made within that review which may have an impact on this project. Chapter 3 focuses on the basic theory needed to understand the report, also a survey of the LoRaWAN architecture. Chapter 4 provides a specification and a solution to the given task. The implementation of which is covered by Chapter 5. The results and the corresponding discussion is provided in Chapter 6 and Chapter 7. Before the last chapter, there is a short conclusion of the project. Before the report finishes of proposing what may be taken further into a master-thesis.

In addition to the nine chapters, there are two appendixes: Appendix A and Appendix B. Appendix A are composed of "borrowed" material. Whereas Appendix B is composed of additional material and illustrations provided by the author.

Chapter 2

# Literature Study

This chapter highlights some of the other written literature written about LoRa and highlights details that may be significant for our task. The following sections highlight those that were deemed important, and relevant for this task.

## 2.1 Data Rates

As it turns out LoRa-WAN supports different data rates. To be specific it provides a bandwidth between $0.3\,\mathrm{kbit/s}$ and $50\,\mathrm{kbit/s}$. Which in terms of embedded are considered plenty. This opens up for applications that may use a bit of bandwidth, or maybe sends messages quite often. LoRa is not intended for such applications however it may be possible(1).

## 2.2 Channels vs Range

Early in the specification for LoRa, the paper notes that there is a trade-off between different channels/data-rates and range. To quote the specification: "The selection of the data rate is a trade-off between communication range and message duration."-(1) Which should mean that a lower data-rate serves better distance and maybe: more obstacles. This is worth noting in the possible case of developing indoor applications.

## 2.3 Data Rates and Power Consumption

It is worth noting that power consumption when broadcasting messages through LoRa, is dependant on the data-rates. As it turns out, the higher the data rate, the lower the power consumption. This makes sense since a transmission on a low bandwidth leads to a longer air-time, thus consuming more power.

Chapter 3

# Theory

The scope of this chapter is to lay down the theory necessary to understand the basic concepts of LoRa. As of today LoRa-WAN consist of the actual LoRa-protocol, which in turn are connected to a network. Meaning network-providers already exists and openly available. The Things Network. In addition to the basic theory, the architecture has been modeled in SysML, to illustrate how LoRa-WAN actually work. However, all of this will be described in detail in the following sections.

## 3.1 LoRaWAN

Lora is coined from the two words Long Range, and WAN stands for Wide Area Network. Also often called LoRa, these are two different things. LoRa is the name physical-implementation and modulation of the actual signal. As LoRaWAN encompasses LoRa as the physical layer and all the way up to the MAC-layer.

### 3.1.1 LoRaWAN Classes

The MAC-Layer have 3 different types of end-device classes, respectively class A, class B, and class C. These make up the three different options for end-device communication. To easier understand how the classes are related to the MAC-layer, fig. 9.2 gives a visual illustration of this.

#### Class A - Baseline

This class is the baseline communication and allows end-devices to communicate bi-directional. However, to decrease power consumption it is somewhat restricted. Each end-device uplink is followed by two short downlink windows. This is the only time the end-device is able to receive a transmission from the back-end.
This class is meant for devices which only sporadic/limited downlink transmissions from the backend. It is important to notice that an end-device of this class, will not be reachable from the back-end unless an uplink is acquired from the end-device.
After sending an uplink the devices have two receive windows: RX1 and RX2. These windows are static and determined by the network. For TTN the receive windows are 1 s each. Basically TX+1 s and TX+2 s. If nothing is received during those two windows, the end-device continues normal operation.

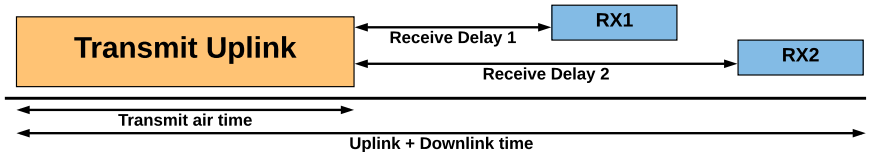



**Figure 3.1:** Simple illustration of the receive windows.

#### Class B - Beacon

Class B devices will have the same functionality as the Class A, with one important difference. This class enables a scheduled downlink window, in addition to the two windows following an uplink. Which means that end-devices utilizing this class is reachable at specific scheduled times.

#### Class C - Continuous

This class has an always open downlink, meaning it is always listening for downlink transmission. This class is more power-hungry than the two previously mentioned classes. However, it provides functionality for a different type of end-device application.

### 3.1.2 Physical Layer

The physical layer is closed and is acquired by the Semtech Corporation(2). So there is little to no information since this is considered proprietary for Semtech. This is important to know when dealing with LoRa, as it put some restriction in terms of freedom of use and development.

### Bandwith

There are some variations of which bandwidth are utilized. This is mostly due to different regulations in different locations around the world. The most common of these bandwidths and their geographical locations are provided in the following table:

| Bandwidth | Regions |
|---|---|
| 902-928 MHz | United States |
| 863-870 MHz and 433.05 - 434.79 MHz | Europe |
| 779 - 787 MHz and 470-510 MHz | China |

**Table 3.1:** Typical bandwidths according to location.

Notice that there are specifications for several other different countries, all of which are found in the "LoRaWAN Regional Parameters" (3).

### Data-Rate

Naturally, the data-rate will depend on the bandwidth and spreading factor. When talking about transmitting data there are several different data-rates available, depending on the channel which the data is transmitted on. The channels have a bandwidth of $125\,\mathrm{kHz}$, $250\,\mathrm{kHz}$ and $500\,\mathrm{kHz}$, which can provide between $0.3\,\mathrm{kbit/s}$ to $50\,\mathrm{kbit/s}$. This depends on what kind of Class the device supports. For all classes, there are 6 different levels of data-rates. To use the least amount of power, it is recommended to utilize the highest data-rate. However reducing the data-rate result in better range, so there is a trade-off between power and range. So the data-rate will, according to the LoRa specification(1), have a direct impact on the feasible range of the device. The following table is taken straight out of the LoRa documentation, more precisely the Regional Parameters document(3).

| Data-Rate(DR) | Configuration | Physical bit rate |
|---|---|---|
| 0 | SF12 / 125 kHz | 250 bit/s |
| 1 | SF11 / 125 kHz | 440 bit/s |
| 2 | SF10 / 125 kHz | 980 bit/s |
| 3 | SF9 / 125 kHz | 1760 bit/s |
| 4 | SF8 / 125 kHz | 3125 bit/s |
| 5 | SF7 / 125 kHz | 5470 bit/s |
| 6 | SF7 / 250 kHz | 11 000 bit/s |
| 7 | FSK | 50 000 bit/s |

**Table 3.2:** Possible data-rates for LoRa End-devices, as stated in regional parameters(3).

It is worth mentioning that any LoRa End-device must be able to utilize DR0-DR5, which means that any LoRa End-device and transceivers comply with these two data-rates.

### 3.1.3 Security

There is no need to go into detail about how security works, or how it is implemented in LoRa. However, it is worth mentioning that the protocol is end-to-end encrypted with AES 128 encryption.

### 3.1.4 Messages

In short, there are two types of messages: Uplink and Downlink. Each message contains either a payload or a join response/request. The payload size is dependant on which data-rate the device is using. For the datarates 0-6 the payload may be 0-255 B, however when utilizing data-rate 7(FSK) the payload is limited to 0-64 B(See table 3.2 and the RN2483 Command Reference(4)).

### 3.1.5 Join Pocedures

An end-device may join a network in 2 ways: Activation by Personalization(ABP) or over the air activation(OTAA).
ABP locks the end-device to a specific network and is a part of this network from the moment it is commissioned. This requires that the end-device have the following information: DevAddr, AppSKey, network session keys. This locks the end-device to a specific network. When it is commissioned, its immediately considered activated.

Over the air activation(OTAA) on the other hand, then you need the DevEUI, AppKey, and AppEUI to join. Compared to ABP the OTAA has different ways to describe the devices. In OTAA a device is considered generic if is not commissioned if it is then it is obviously described as commissioned but not activated. And when it joins, it is considered activated, and as an active part of the network just like an ABP-Activated device. This is illustrated in fig. 9.1 in Appendix A.

## 3.2 The Things Network

The Things Network is an open global network for IoT, and as of 2018, LoRa only. As mentioned it is an open network, which means that anybody anywhere can contribute to expanding the network. They are planning to expand their network not only by LoRa but also other protocols that are appropriate for use in IoT. The Things Network describe themselves like this:
"The Things Network is about enabling low power Devices to use long-range Gateways to connect to an open-source, decentralized Network to exchange data with Applications." - The Things Network(5).

### 3.2.1 TTN Console

The TTN-console is a tool for developers interested in using TTN's services, in all its essence it is an online GUI for developers. It lets a developer manage the different applications and devices connected to these. It also auto-generates keys for end to end encryption, such that all applications are kept secure.

### 3.2.2 Devices

By devices, it is meant any end-device which is connected to the network and publishes information to it. So in IOT, this can be anything and everything. As long as a device contains information which can be shared and used, and it is connected to the LoRa-network, it is an end-device. As of 12.2018 TTN only supports the LoRa protocol. However, they have plans to expand their network to include other protocols, which ones are yet to be determined.

### 3.2.3 Gateways

Gateways are the physical "bridge" between the device and the internet. However, a gateway and a device don't "care or know" about each other. This means that the end-device broadcast a message, and whatever gateway that receives the message forwards it. They also listen to the TTN-handler which may queue a downlink for a specific device. This queue is very important since all end-devices cannot be reached at any time(see section 3.1.1). So in the case that a downlink window has been missed, the message is queued for later airing. Thus guaranteeing delivery. These gateways operate on unlicensed bands, meaning anyone can install such device, and by installing a gateway for personal use, expanding the network of TTN.

### 3.2.4 Application

An application is the back-end software utilizing the information an end-device provide. For TTN it refers to an application server. TTN provides two main interfaces: HTTP and MQTT. In our case, the MQTT-protocol has been utilized. The TTN server then works as an MQTT-Broker. This makes it easy to develop custom Clients which interacts with said clients. This will be explained in detail in the following sections.

#### SDK's

The Things network provide an SDK supporting several high-level programming languages:

- Go
- Java
- Node-Red
- Node js
- Python

These libraries enable users to set up a remote MQTT-client, on which they can develop their application. The libraries are all open-source, such that source-files and documentation are available on Github[1].

### 3.2.5 MQTT

Citing The things network: MQTT is a machine-to-machine (M2M)/"Internet of Things"(6). In short, MQTT is a publish-subscribe type architecture, this type of architecture is more scalable than a traditional client-server type architecture. It enables parallel-operation in addition to enabling every end-device to be reachable from any back-end software.

#### Broker

The MQTT-broker is a part of the TTN-handler. These post the even in the form of an uplink message. This is broadcasted to all its clients, which will be notified and receive the said uplink-message. The message which is being sent out a specific event-message which contains the payload from the uplink, but also a big chunk of meta-data. The broker also listens for an answer from the client, this is a downlink-message. A message which will be sent all the way down to the end-device.

[1] The Things Network Application SDK for Python, Available: `https://github.com/TheThingsNetwork/python-app-sdk`

### Client

The client is simply an object which is available from TTN's Git-repository or by using pip(if using python), this is dependant on whichever environment that is being used.
As for the client itself, It works like a standard MQTT-Client. However we need to connect to an application specific broker, an access-key and application-ID must be provided.
The key is an auto-generated which acquired from the TTN-console (see section 3.2.1). This is an aes-128 key, to provide security between the TTN-handler and the client. In addition an application-id must be specified, this is because several applications can be implemented, with different functionality and connected end-devices. So to distinguish between applications, the TTN-handler differentiates between different application id's. It is just like topics in a conventional publish-subscribe architecture.
The handler has a long list of operations and functionality, however, in this case, only one will be highlighted. More precisely the callback. This is a routine the client run every time it receives an event from the broker. An event is a UplinkMessage-Object from the TTN-broker/Handler, this object is constructed specifically from each uplink an end-device sends. A detailed value description can be seen in Appendix B.
The callback-routine can be customized, and an answer constructed if needed. In addition to the message-payload, meta-data is attached to the event message. This is important if one would like to message-values such as RSSI, SNR, which gateways received the message, who sent it, etc. To see specifically what it contains see fig. 3.6. This is just a brief introduction to how the client works, more can be read in the documentation(6).

## 3.3 LoRa-WAN Architecture

The architecture of LoRaWAN was modeled in SysML[2] to easier describe the architecture of the system. In addition, it provides a clearer picture of how the system actually interacts not only internally, but also with its surroundings.
SysML is a standardized modeling language abbreviated from UML(A part of OMG[3] standards). As SysML describes itself: SysML is a dialect of UML, which is especially meant for use in system engineering. Thus making it a perfect tool to describe our system. This is an attempt made by the writer to illustrate and describe how the system works. The diagrams speak a very clear language, such that this section will be more illustrative than explanatory. All the following diagrams are made following the SysML-standard.
It's important to state that the following illustrations are based on TTN architecture, which is a common way to implement such a network.

### 3.3.1 Context

The following context diagram was created to illustrate how LoRaWAN interacts with its surroundings. As stated earlier, the architecture is based upon TTN's implementation.

