Solar Panels and LoRaWAN

By:
Max Leijtens
s4548027@ru.nl

Supervisor 1:
Erik Poll

Supervisor 2:
Pol van Aubel

June 18, 2018

Abstract

This thesis explores the possibilities of the Long Range Wide Area Network (LoRaWAN) for monitoring solar panels. LoRaWAN is a network similar to GSM except for consuming less energy at the expense of having less bandwidth.

The explored possibilities are mainly focused around whether or not it is possible to use this network for solar panel measurements. Other than the possibility, the pros and cons of using either LoRaWAN or WiFi are discussed in general, but also specifically for this particular application scenario with the solar panels on the roof of the Huygens Building of the Radboud University.

This particular application scenario currently consists of a solar sensor connected to a microprocessor that transmits its data over WiFi to the nearest router, which requires it to be placed close to one because WiFi has a too weak signal for long range connections. This data is then sent from the router to the Particle Cloud, a cloud based IoT platform, which processes it and sends it through to the client's server.

A Proof of Concept made to prove the possibilities of the network is unsuccessful, mainly because the coverage of the network was too small and the hardware used was unsuitable for the job.

In principle however, switching from WiFi to LoRaWAN is perfect to solve all the posed problems that this particular scenario has.

Contents

Terminology

Some of the important terms used in this thesis:[1]

- **LoRaWAN** - Long Range Wide Area Network.
- **End Device, Node** - an object with an embedded LoRaWAN communication device.
- **Network Server** - servers that route messages from End Devices to the right Application, and back.
- **Gateway** - antennas that receive broadcasts from End Devices and send data back to End Devices.
- **Uplink Message** - a message from a Device to an Application
- **Downlink Message** - a message from an Application to a Device or from the Network Server to a Device.
- **Denial of Service (DoS)** - an end-device, gateway or network server no longer responds to requests and communication ceases to function to some degree. (Could be complete or partial.)
- **Over the Air Activation (OTAA)** - The activation protocol used by end-devices to automatically configure their settings to set up communications with a specific network server.
- **Authentication By Personalisation (ABP)** - The manual configuration of end-devices to set up communications with a specific network server.

[1] Partially from: `https://www.TheThingsNetwork.org/docs/lorawan/`

Chapter 1

Introduction

A Low-Power Wide-Area Network (LPWAN) is, as its name shows, a long range network that consumes relatively little power. A few examples of these networks are Sigfox[1], Narrowband-IoT[2] and Long Range Wide Area Network (LoRaWAN) [1]. This thesis only looks at LoRaWAN.

The LoRaWAN is a network with a range of up to 5 km that can be used in similar ways as WiFi or mobile data internet. Even though it has a clear advantage over WiFi when concerning range, its disadvantage is that it allows only quite a low bandwidth.

LoRaWAN has been designed for communication with Internet of Things (IoT) devices; a low bandwidth network with a long range that requires very little power to operate. One of the Dutch LoRa network providers is KPN. On their website[3] they claim that even a simple device, working on two AA-batteries, is able to operate and communicate with a server for up to 10 years. They also claim to have national coverage, both inside and outside of buildings.

The Radboud University has a number of solar panels on the roof of the Huygens Building of which the performance is currently being measured by a solar sensor. This solar sensor measures the strength of the light it senses. It then communicates this data over a WiFi network to a server in which the data is processed. It could be a lot more practical and applicable on a larger scale if this connection would work using LoRaWAN.

Therefore, this thesis investigates whether these communications can be achieved through LoRaWAN and then compares the LoRaWAN-solution to the current WiFi-solution. This thesis is based on the following research question:

> Is it possible to use LoRaWAN for solar sensor measurement communications and what are the pros/cons compared to the current situation?

The research question is answered by creating a *Proof of Concept* (PoC) and by theoretical analysis of the requirements given the LoRaWAN capabilities. In this *PoC*, the solar sensor is modified to use LoRaWAN instead of WiFi. After

[1] `https://www.sigfox.com/`

[2] `https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/`

[3] `https://www.kpn.com/zakelijk/grootzakelijk/internet-of-things/lora-netwerk.htm`

modifying it in this way, the two setups are compared and the pros and cons are concluded from the results.

The Proof of Concept (PoC) is done in several steps. First, the theoretical differences between the WiFi solution and the LoRaWAN solution and their own advantages and disadvantages are discussed in Chapter 4.

Then the PoC begins with a toy example to acquaintance you with the possibilities of LoRaWAN. Finally the possibilities of LoRaWAN are tested to see whether or not it is able to replace the WiFi solution. The PoC can be found in Chapter 5.