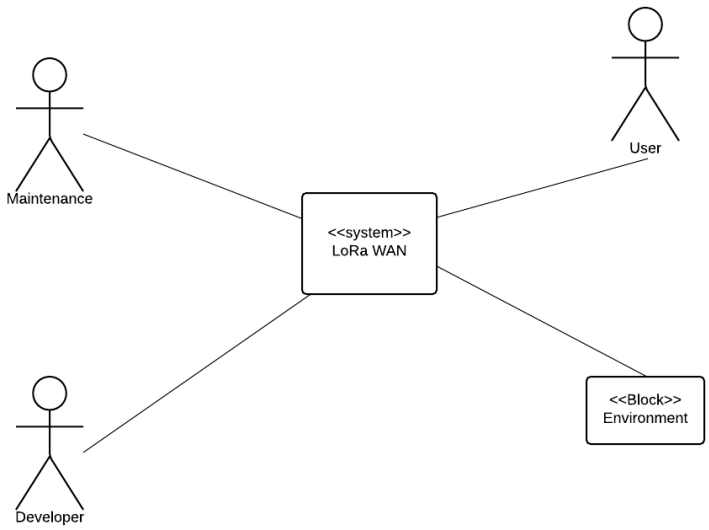



**Figure 3.2:** Context diagram of LoRaWAN, with interfaces to it's surroundings.

[2]SysML Open Source Project, Available: `https://sysml.org/`
[3]OMG | Object Management Group, Available: `https://www.omg.org/about/index.htm`

As we can see in fig. 3.2, there are three actors:

- **User** - Anyone using an application
- **Maintenance** - The Network provider, i.e TTN
- **Developer** - Anyone developing an application or end-device
- **Environment** - Where ever the information is collected

### 3.3.2 Structure Diagram

The following diagram shows the basic structure of LoRaWAN.

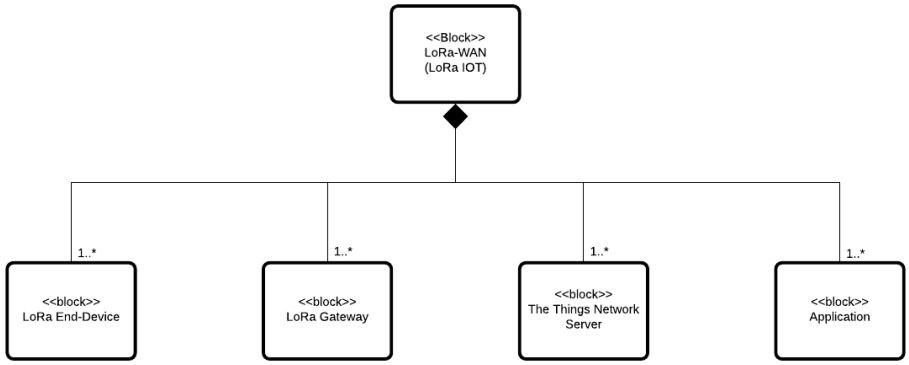



**Figure 3.3:** Basic structure of TTN's implementation of LoRaWAN.

It is worth mentioning that the Application and Server block[4] can be merged together, to form an application which runs on a server. In this example the server works as a broker, and the application as a client(See section 3.2.5 and section 3.2.5). As seen in fig. 3.3, there is a minimum of one instance of each of the 4 subsystems.

### 3.3.3 Sequences

A sequence diagram shows parallel processes distributed throughout the system and their interactions in the case of an event. For our diagram, the case/event is when an uplink-message is sent, the diagram shows all the actors and the communication between them during such an event.

[4] [ 1..* ] mean: At least one or more instance of this block.

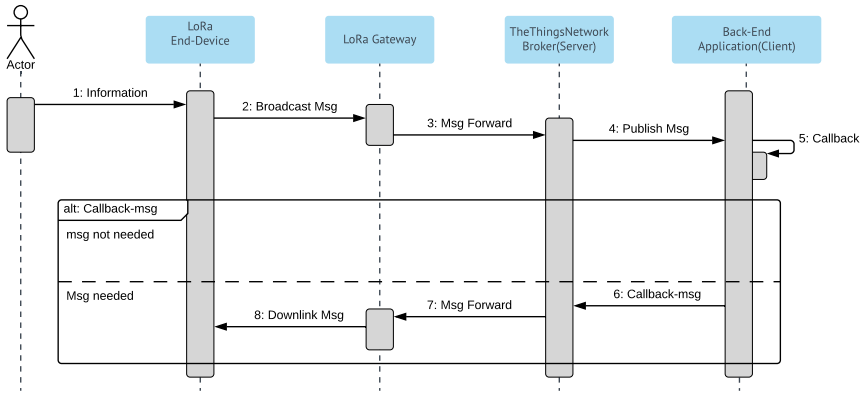



**Figure 3.4:** Diagram showing a sequence of sending a uplink message from an end-device.

### 3.3.4 Collaborations

The following collaboration diagram was made to show collaborations between actors within the system. This puts emphasis on how each actor communicates, however not how they work internally.

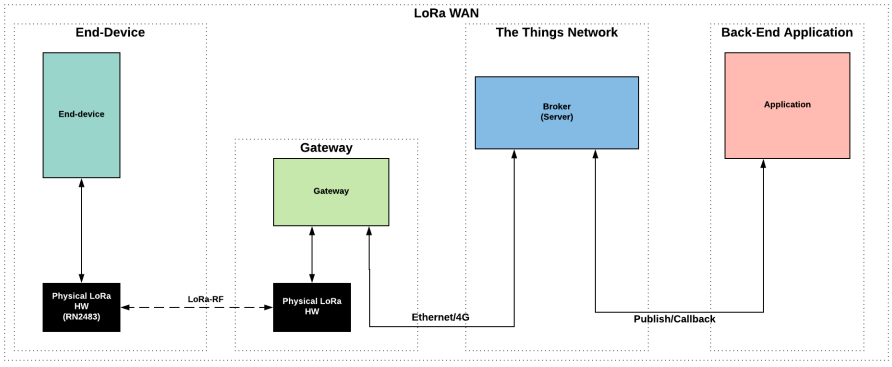



**Figure 3.5:** An illustration showing how each actor within the system communicates.

### 3.3.5 Class Diagram

A class diagram is a static structure diagram which shows the systems classes and their relationships to each other. In our case, it shows the systems actors and their underlying attributes and relations. It also shows some important value types and how they are distributed throughout the system.

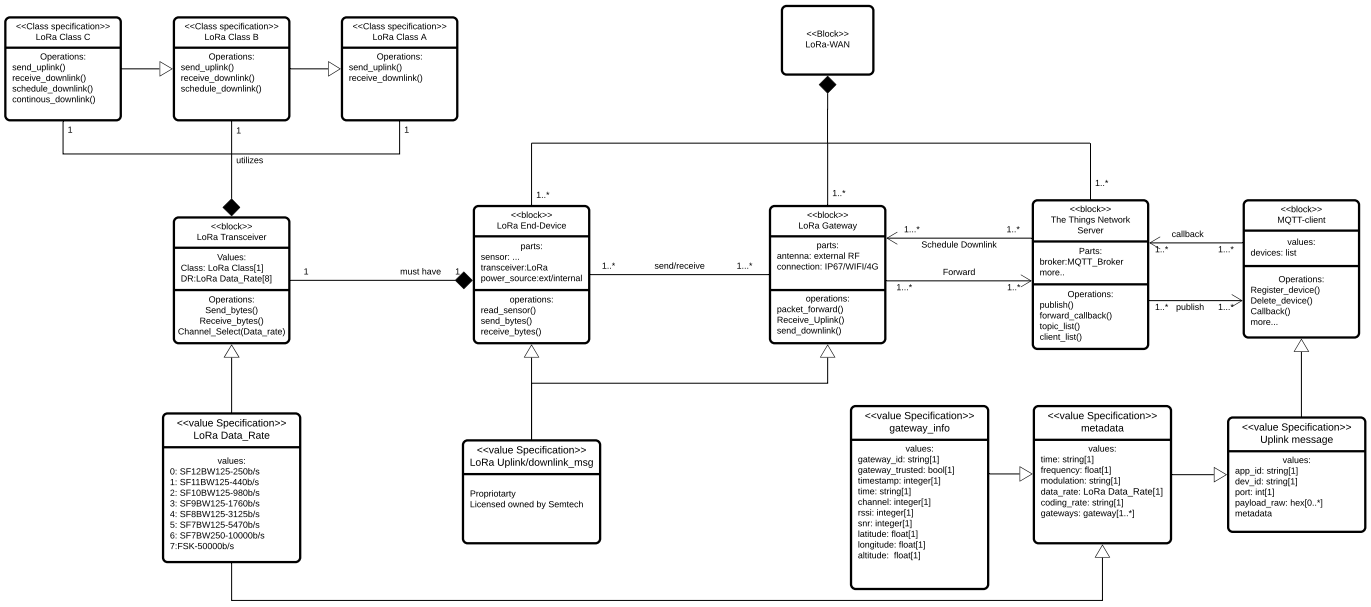



**Figure 3.6:** Class diagram showing the entities throughout LoRaWAN, as TTN has implemented it.

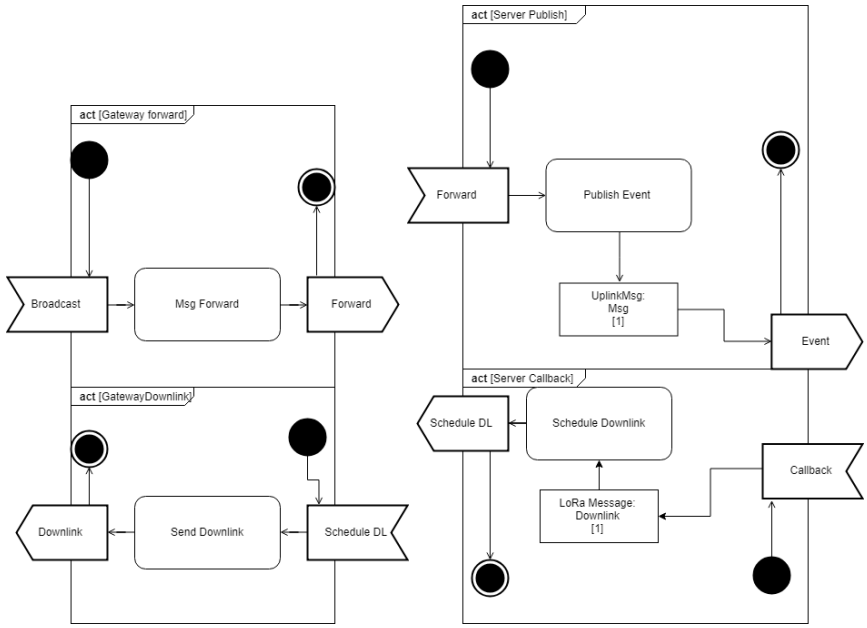
There are some elements in fig. 3.6 that needs to be highlighted. Now we can easily see the relationship between the different LoRa Classes from section 3.3.5. Now we can easily see how the previously objects mentioned in section 3.1 and section 3.2 relate to each other. In addition to this, the class diagram puts some emphasis on what physical and operational attributes each object has.

### 3.3.6 Data Flow

An activity diagram shows how data flows throughout the system, and how the system reacts to different types of data. With that in mind, each of the actors in our system has a corresponding activity diagram. It must be mentioned that this is done on a superficial level, meaning that some functionality is hidden an seen as atomic actions. An example is the gateways and the server, which are illustrated as easy as possible. This is done to make the illustrations easier to understand, and does not go at the expense of how the system works. This is explained thoroughly in the literature about LoRa(see (3)(1)(5)(7)(8)).
The complete activity diagram can be found in Appendix B