1.1 Why LoRaWAN?

Since the solar sensor does not transmit a lot of data and because currently its deployment is limited by the range of the WiFi router, switching to LoRaWAN seems to be an ideal solution. For a broader explanation of why LoRaWAN would be a better solution than WiFi, see Section 3.3. To get an overview of all the differences between WiFi and LoRaWAN, see Chapter 4.

1.2 Document outline

Chapter 2: Background on LoRaWAN. In this chapter the LoRaWAN specification and the most important factors related to this research are discussed.

Chapter 3: Current Setup. This chapter describes the current setup of the solar sensor and reviews the posed problem of the sensor communicating over WiFi.

Chapter 4: Comparison between the LoRaWAN- and the WiFi-solution. In this chapter the LoRaWAN-solution is compared with the current WiFi-solution.

Chapter 5: LoRaWAN Proof of Concept. In this chapter the procedure of building the *PoC* is described and the hardware choices will be explained.

Chapter 6: Future Work. This chapter describes future work that can be done as a continuation of this research project.

Chapter 7: Conclusions. Based on the previous chapters, the final conclusions are described in this chapter.

Chapter 2

Background on LoRaWAN

The long range wide area network, LoRaWAN, the network used for this research, is a form of wireless communication that requires only a low amount of energy. Just like WiFi, this network can be used to transfer data. Unlike WiFi, it can only be used for small amounts of data-transfer. However, it can transfer this data at a much longer range than the average WiFi-device is able to.

In some online sources the terms LoRa and LoRaWAN are used interchangeably. However, they are not the same; LoRa is the Physical Layer (OSI layer 1[1]); the radio waves used for this system. Because whenever LoRa is concerned, the LoRa Radio Frequencies are concerned, these frequencies are also referred to as LoRa RF. LoRaWAN is the media access control (MAC) layer protocol (layer 2); the protocol using these LoRa radio waves. In this thesis these terms are used as described here.

In this chapter the more technical details of LoRaWAN will be discussed. Section 2.1 gives a high level overview of what exactly LoRaWAN is. In Section 2.2 an overview of the used versions of LoRaWAN is given.

Section 2.3 describes the usual architecture of a LoRaWAN network. In Section 2.4 the different end-devices that exist in LoRaWAN are described. Section 2.6 describes the procedure to connect an end-device to an existing network, along with the requirements of setting up a LoRaWAN. Then Section 2.7 is a list of the different communications between network servers and end-devices.

Finally, Section 2.8 describes security vulnerabilities found in LoRaWAN and their relevance to this research.

2.1 High level description of LoRaWAN characteristics

LoRaWAN is a form of Low-Power Wide-Area Network, which means that it consumes only a small amount of energy, but is still able to communicate at long ranges [2] with a gateway. The long range here can reach up to 5 km in urban areas and up to 10 km in suburban areas.

[1] `https://en.wikipedia.org/wiki/OSI_model`

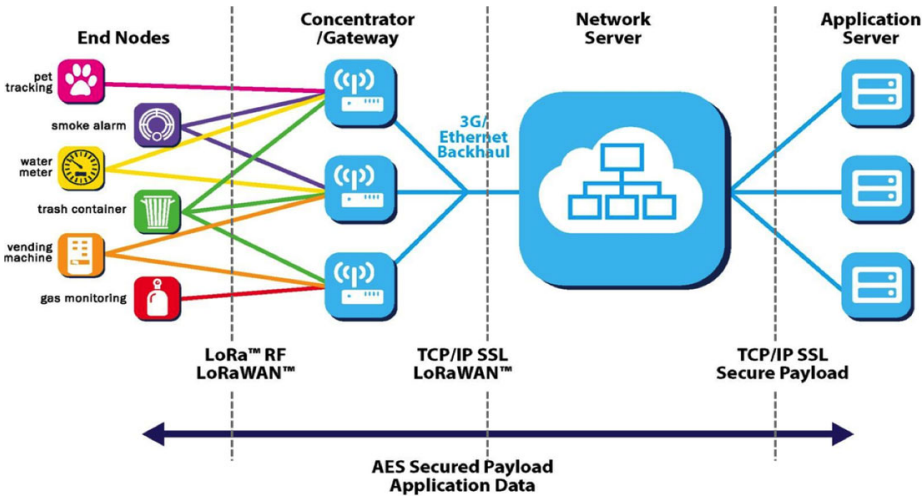


Figure 2.1: An overview of the topology of LoRaWAN. [6]

The low consumption of energy of LoRaWAN allows LoRaWAN devices to be operational for up to 10 years [3]. Because all the gateways of a network are connected to only one network server, devices using LoRaWAN are able to operate on a regional, national or global scale.