#### Class A End-Device

#### LoRa Gateway and Server




**(a)** Gateway **(b)** Server/Broker

**Figure 3.8:** Illustrating an simplified view of the gateway and server.

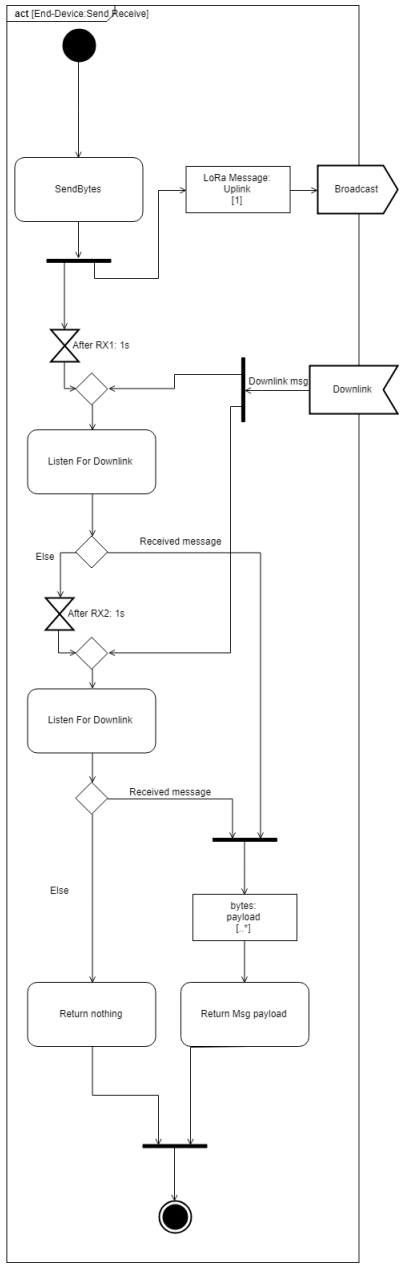



**Figure 3.7:** Activity diagram showing the inner workings of a LoRa end device Class-A during transmission.

### Back End Application

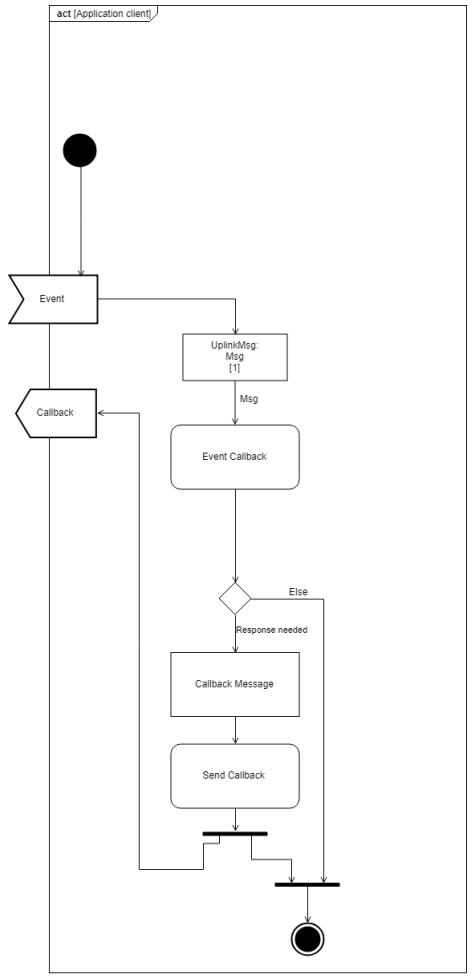



**Figure 3.9:** The inner workings of a typical MQTT-Client upon receiving an uplink.

### 3.3.7 Network Diagram

The network diagram in fig. 3.10 does not only show how the system communicates. Also, it shows how the main pieces of hardware and software are distributed throughout the system.

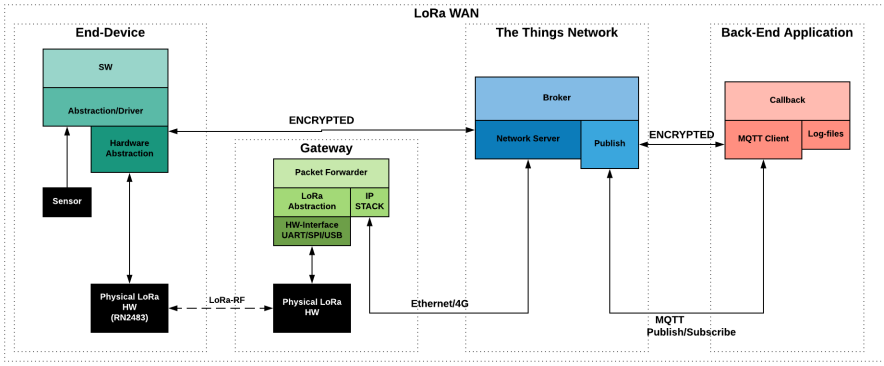



**Figure 3.10:** The inner workings of a typical LoRaWAN implementation, in this case TTN.

Chapter 4

# Specification and Design

The main scope of this task is to investigate and learn how to implement a LoRa end device, and facilitate an investigation of the connectivity properties of LoRa.
To realize this task, it was decided to create a door-type keypad which reads a 4-digit code, then verify that whether the code is correct or not, by passing it to a back-end application. Upon sending the code it must receive an answer with an acknowledgment. This covers the full functionality of LoRa Class A, as stated in section 3.1.1. However, it is important to state that anything hardware related in the process of creating this device, was outsourced as a specialization-project: TTK8. Thus we only consider the design and implementation of the SW in detail.

## 4.1 Hardware

The hardware is as mentioned in the introduction, covered by TTK8. A summary is provided below defining the specification for the hardware, for a detailed view refer the report(9).

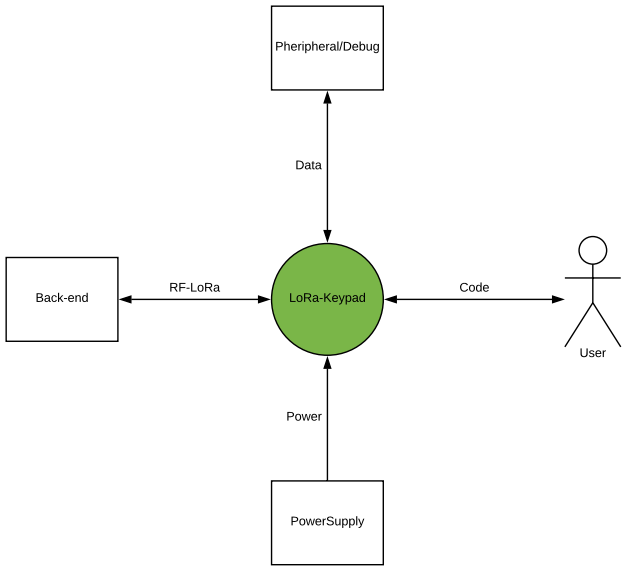



**Figure 4.1:** Context-diagram taken from TTK8-report(9).

### 4.1.1 Specification

Based upon fig. 4.1 the functional specification was derived:

- Be able to communicate via LoRa.
- Have a keypad-interface.
- Indicators to inform the user of the current state of the keypad.
- Some kind of Micro-controller, for processing data and to tie things together.
- Have some way to write to a screen(for debug).
- Ability to be programmed/reprogrammed.

**Table 4.1:** The hardware specification as listed in the report(9).

This covers the functional specification of the Hardware and the basic functionality of the physical keypad. As mentioned this is well covered in the TTK8-Report(9).

### 4.1.2 Hardware Design

Based on these specifications a rough design of the LoRa-keypad was made:

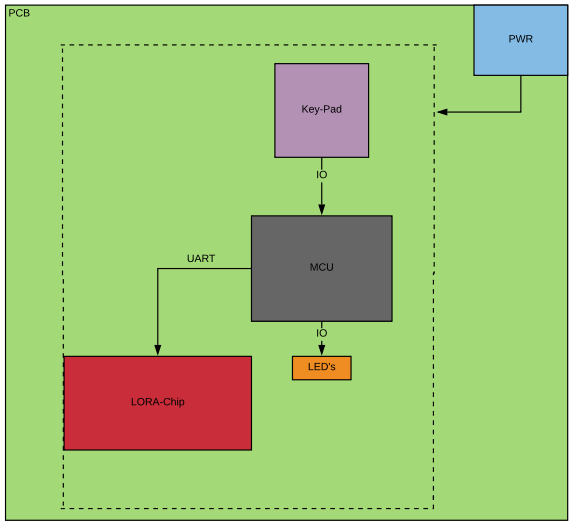



**Figure 4.2:** Block diagram showing the basic layout of the keypad as stated in (9).

The keypad consists of the following hardware: a LoRa transceiver, a micro-controller, 3-separate LED'S, and a keypad. In addition to this, it has some kind of interface for power. Considering that this is an embedded system, the distribution of software will have the same outline as the hardware. Thus fig. 4.2 serves as the basic design for the software as well as the hardware.

## 4.2 Embedded Software

In this case, there is a door-style keypad which utilizes LoRa to check a database whether a code is valid or not. So context diagrams were made to visualize how our system interacts with the environment. This is to easier determine which requirements the system has, both functional and non-functional. This system will contain 2 separate entities: A LoRa End-device(let's call it LoRa-keypad) and a server/handler(back-end) which manages the codes. So we must consider both in terms of interactions and context.

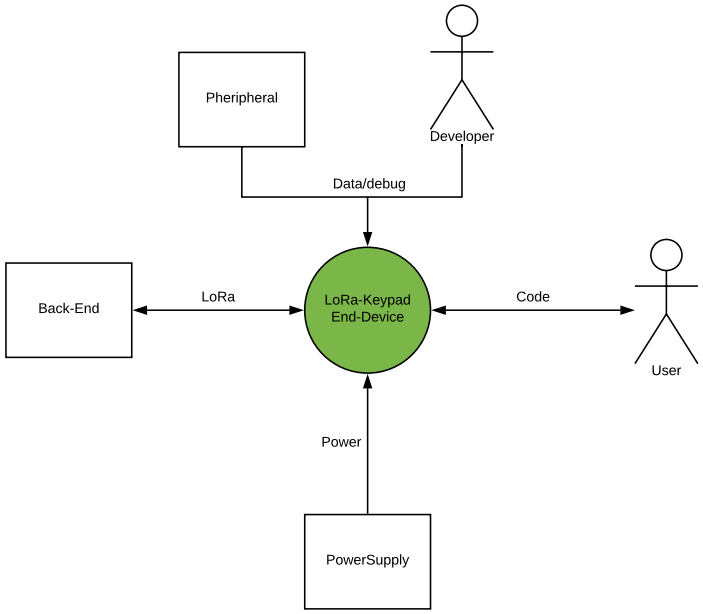



**Figure 4.3:** Context diagram, showing the LoRa-keypad and its interactions with external actors.

By looking at the context diagram we can easier identify how the LoRa-keypad can interact with external actors, and which requirements this puts on the keypad itself.

### 4.2.1 Functional Requirements

The requirements were identified and are listed as follows:

- Have a LoRa interface, to communicate with the back-end.
- Send messages on specific channels, chosen by the operator(for signal investigation purposes).
- A keypad-interface which can take in a code from a user.
- Some indication for feedback to the user.
- Interface for data, for peripherals/developer.

**Table 4.2:** The functional specification derived from fig. 4.3.

### 4.2.2 Acceptance Criteria

These sections specify the acceptance criteria for the keypad, but the functional requirements have not been defined yet. These are covered in the TTK8-report and are the same for both scopes. The difference as mentioned earlier, is that TTK8 focuses solely on HW, while here the focus is SW. To summarize the acceptance criteria for the LoRa-keypad:

1. Must be able to convey a code to the back-end by using the LoRa protocol.
2. Must be able to receive answer from the backend for code confirmation.
3. Logic to choose different data-rates.
4. Be able to take in a 4-digit code from a user.
5. Visually notify the user of the current state with 3 separate LED's(Red, Green and Yellow).
6. Be able to convey data over for debugging or peripherals.
7. Do all the above in a seamless way, from the users perspective.

**Table 4.3:** The functional criteria derived from the functional requirements.

## 4.3 Back-End Software

The back-end is a simple MQTT-client which handles the codes sent from the LoRa-keypad. Additionally, it must log signal-specific meta-data for later analysis. The following context diagram was made:

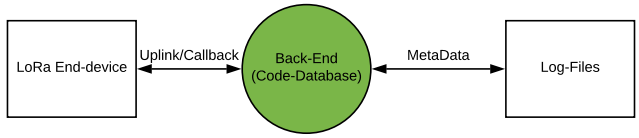



**Figure 4.4:** Context diagram, showing the back-end and its interactions with external actors.

### 4.3.1 Functional Specification

Based on the diagram provided in fig. 4.4, the functional specification can be stated in the following list.

- Serve LoRa-keypad by verifying incoming codes.
- Log meta-data.

**Table 4.4:** The functional requirements derived from fig. 4.4.

### 4.3.2 Acceptance Criteria

As fig. 4.4 indicates there are only 2 interfaces: callbacks/uplink and writing Log-files. So a short list of acceptance criteria was made:

1. Serve the end-device seamlessly as long as it is running.
2. Contain a data structure containing valid codes.
3. A uplink containing a code should always be answered with an downlink containing an ack or nack depending of the code-validity.
4. The signal-specific meta-data of every uplink-message received should be logged to files and on a reasonable format.
5. The logs should be gateway-specific, i.e one file for each gateway.
6. The logs must be distinguishable in terms of date and time.

**Table 4.5:** The acceptance criteria for the back-end.

Chapter 5

# Implementation

The implementation of the system was done in three different processes: The End-device Hardware, End-device Software, and Back-end Application. This chapter covers each one of these processes, the sum total of which becomes our entire system.

## 5.1 Hardware

As stated several times before: this was covered in the TTK8-Project. A more detailed view is therefore provided in the report itself(9).
The main tool for implementing and developing the hardware was Altium Designer, this was used to develop the circuit design as seen in Appendix B. But also the PCB-layout also seen in Appendix B. However, the components were selected each for its own reasons.

### 5.1.1 Components

The hardware components were selected based upon the specification, but also in terms of compatibility with each other. A short summary of which components were chosen and why is provided below, for details refer the report(9). This summary only covers the components vital for software development, so components like crystals and power regulators are not mentioned here.

#### LoRa Transceiver: RN2483

The transceiver which was chosen is a LoRa transceiver from Microchip. The RN2483 supports all LoRa Class A operations(see section 3.1.1), and datarates DR0 - DR5(see table 3.2). This combined with the fact that it is one of the most commonly used transceivers for LoRa, made the decision fall upon this component.

#### Micro-controller: Atmega324PB

The micro-controller was selected based upon the typical implementation of the RN2483 and the peripheral constraints on the system. As stated in fig. 4.3, it must provide data-interfaces to a Peripheral, Debug and a LoRa-transceiver. Since the most commonly used serial standard is UART, and the fact that the RN2483 uses UART for external communication, the MCU had to support at least 3-UART interfaces. The MCU had to be programmable/re-programmable, and a standard utility for this is JTAG. Thus the Atmega324PB was chosen based upon these criteria. It has 3-UARTs, and Atmel studio provides support for programming and debugging with JTAG.

#### Passive Components

There are 2 main components that are needed however they are passive, these are the LED and the keypad. The keypad chosen was a simple 7-pin 3x4 Matrix-style keypad.
The LED's were chosen to be low-power(only draws $2\,\mathrm{mA}$ each), and has the colors: Red, Green, and Yellow.

### 5.1.2 Final result

The final device met all the necessary criteria and was deemed a success. The only acceptance criteria which were not met was the fact that the power-led did not work. This does not affect the performance in any conceivable way and therefore considered negligible. The final result can be seen in fig. 5.1.

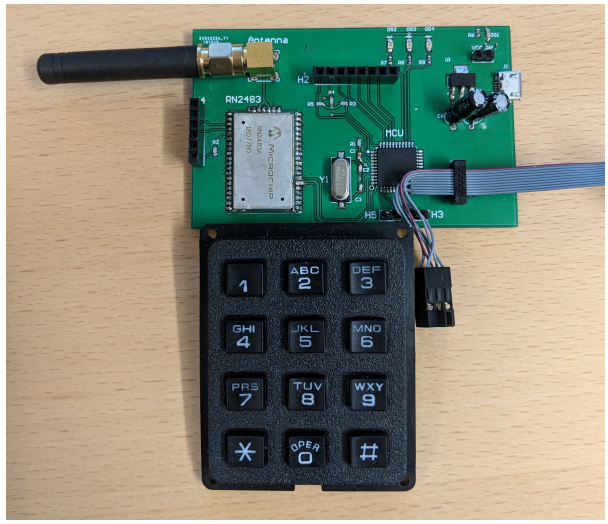



**Figure 5.1:** The final result of TTK8-project. The keypad is detached from its header H2.

## 5.2 The Things Network Console

As mentioned in section 3.2 the network which has been used in this project is The Things Network. Before anything can be done, one has to become a member, which is free. Each member gets their own development console, this is a GUI in which developers can manage their applications and devices attached to these applications. So early in the implementation, an application was made and the end-devices registered under this application. Each application gets a unique identifier, as well as access keys for back-end applications and end-devices. These are end-to-end encrypted. A screen-shot of the console is provided in the Appendix B.

## 5.3 Embedded Software

The only embedded software in this system is on the End-device/LoRa-keypad. This section covers how this was implemented and which tools were utilized.

### 5.3.1 Language and Tool-chain

As the micro-controller which was to be programmed was an Atmega, the main tool for developing the software fell naturally on Atmel Studio 7. Atmel studio provides compatibility with the *avr-gcc* tool-chain, which is the one used here. However, instead of programming in standard C (which is common), it was decided that C++ provides more possibilities. The tool-chain stays the same for C and C++, thus the only main difference

is the introduction of classes and some. An alternative to using Atmel Studio is to use the tool-chain directly and *avr-dude* for flashing. It was deemed that using Atmel Studio is not only simpler but also more convenient. Since Atmel Studio also offers the possibility for debugging with Atmel-ICE through JTAG.

### 5.3.2 LoRa Transceiver: RN2483

Early in the process of developing the embedded software, the creation of a driver for the RN2483 was deemed necessary. There are drivers available online which is written for Arduino. However, in an embedded-application like this, the dependencies/overhead the Arduino libraries bring with them are undesirable and hard to track. Due to the lack of availability to an RN2483-chip, it was decided to ditch the effort of creating the said driver, and going for the already functional Arduino-driver. This lead to the inclusion of Hardware/Software streams from the Arduino library. This the driver was provided by Wireless Trondheim(10) which is a facilitator for IOT-development in Trondheim. They also provided with a LoRa development-kit: *SODAQ ONE*. However this kit is meant for commercial use, and therefore only supports the Arduino tool-chain. However to be able to use the driver, understanding it was crucial. Thus the used routines from the driver are provided below.

#### Joining A Network

For a device such as the LoRa-keypad, it must use the OTAA procedure(see fig. 9.1). Thus it must provide an appEUI(Application identifier), appKey(Encryption key) and the devEUI(Device identifier). For the RN2483, these are set as parameters for the transceiver itself, by writing commands through UART directly to the chip. These commands are specified in the Command Reference(4), the same applies to the Data-Rates. Upon joining the network it is either accepted or not. This procedure must be done every time the device gets powered up.
All this is done by using the MAC-layer protocol for LoRa, this is not discussed in detail in this paper.

#### Transmitting Bytes

Transmitting on LoRa is done in bytes, thus every transmission must be in bytes and if not be decomposed into bytes. For a 16-bit integer, for instance, it needs to be decomposed into 2 bytes before being sent. Then re-assembled upon arrival at the back-end, this means that dependant on the payload format, additional logic must be implemented in the end-device and back-end. For the RN2483 the data must be in hexadecimal value. For instance, the value 0x458374 will be sent as 3 separate bytes: 0x45, 0x83, and 0x74.
Receiving messages by listening according to the receive-windows as stated in section 3.1.1.

### 5.3.3 Additional Drivers

In addition to the formerly mentioned RN2483-driver, 4 more drivers were developed. These were for the 3x4 Matrix Keypad, LED's, debug and peripheral. Each of these was wrapped inside a C++ class and declared as an object upon initialization and use.

### 5.3.4 Result: The Lora Keypad

All the drivers and software mentioned above were put together to realize the LoRa keypad. For explanatory reasons, it is easier to use the final state-machine to visualize: fig. 5.2.

As seen in fig. 5.2 the keypad has 4 states. Initialize, IDLE, POLLING and BUSY. Each one will be described in detail.

#### State: Initialize

This is the first state which the device encounters, initialization. Here all drivers are initialized and prepped for usage. In this case, all drivers are objects, thus initializing them is the same as declaring the objects.
As seen in fig. 5.2, the main elements that are being initialized are:

- The LED's
- UART(for communication with the LoRa transceiver)
- Keypad
- The LoRa transceiver(RN2483)

When comparing this list with the previously stated drivers in section 5.3.3, there are some that are not initialized. Debug(UART) and peripherals(UART), as these are not utilized in normal operational circumstances for the keypad. Debug is only utilized during development, and peripheral communication is for possible peripherals(not current).
The last thing happening in the initialization process is the OTAA procedure, as this is an generic device. As formerly stated this requires a device-id, application-id and an application key. These must be provided in the software for OTAA, to read more about the actual procedure see section 3.1.5.

#### State: IDLE

After initialization, the keypad enters an IDLE-state. In this state, the keypad is regularly polled and the yellow and green LED are lit. Upon polling a key-stroke, the key is saved to a temporary variable and the device switches state to POLLING.

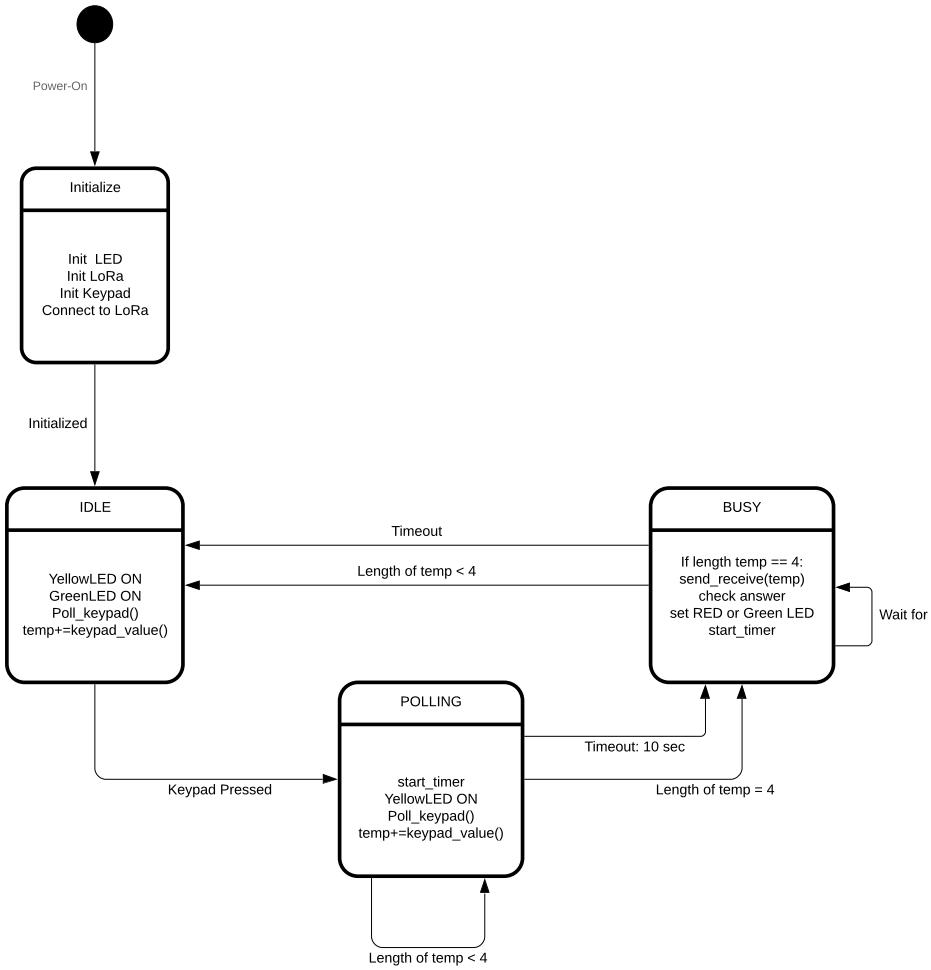



**Figure 5.2:** State diagram showing how the LoRa keypad operates.

**State: POLLING**

In polling mode, only the Yellow LED is lit, this is to indicate that the device is currently waiting for more input. Upon entering this state an interrupt timer is started, this is to give the user $10\,\mathrm{s}$ after each key-stroke to push another key. If a key is pressed, and $10\,\mathrm{s}$ passes without any keys being pressed, the keypad goes into timeout and switches to BUSY-state(see fig. 5.2).
Each key pressed is stored in the temporary value, when the length of the temporary value reaches 4. A four-digit code has been entered, at this moment the device also switches to BUSY-state.

**State: BUSY**

There are 2 ways for the device to enter this state. Either by timeout or when the entered code is 4-digits long. Thus this state start with checking the temporary code length, if the code is less than 4 digits long, it switches straight to IDLE(since it already timed out). However, if the code, in fact, is 4-digits long, the code is decomposed into bytes and aired via LoRa. When an answer is received, it contains either an ack or a nack. The green or red led is lit based upon this answer, these stays lit for $5\,\mathrm{s}$ before timing out. Which when it does, switches states to IDLE. If however, the device never receives an answer, it lights up the red LED for the same period of time, before switching states.

### 5.3.5 Data Collect Mode

In addition to working as a standard keypad, the LoRa-keypad was fitted with another project-specific state: Data Collect Mode. To enter this mode, SQUARE has to be pressed when powering up, until all the LED's turn on. In this mode, the operator is able to send a 1-byte message in any data-rate which the RN2483 supports. To send a message, the buttons between 1-6 must be pressed, this sends a message on the channels DR0-DR5 respectively. The mode was added for the ability of data-collection on different channels. As seen in fig. 5.2, this state is not present. This is a developer-feature and is really just implemented for the task's sake as stated in point 3 in Task Description.

## 5.4 Back-end Software

The back-end software was realized by using the SDK's provided by TTN(see section 3.2.4). As python is a familiar and widespread language, the choice fell on the Python-API. And more precisely the MQTT objects which TTN provides. Further explanation is provided in the following sections:

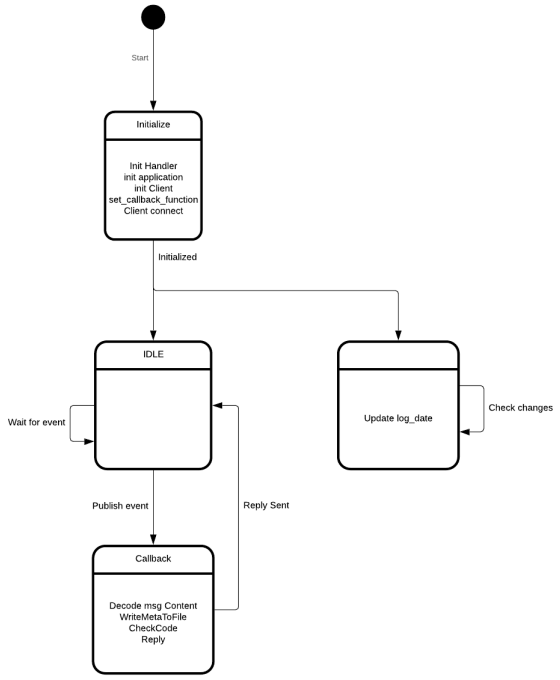



**Figure 5.3:** A state diagram illustrating the states for the back-end/code-handler.

### 5.4.1 MQTT Client and Callback

The API provided by TTN is a complete library with containing 3 separate objects:

- HandlerClient
- ApplicationClient
- MQTTClient

In this case, only the MQTTClient were used. It connects to an application which is registered on TTN's handler. So the application is the one created in section 5.2. So in order to connect to the TTN-handler/broker the client must be configured with the right application-id and key, thus subscribing to the specific application and all its end-devices. While it may be connected, routines need to be specified. All of these routines are callbacks of some kind. Here it is an Uplink callback. To specify, this a function which runs whenever an end-device sends an uplink-message. An illustration of this series of events is provided in section 3.3.6.
So for the callback routine for the code-handler could be summarized through the illustration in fig. 5.3 and by the following list:

1. Decode Message payload(from base64 to integer)
2. Write meta-data to file.
3. Check code-validity.
4. Reply with ack/nack.