For the communication with end-devices, LoRaWAN specifies a LoRa Radio Frequency network. This specification allows a number of frequencies, with 868MHz being common in Europe and 915MHz in North America [4]. One major downside to LoRaWAN is that it only allows low data rates of at most 0.5 kbps[5].

2.2 Version details

The LoRaWAN details in this document are based on the LoRaWAN specification version 1.1 [1] and Regional Parameters version 1.1 [4], This is currently the most recent version of the LoRaWAN specification. This version is backwards compatible with version 1.0 and is almost the same, except that in version 1.1 some security flaws that would have impacted this thesis have been fixed, as discussed in Section 2.8.

2.3 Architecture

A LoRaWAN setup consists of *end-devices* (a.k.a. end nodes), *gateways, a network server* and usually one or more *application servers*. See Figure 2.1 for an overview.

The end-goal of the network is to have an end-device communicate with an application. This happens in several steps. First, an end-device uses the LoRa Radio Frequency to communicate with one or more gateways. Then a gateway creates a virtual bridge between the LoRa Radio Frequency on one end and a

standard IP connection (over Ethernet) with the network server on the other end.

The Network Server has two tasks, on one hand it processes the end-device data in such a way that it can be provided to the application servers which can relay it to their applications for further processing. On the other hand it controls the network.

This network control is important because most end-devices do not send their data specifically to a certain gateway, but instead they broadcasts their uplink messages. These broadcasts cause the messages to be able to be received and relayed by multiple gateways. Here the network server filters out the duplicated network packets [6]. Based on certain criteria of the latest received uplink messages (like signal strength), the network server selects a gateway that is best for downlink messages.

Even though downlink messages are possible, they are sent with a much higher latency than uplink messages because there is less bandwidth available for downlink. This causes the downlink messages to be used less than uplink messages.

2.4 End Devices

The LoRaWAN specification [1] defines three device types (Class A, B and C). The difference between these types is a trade-off between battery lifetime and up- and downlink communication options. All devices have to implement Class A, whereas both Class B and C are extensions to the Class A specification. Every device starts out as a Class A device, but can be reconfigured at any time by the application to become a Class B or Class C device. This reconfiguration can also be undone at any time.

2.4.1 Class A Device

Class A Devices support bi-directional communication between a device and a gateway. This class is optimised for battery-powered end-devices and is the Class that uses the lowest amount of power. Class A has to be supported by all devices on the LoRaWAN and every device starts their communications as a Class A end-device.

Class A end-devices can send uplink messages at any time (unscheduled). After the uplink communications end, the device opens two receive windows at regionally specified times, most often at one and two seconds after the ending of the transmission. During these receive windows, the server can open downlink communications (which are allowed to last longer than the time the receive windows stay open). The receive windows only dictate the time during which a downlink transmission can be opened. A server can use either the first or the second timeslot to start its communication, but not both during the same downlink.

If the server does not use either of these two time frames, the next opportunity for the server to communicate with the end-device will be after the next uplink transmission from the device.

2.4.2 Class B Device

Class B Devices extend Class A by adding a scheduled receive window for synchronised downlink transmissions (receive windows) from the server. The advantage of these these time-synchronised receive windows is that the end-device is ready for reception on a predictable and specific time, rather than depending on non-deterministic downlink windows available in the default Class A specification. These added synchronised receive windows are created *in addition* to those created in the Class A operations. This extra scheduling and these extra receiving windows require more power than Class A, but Class B is still optimised enough to be used in battery-powered end-devices.

The scheduling in this operation mode happens every (regionally specified) Beacon Period (usually between 2 and 3 minutes). At the start of this period, all Class B devices receive a beacon frame (transmitted by the gateways) which contains a specific receive window time during which the server is able to send communications to all Class B devices in the area of that gateway, but is also able to send specific listen times to a certain device.

2.4.3 Class C Device

Class C Devices extend Class A by keeping the receive windows open unless they are transmitting. This allows for low-latency communication, but is many times more energy consuming than both Class A and B devices.

This Class provides the maximum amount of send/receive possibilities; the network server can initiate downlink communication at any time as long as the end-device is not already transmitting. However, having a constant communication window costs more energy compared to Classes A and B and is therefore only used by devices that have sufficient power available.

2.5 LoRaWAN Environments

There are several LoRaWAN environment choices available, either you can set up your own network, or ask permission to use an already existing network. Examples of already existing networks in the Netherlands are KPN-Things or the open initiative The Things Network (TTN)[2]. KPN-Things is a commercial LoRaWAN network that claims to have national coverage both inside and outside of buildings in the Netherlands.

The Things Network is an open network to which everyone can add their own gateway and is free of charge to use. The downside of this network is that it does not have any guarantees about its coverage.