This is a simple implementation of the the Client and callback-routine. And is sufficient in serving the keypad.

### 5.4.2 Logging Meta-Data

Whenever an uplink is sent, the TTN-handler constructs an Uplink-Message Object. This is illustrated in Appendix B. The data which is most interesting is the Data-Rate, time-stamp, RSSI, SNR, and gateway-coordinates. All of these values are being written for to *.csv* files, one file for each gateway and each day. Each uplink is written in the following format as a line in a .csv file:

| *Data-rate, time-stamp, RSSI, SNR, Latitude, Longitude, Altitude* |
|---|

An example:

| *0, 1541677183, -114, 2.75, 63.397568, 10.400948, 21* |
|---|

A list defining each value:

| | | |
|---|---|---|
| Data-rate | - | Integer from 0 - 5, defined in table 3.2. |
| time-stamp | - | Integer for local time in Posix-format. |
| RSSI | - | Integer in $\mathrm{dBm}$. |
| SNR | - | Integer in $\mathrm{dB}$. |
| Latitude/Longitude | - | Float for Lat/Long-coordinate for gateway. |
| Altitude | - | Integer for gateway altitude. |

This can easily be read and analyzed in mathematical-tools like Matlab. Also to see how this relates to the UplinkMessage see the value description in Appendix B.

## 5.5 The Complete System

The complete system consists of the keypad, TTN, and The back-end code-handler. The complete system can be modeled to better illustrate how the flow of information propagates through the system. The following sequence diagram shows how the system will function.

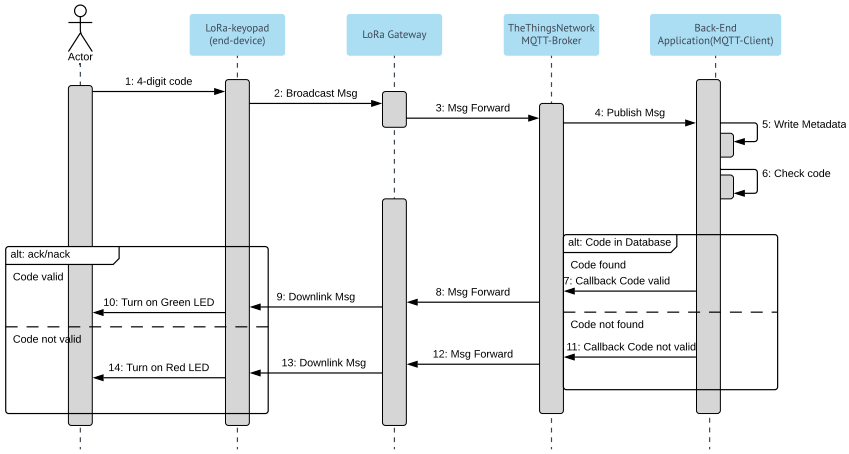



**Figure 5.4:** Sequence diagram showing how the system convey signals when a code has been entered. The signal-list is can be found in Appendix B.

Similarities can be seen between fig. 5.4, and the sequence in fig. 3.4. Also to put everything in perspective and see the relations between each actor in the entire system, see the class diagram in Appendix B

Chapter 6

# Testing and Results

As point 3. in the Task Description points out, a big part of this task concerns the investigation of connectivity properties. Thus the focus will shift over to the connectivity properties from this chapter and throughout the discussion. This chapter is therefore split into 2 parts, Testing, and results for the Keypad-system. Then testing and results of connectivity properties.

## 6.1 Testing

This section will be divided into two parts: One concerning the LoRa-Keypad system, and another clarifying the approach for data-collection and investigation of connectivity.
The LoRa-keypad was tested both one part at a time and as a device. In the TTN-console developers can see every uplink-message containing payload and meta-data in real time. This feature was used to refine the payload format and to develop the keypad alongside the back-end.
The back-end was tested in a similar manner, first locally part by part. In the TTN-console, developers have the option to simulate uplinks. This method was used to test the callback-function.
The entire system was tested by monitoring the Back-end while using the LoRa-keypad as a normal keypad over and over again, trying to break the system.

## 6.2 Testing Results

Testing of LoRa-keypad was done in the TTK8-project(9), where it was tested up against each component and the back-end. These results can be seen in the report. As a whole, both the LoRa-keypad and the back-end passed all points except for an issue regarding downlink-messages. The issue is: A uplink/downlink U1 and D1 corresponds to a message and an answer. The downlink D1 would not arrive in time according to the receive windows for class A, which is explained in section 3.1.1. However, D1 is not lost but scheduled as a downlink. Then when a new uplink U2 is sent, the keypad receives the scheduled downlink D1. However it is out of sync, U1 and U2 may be hours apart. Thus far out of it's time limits. Apart from this issue, which in the end was fixed, the system worked fine.

## 6.3 Investigating Connectivity Properties

In order to tell something about the connectivity properties a set of data had to be collected. This has to be done in an organized manner, and in a way that provokes differences that which is interesting to analyze. As stated in section 2.1, data-rates should have a direct impact on effective range. This should again be reflected in the data which are already being logged: RSSI and SNR. Thus testing data-rates vs range seemed like a reasonable place to start. So initially three questions arose:

1. Does RSSI and SNR values correlate with the statement in section 2.2
2. How does LoRaWAN cope with a flat urban environment?
3. Does lower data-rates provide better range?

To answer these questions, the following test were constructed:

### 6.3.1 The Test

The test is simple: Find several locations which has similar conditions and in "free view[1]" of the GW. In each of these locations, take a set of samples by sending messages in different data-rates/channels. In addition to these locations, some locations must be chosen within the confines of the city, to test within the urban area.

#### The Sample Locations

So the following map shows where the gateways are located(Blue), and where the samples was taken(Red).

[1] They cannot be seen, however, they are generally un-obstructed.


**Figure 6.1:** Map of each and every test location.

On each of these 6 sites, 8 messages were sent for each of the 6 data-rates. These messages were picked up by 3-Gateways. Giving approximately 864 samples, however, it was taken into consideration that not all gateways would catch each and every message since they were located in a different place under different conditions.
table 6.1 gives the exact location for the sites and gateways where the samples were collected. The information for all these is taken from measurement with a cell-phone and pinpointing the exact location on the map. The main tool was `www.norgeskart.no` which is a service provided by the Norwegian map-administration: Kartverket.

| **Location Name** | **Latitude** | **Longitude** | **Altitude** |
|---|---|---|---|
| Graakallen | 63.420365 | 10.253402 | 542 m |
| Studenthytta | 63.416422 | 10.268119 | 445 m |
| Parking lot | 63.416838 | 10.292888 | 298 m |
| Utsikten | 63.421167 | 10.371173 | 110 m |
| Stiftsgaarden | 63.432012 | 10.395571 | 7.6 m |
| Central Station | 63.435998 | 10.397895 | 3.6 m |
| **Gateway** | **Latitude** | **Longitude** | **Altitude** |
| Samfundet | 63.422495 | 10.395201 | 20 m |
| Olav | 63.433592 | 10.403295 | 19 m |
| Sluppen | 63.397568 | 10.400948 | 21 m |

**Table 6.1:** Table showing sample locations and gateway locations.

This should be sufficient information for anybody which would like to re-create these results.

## 6.4 Plots Showing Collected Data

The collected data was extracted from their .csv files and plotted by using Matlab. The following plots show the results.

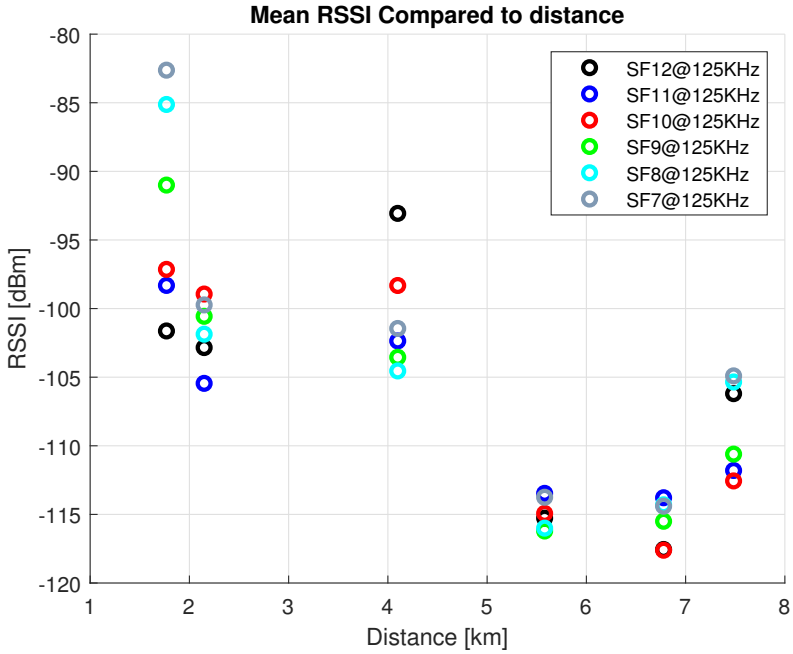



**Figure 6.2:** Mean RSSI compared to distance.

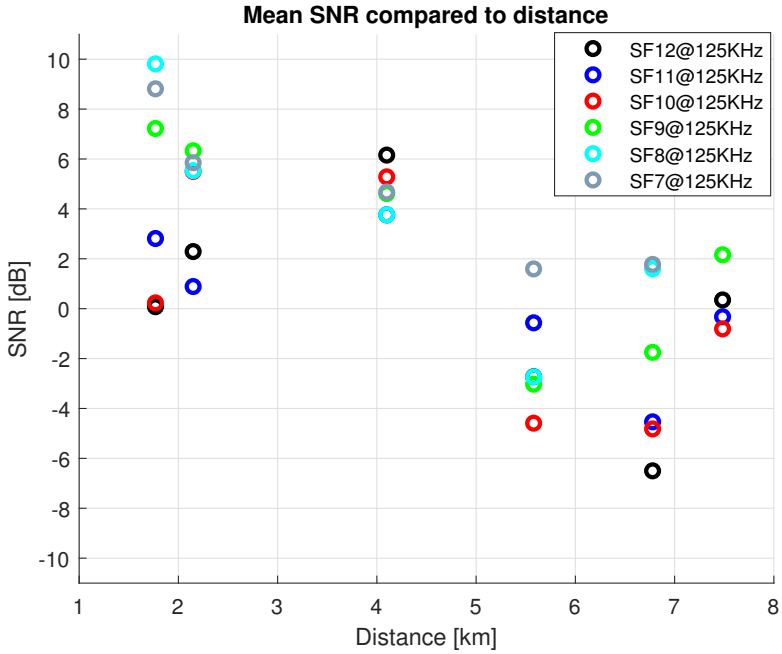



**Figure 6.3:** Mean SNR compared to distance.

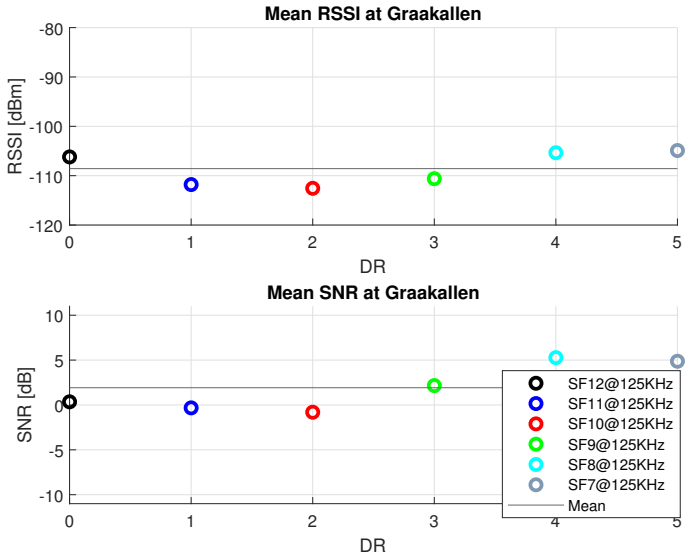



**Figure 6.4:** Mean signal-values at Graakallen.

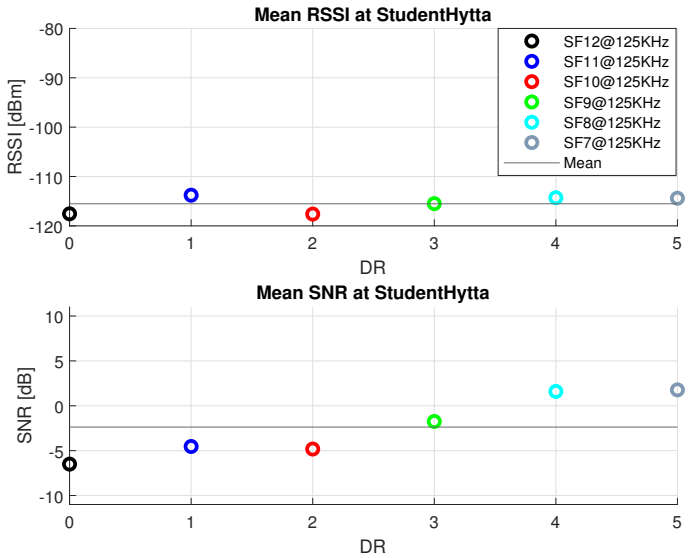



**Figure 6.5:** Mean signal-values at Studenhytta.

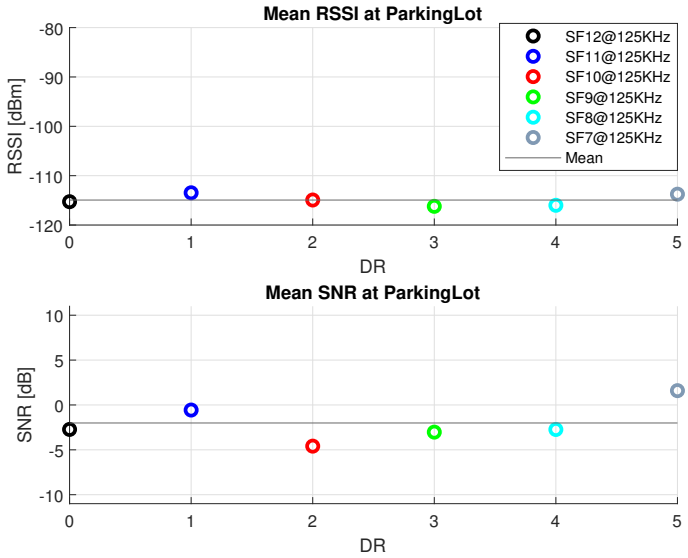



**Figure 6.6:** Mean signal-values at the ParkingLot.

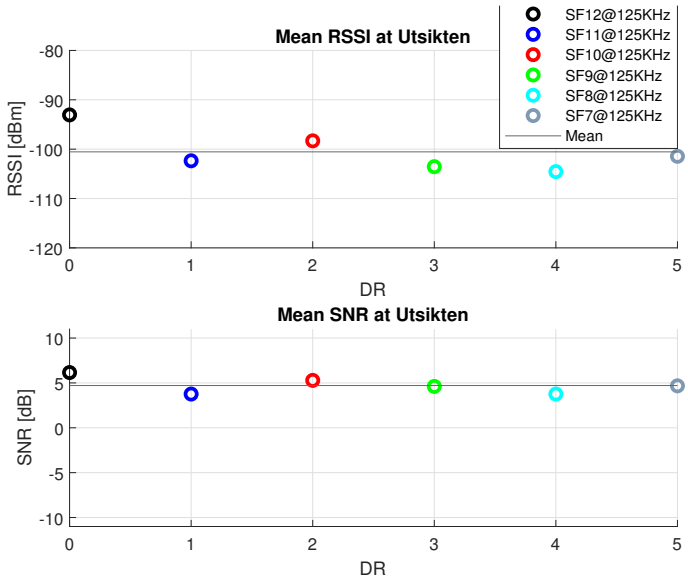



**Figure 6.7:** Mean signal-values at Utsikten.

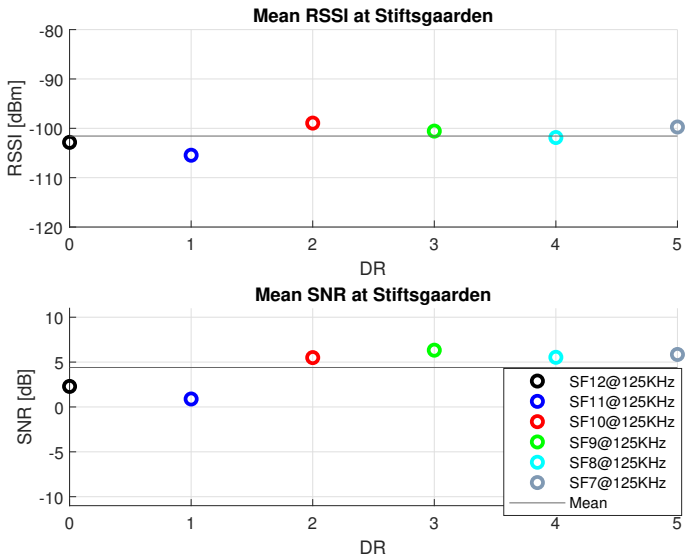



**Figure 6.8:** Mean signal-values at Stiftsgaarden.

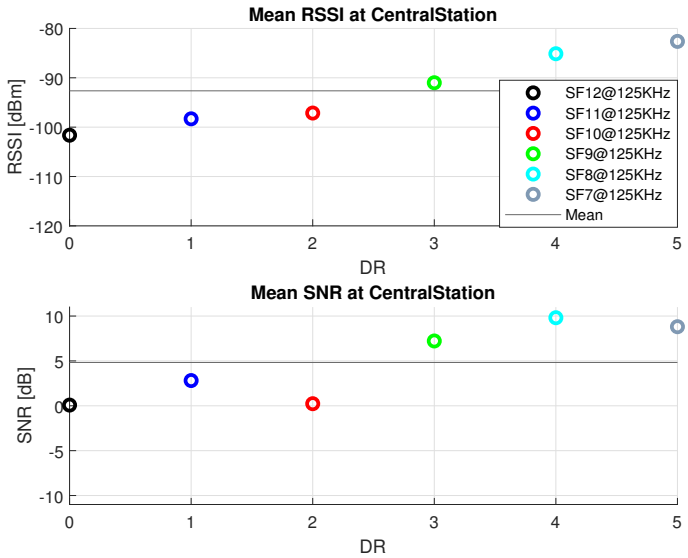



**Figure 6.9:** Mean signal-values at CentralStation.

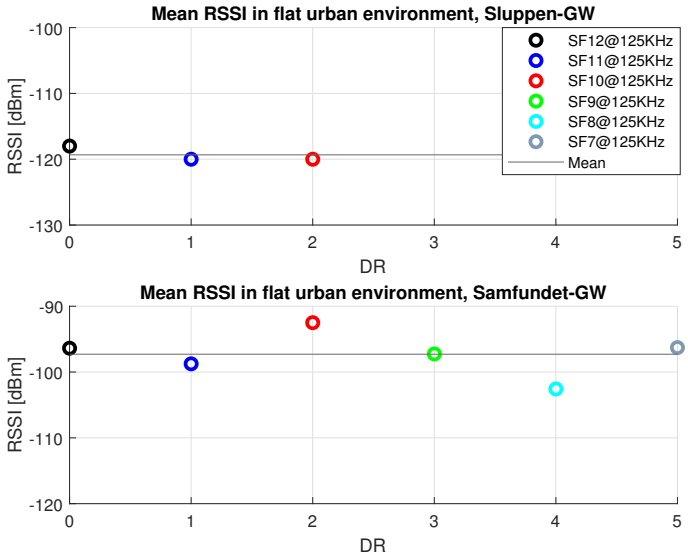



**Figure 6.10:** Mean RSSI across a flat urban area.

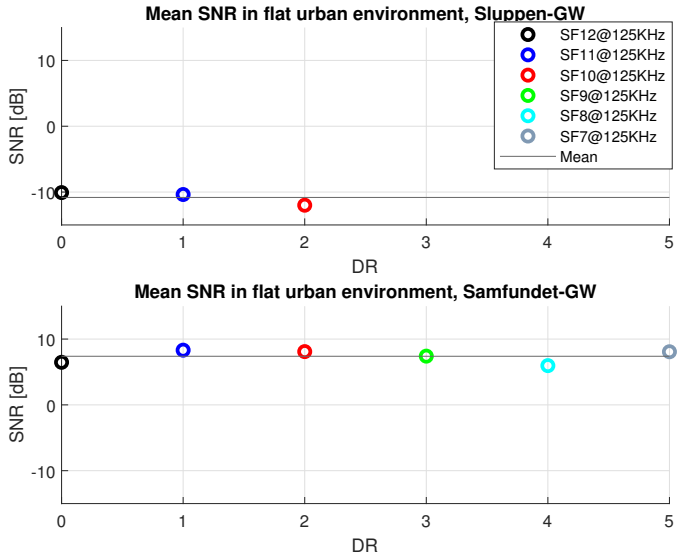



**Figure 6.11:** Mean SNR across a flat urban area.

Chapter 7

# Discussion

This chapter focuses mainly on the testing and the produced results, these results will be compared to the requirements and specifications as given in chapter 4. Also, the results of the connectivity-test given in section 6.3.1 will be reviewed.

## 7.1 Results

This section will focus on the results from testing the entire system. The system as implemented by the author is divided into two sections. These sections will discuss the results and put them up against the previously mentioned specifications.

### 7.1.1 LoRa-Keypad

So a full test of each component was conducted in the TTK8-report(9). And it was verified that the Keypad passed to point 1, 2, 4, 5 and 6 as stated in the acceptance criteria in section 4.2.2.
Concerning point 3, logic was implemented to support this requirement. As stated in section 5.3.5, the logic was implemented as a special state in which a message can be sent in any data-rate supported by the RN2483. And this was successful as chapter 7 will prove.
As for point 7, some issues were discovered when receiving downlinks. This may be due to the fact that after sending a message, since this is a Class A device, a certain amount of time is given for the reply to arrive. However, whether this problem is due to the back-end failing real-time demands or if it is configuration-faults of the transceiver, is not known. Regardless even if it is in need of a little tweaking, overall the system worked as a proof of concept. It may also be the driver which as Arduino, utilizes hardware/software streams. How much overhead these provide is unknown. Additionally, these use dynamic

memory allocation in addition to some timing interrupts. If the porting is done poorly, these may propagate errors. While point 7 in section 4.2.2 is not passed as it does not operate seamlessly, the concept works.

#### Back-End

The back-end contained a simple python-list of codes, which passes for point 2 in section 4.3.2.
For point 4, 5 and 6, these were passed as the system was able to log data and store them in a specific manner. This will also be proved by section 6.2.
It may be hard to decide on point 1 and 3 since, as already stated in section 7.1.1, there are problems in receiving uplinks. Whether or not this is an error caused by the back-end or if it is caused by the end-device is unknown. Nevertheless, the back-end actually always send a response which may or may not arrive in time, thus passing point 1 in table 4.5.

#### The Complete System

Except for the timing issues, the system works as it should. The timing issue was fixed by sending 2-messages for each code, one to send the code, the next to open receive-windows for the downlink. It turned out to be a solid solution. An end-device has been implemented and tested. A back-end has been implemented and tested. As such it should be considered a success.
As for wetter or not the system can send messages often, as stated in section 2.1. It seems as if LoRaWAN, at least in the way TTN has implemented. Restricts the amount of messages which can be sent. Under testing and collecting data-samples, it was discovered that the airtime was averagely around $30\,\mathrm{ms}$. Furthermore, the gateways seemed to refuse to receive messages which were "spammed" from an end-device. After about 6-7 messages it seemed as if the Gateways put the end-device on a cool-down for about 7 minutes before be able to receive again. This however, may be a part of the implementation of TTN, as it is designed for huge amounts of end-devices.

## 7.2 Collected Data

So in section 6.3 three questions were posed. The goal of this section is to discuss whether or not these can be answered.

#### Question 1: Is there a correlation between RSSI/SNR, data-rate and range?

As we can clearly see in fig. 6.2, there is an apparent trend in the RSSI-values compared to distance. The samples taken at close range it has a significantly higher signal-strength than those taken from afar. However, when looking at the data-rates, it seems to look like the

"higher" data-rates provide better in terms of signal strength. The differences, however, are not consistent, looking at the extremes: SF12 vs SF7. It can be argued that SF7 all over has better or similar signal-strength. This may be due to the fact that it uses much more power compared to the SF12, as stated in section 2.3. Also looking at the plots from fig. 6.4 to fig. 6.9. A case cannot be made if whether or not RSSI and channels are correlated, as all channels seem to perform pretty much the same compared to each other. When reviewing fig. 6.4 vs fig. 6.5 and fig. 6.6, there is quite a difference in mean RSSI, $-107\,\mathrm{dBm}$ vs $-116\,\mathrm{dBm}$. This may be due to the fact that there is some vegetation blocking parts of the view at Studenthytta. And reviewing the plots, it seems to have made some effect on the signal strength.

Moreover, looking at the SNR values it can clearly be made a case that the higher data-rates yields a better SNR. Which also makes sense on the same basis of power-consumption and data-rates stated in chapter 2. And logically enough, it can be seen in fig. 6.3: the further the distance the more apparent the noise-disturbance becomes, or so it may seem. Looking closer at the plots from fig. 6.4 to fig. 6.9. The SNR-values seems to be better for "higher" data-rates, and worse for the "lower" ones. As for fig. 6.4, fig. 6.5, fig. 6.8 and fig. 6.9. It is clear that the lower data-rates perform under the mean at the same distance. Thus an argument can be made that higher-datarates yield better SNR. Which also seems logical since these data-rates also utilizes more power.

**Question 2: How does LoRaWAN cope with a flat urban environment?**

As seen in table 6.1 both Sluppen and Samfundet has roughly the same altitude, map provided in fig. 6.1. So the only apparent difference is the distance, and obstructions between them. As already seen in the previous section, distance does make a difference, however not a huge one. So the main difference between these two locations must be the obstructions: the amount of building/urban area between the gateways and the end-device. The distance between them is approximately $2.8\,\mathrm{km}$, which considering the earlier results is not a lot. However huge differences are present in fig. 6.10 and, not only in the RSSI/SNR but also in which of the messages even arrived. It can be easily seen that for the GW at Sluppen, the only messages which actually arrived had the datarates between SF12 and SF10. And taking a look at the SNR, it is below $-10\,\mathrm{dB}$ average. This might mean that the lower data-rates are robust in terms of noise, which might also mean that have better capabilities over long-ranges with varying topology. This might also be part of the answer for**question 3** in section 6.3.

## 7.3 Areas of Improvement

After considering the results, there are some areas which can be improved. Said areas and a proposal for improvement are put forward in this section.

### 7.3.1 RN2483 Driver

As earlier stated in section 5.3.2, the drivers for the RN2483 is written for Arduino. As lack of available RN-2483 chips made it hard to develop such a driver. This also means that all the dependencies from the Arduino library, such as HW/SW-Streams and Strings, are implemented. The solution was to port these over to the Atmega324PB, which apparently was a success. However this may have added an unknown amount of overhead, and the port is most definitely prone to Errors. Moreover, these dependencies bring a huge amount of error-sources with them. Thus an embedded-specific RN2483 driver should be developed, most preferably in both C and C++. This would make the embedded-system not only simpler, easier to debug but also most likely faster. This may also fix the issue with downlink-messages not arriving in time.

### 7.3.2 Method for Collecting Data

The current way of collecting data is kind of a brute-solution. Manually pressing which data-rates and visual verification of actual reception. This could and should be done in a more elaborate and smarter way. A solution is a self-testing system. In which the backend responds based upon how many samples each gateway has. Also, the ability to lock-the end-device to a certain GW will ease the procedure, and also make it possible to come with more conclusive tests.

### 7.3.3 Logging Method

The logging method is very crude, it saves all meta-data from all gateways. It could be added logic to do some calculations in the back-end. For instance, when collecting samples, the back-end only saves the mean and number of samples of each channel. Not each message and its corresponding meta-data.

Chapter 8

# Conclusion

In this project, an embedded-system utilizing LoRa have been developed. The goal was to create a proof of concept by making a door-style keypad which communicates via LoRa. The Keypad is able to take in codes and verify them off-site via LoRa.

In addition to the keypad-system, a mechanism for logging data has been set up. This logging-system combined with a specially designed test formed an investigation into the properties of LoRaWAN. The resulting data were analyzed and plotted.

On top of this, a survey has been conducted into the architecture of LoRa, and the system has been modeled using an OMG-standard modeling language. This to provide a better understanding of not only how LoRaWAN works, but also an insight into how hardware and software are distributed.

Chapter 9

# Further Work

This chapter contains only a list of what may be improved upon and suggestions to areas of interest in a possible master-thesis.

- Develop an Embedded-Specific RN2483 driver.
- A survey into alternatives for the RN2483.
- Improve the way in which data is collected, make it suitable for large-scale collection.
- Implement an own gateway and investigate how to lock end-devices to specific gateways.
- With an own gateway, investigate the capacity of the system in terms of message-spamming.
- Investigate further into how data-rates affects range and connectivity.
- Investigate connectivity indoors.

# Bibliography


[1] LoRa Alliance, "LoRaWAN 1.0.3 specification," Tech. Rep. 1 [Online], Accessible: https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf, 2018. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf

[2] Semtech, "LoRa™Modulation Basics Semtech," Tech. Rep., 2015. [Online]. Available: www.semtech.com

[3] LoRa Alliance, "LoRaWAN 1.1 Regional Parameters," Tech. Rep., 2017. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/lorawantm{_}regional{_}parameters{_}v1.1rb{_}-{_}final.pdf

[4] Microchip Technology Inc., "RN2483 LoRa™ Technology Module Command Reference," Tech. Rep., 2015. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/40001784B.pdf

[5] The Things Network, "The Things Network," 2016. [Online]. Available: https://www.thethingsnetwork.org/

[6] ——, "LoRaWAN | The Things Network MQTT." [Online]. Available: https://www.thethingsnetwork.org/docs/applications/mqtt/

[7] LoRa Alliance, "LoRaWAN™ For Developers," 2016. [Online]. Available: https://www.lora-alliance.org/For-Developers/LoRaWANDevelopers

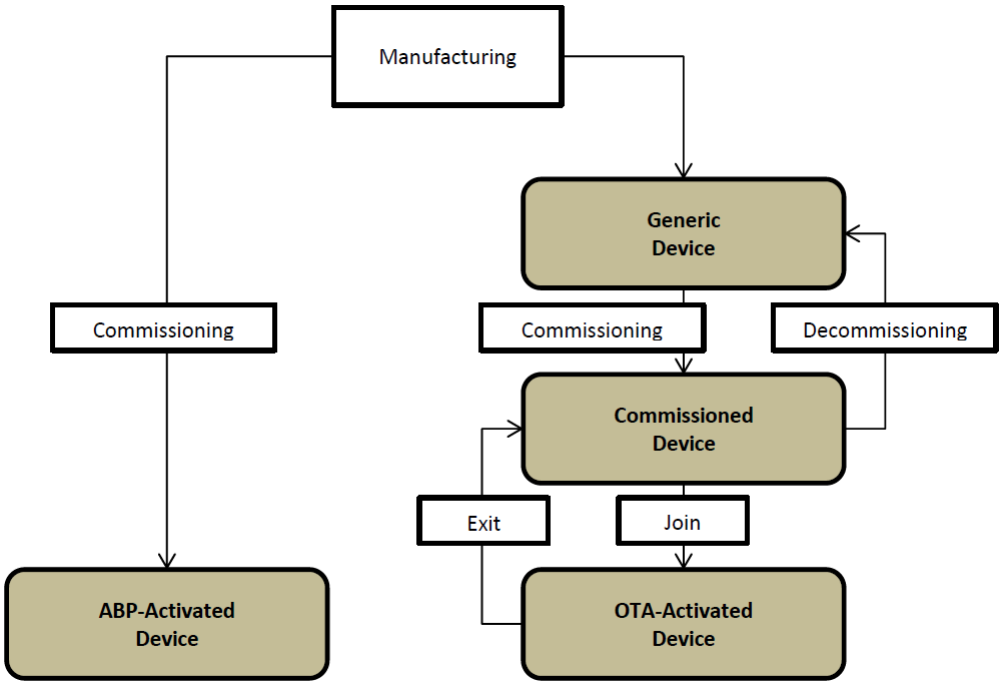
[8] ——, "LoRaWAN™ Backend Interfaces 1.0 Specification," Tech. Rep., 2017. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/lorawantm-backend-interfaces-v1.0.pdf

[9] T. U. Rasmussen, "TTK-8 Design and Implementation of a LoRa End-Device with keypad," no. November, 2018.

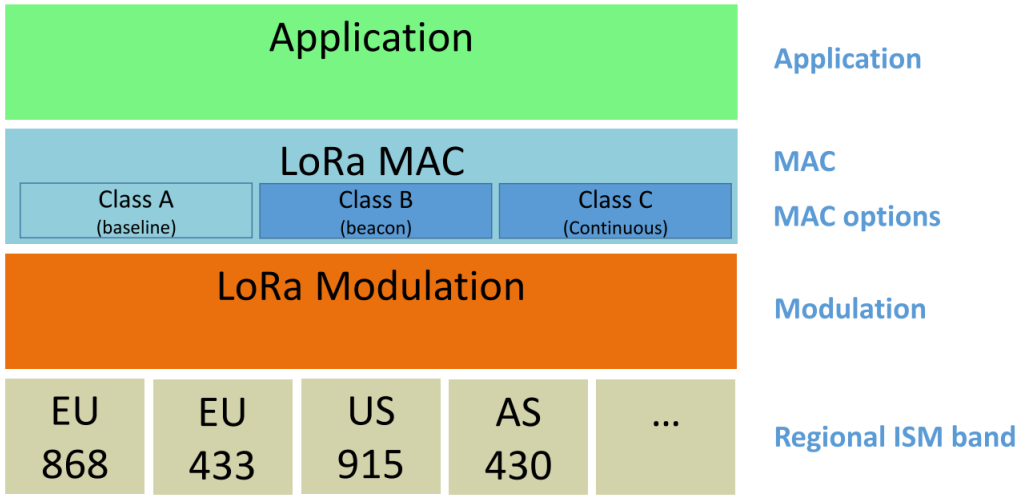
[10] Wireless Trondheim., "Wireless Trondheim." [Online]. Available: https://wirelesstrondheim.no/

# Appendix A

## End-device Activation

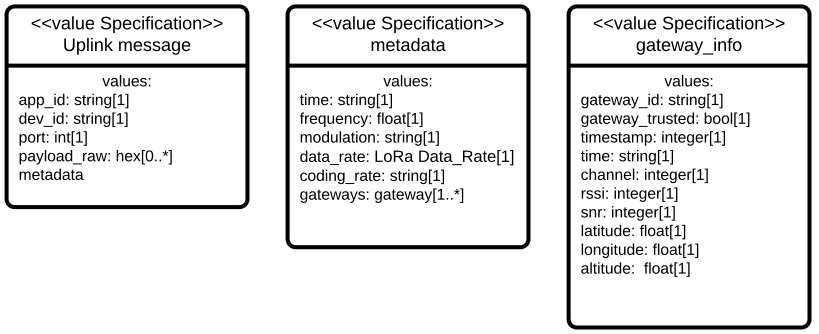



A illustration taken out of the LoRa specification for back-end interfaces(8).



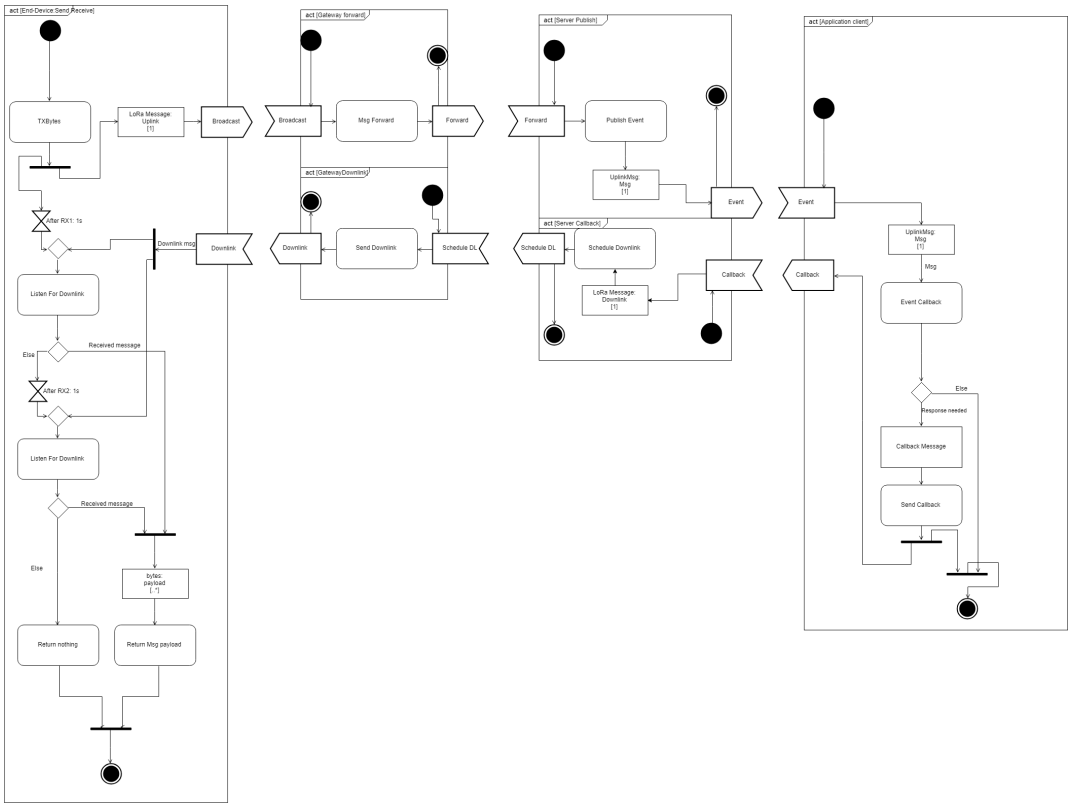

A illustration taken out of the LoRa specification(1) showing the MAC-layer.

# Appendix B

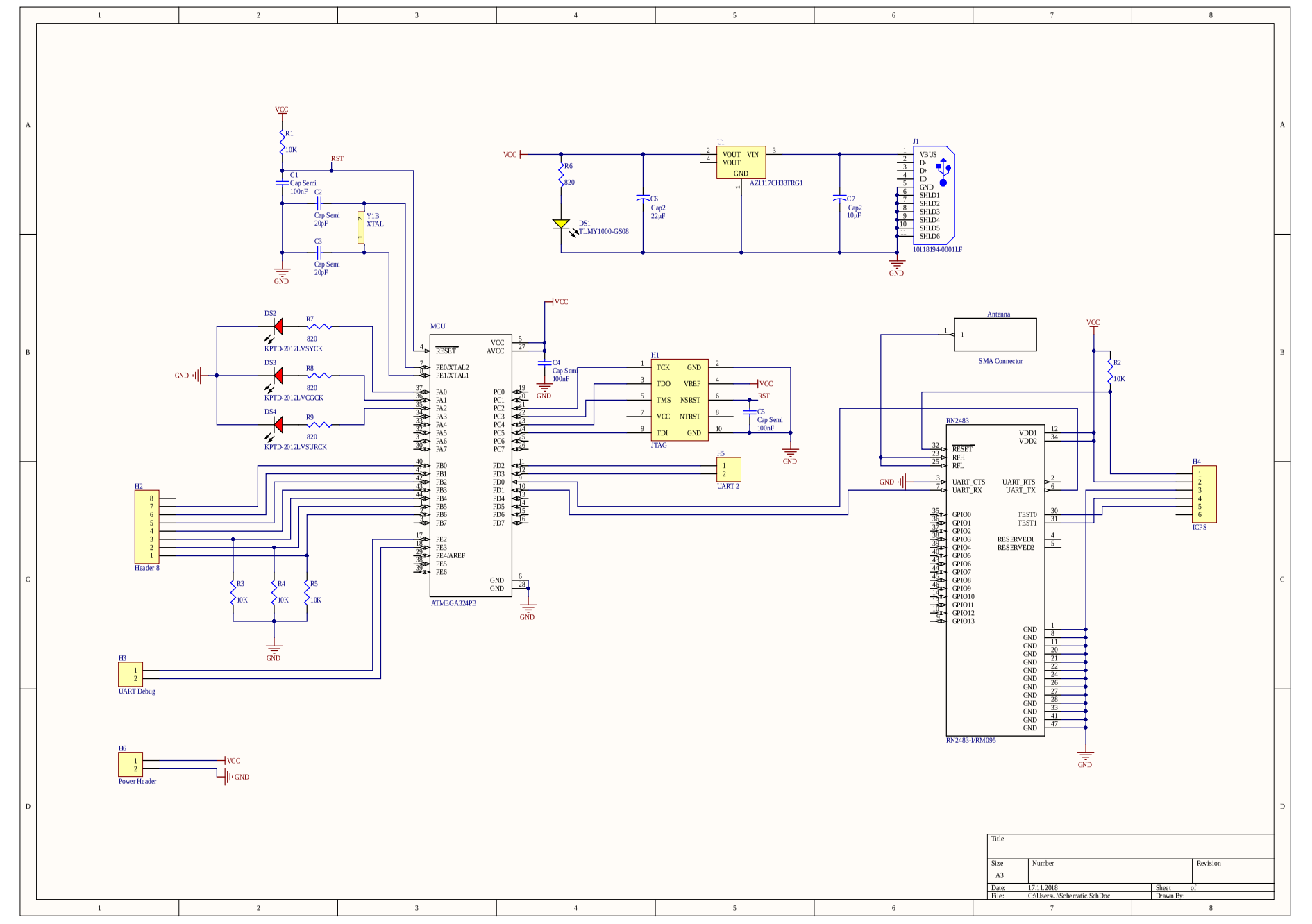



Value specification, showing uplink message and its contents. Relations can be seen in fig. 3.6

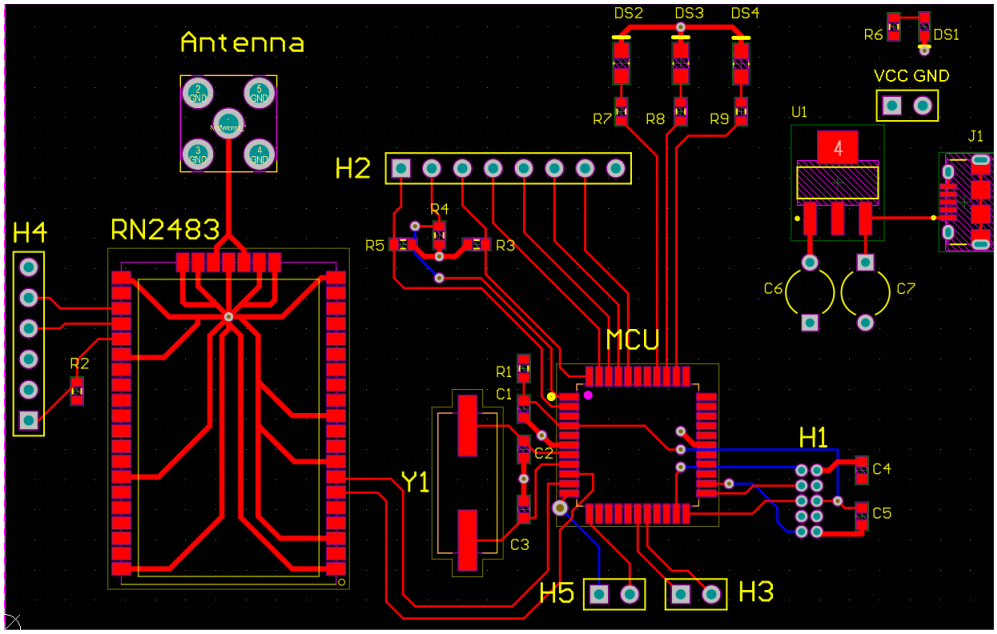
# Full LoRaWAN Activity Diagram



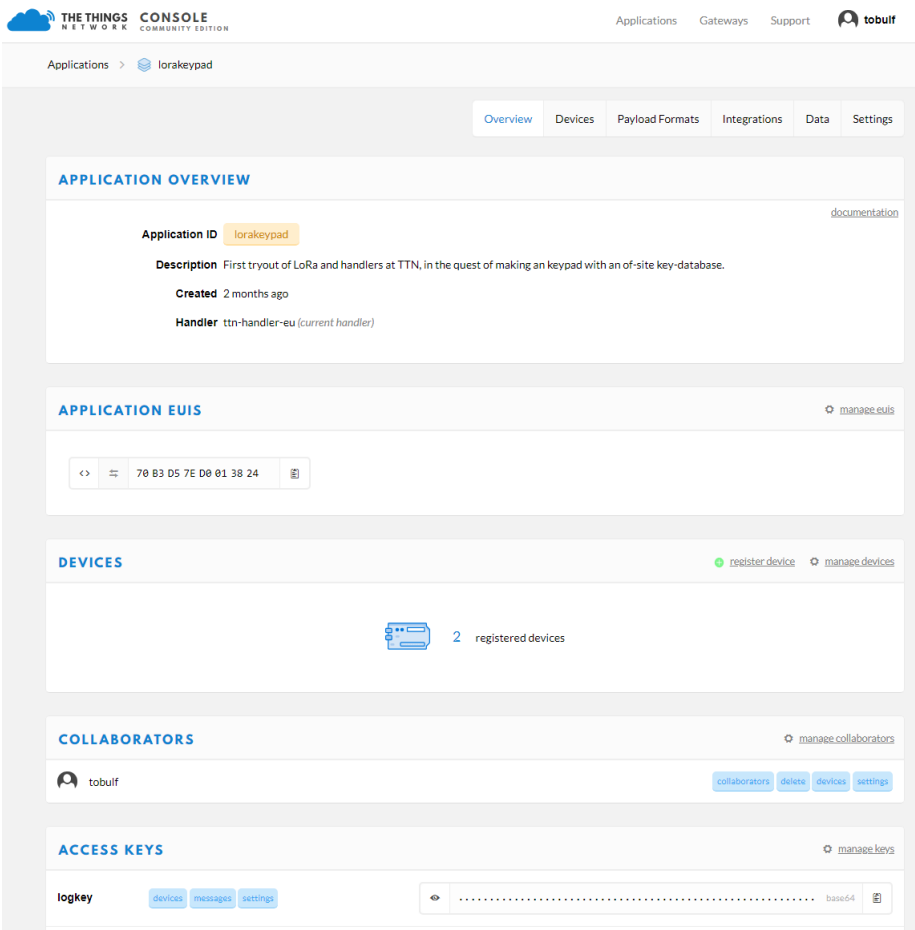

The complete view of the LoRaWAN activity diagram.




| | | |
|---|---|---|
| Title | | |
| Size: A3 | Number | Revision |
| Date: 17.11.2018 | | Sheet of |
| File: C:\Users\..\Schematic.SchDoc | | Drawn By: |



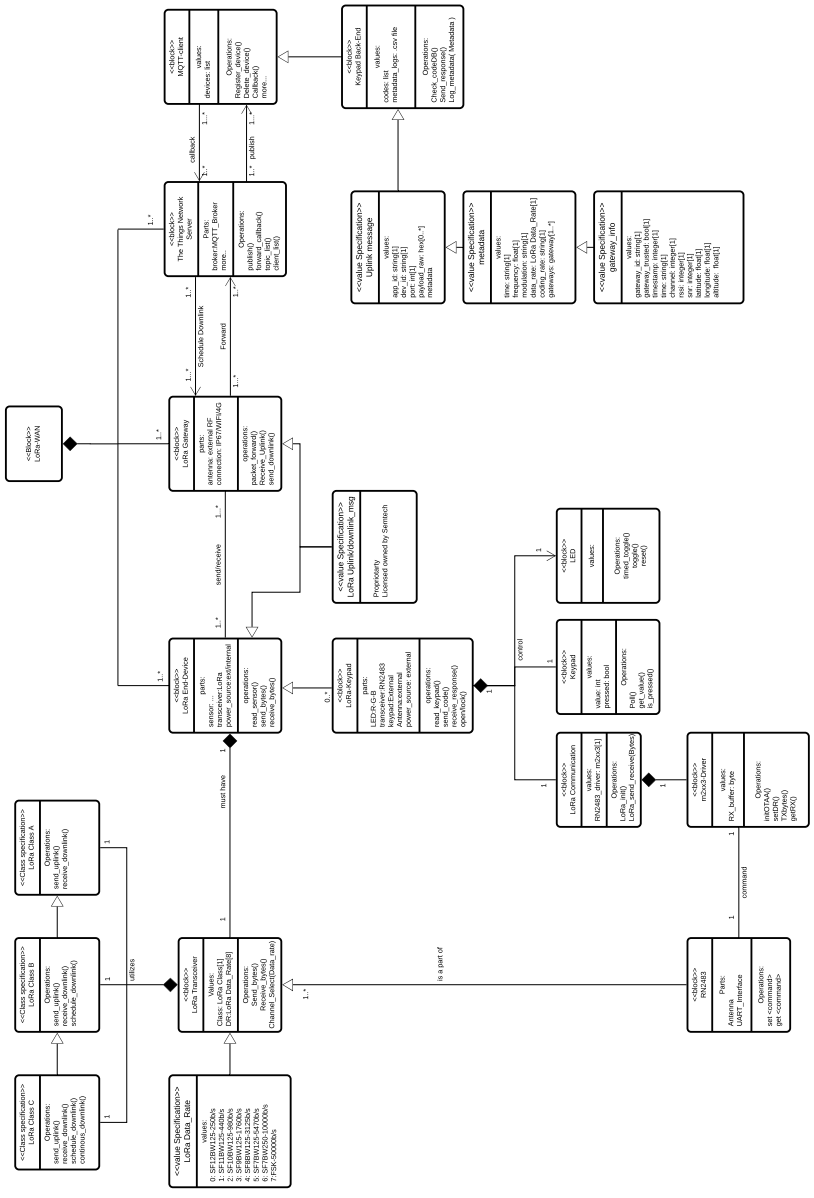

The PCB layout for the LoRa-Keypad.




A screenshot from the TTN console.

| Sequence Num. | Signal Name | Datatype | Transmitter | Recipient | Parameter |
|---|---|---|---|---|---|
| 1 | 4-digit code | 4-Digit Code | User | LoRa-Keypad | Integer |
| 2 | Broadcast Msg | Uplink-Message | LoRa-Keypad | Gateway | Payload[Bytes] |
| 3 | Msg Forward | Uplink-Message | Gateway | TTN | Payload[Bytes] |
| 4 | Publish Msg | UplinkMessage(Object) | TTN | Back-End | Defined: Appendix A |
| 5 | Write Metadata to Log | Meta-data | Back-End | Back-End | Defined: Appendix A |
| 6 | Check code | Internal call | Back-End | Back-End | Boolean |
| 7 | Callback Code valid | Callback-msg(Object) | Back-End | TTN | Payload[bytes] devID[1] |
| 8 | Msg Forward | Downlink Message | TTN | Gateway | Payload[bytes] devID[1] |
| 9 | Downlink Msg | Downlink Message | Gateway | LoRa-Keypad | Payload[Bytes] |
| 10 | Turn on Green LED | Physical | LoRa-Keypad | LoRa-Keypad | IO |
| 11 | Callback Code not valid | Callback-msg | Back-End | TTN | Payload[bytes] devID[1] |
| 12 | Msg Forward | Downlink Message | TTN | Gateway | Payload[bytes] devID[1] |
| 13 | Downlink Msg | Downlink Message | Gateway | LoRa-Keypad | Payload[Bytes] |
| 14 | Turn on Red LED | Physical | LoRa-Keypad | LoRa-Keypad | IO |

Signal list for the sequence diagram in fig. 5.4.




Class diagram showing the entities and their relations throughout the LoRa-Keypad system.