The final option is to set up your own network, this would, depending on the area you wish to cover, involve much larger costs, as explained in Section 4.1.2.

2.6 LoRaWAN Setup

To get the LoRaWAN environment working, some steps need to be taken before the network is operational. These steps can be seen in Figure 2.2. First you

[2] `https://www.TheThingsNetwork.org/`

would need to deploy a Network Server, which also has to be configured. Then you would need to deploy and configure one or – most likely – more gateways. Only then can you start adding end-devices.

Since we will make use of already existing networks, namely The Things Network and KPN-Things, the Networks Server and Gateways have already been deployed and configured and the end-devices can immediately start being deployed.

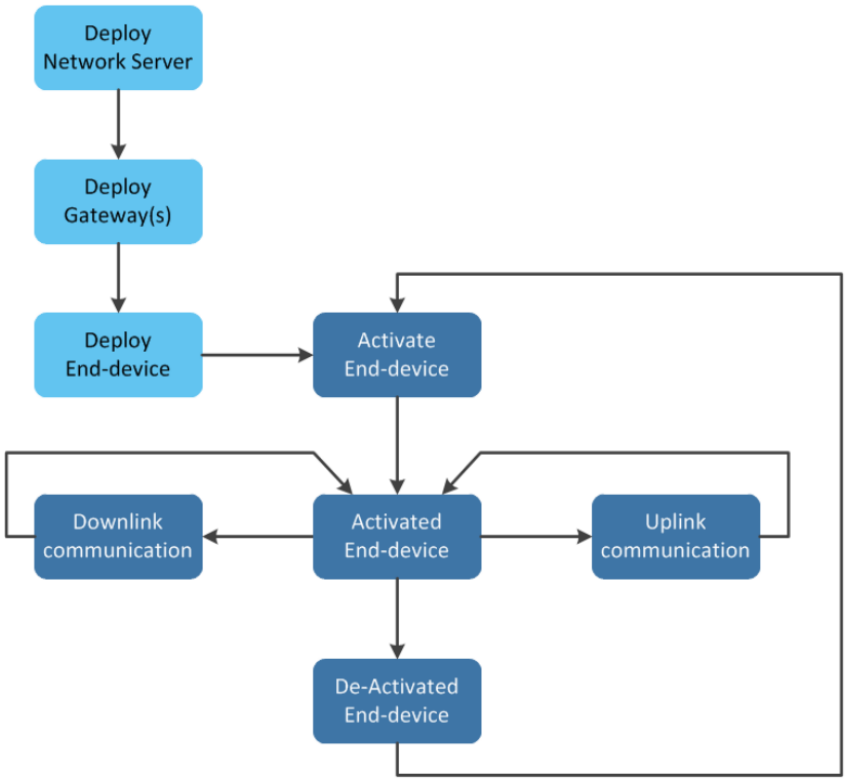


Figure 2.2: A high level overview of LoRaWAN lifecycle. [6]

Once you deploy an end-device, it has to be activated within a certain LoRaWAN environment. The activation process is described in Section 2.7. Once activated its up- and downlink functionality can be used.

These functionalities can be used until the device is de-activated. De-activation can occur when the device stops being synchronised with the network server in respect to security keys and other configuration settings. This can happen for example after power loss on the end-device, or by turning the end-device off for a while. Re-activation is simply a matter of running the activation procedure again.

2.7 LoRaWAN Communication

LoRaWAN specifies several different types of communication between network server and end-devices, each with a specific use-case in mind [6].

Activating an End-Device As shown in Figure 2.2, an end-device needs to be activated before it is able to send or receive data. There are two ways to activate an end-device, either by manually configuring the network details, called Authentication By Personalisation (ABP), or by allowing for Over-the-Air-Activation ($OTAA$), in which after selecting a network, the rest of the configuration is done automatically.

The LoRaWAN specification defines two messages for $OTAA$. A Join-Request, which requests the network server to be allowed on the network and a Join-Accept, which is send by the network server when accepting the end-device on the network. This second message also contains the end-device configuration details.

Sending data The reason this network exists is to be able to transfer data. This is possible, both as an uplink transmission (end-device to network server) and as a downlink transmission (network server to end-device).

Multiple End-Device communication It is also possible to send downlink communication to multiple end-devices at once if required. This is only possible if a group of end-devices is assigned the same device address.

Media Access Control commands These commands can be used to manage the network. They can be used to change communication speeds, frequencies, request a device's status, etcetera.

Beaconing Finally it is possible to generate and broadcast time-synchronisation beacons used by Class B devices. These beacons contain timing references and optionally gateway information.

2.8 Security vulnerabilities of LoRaWAN

Like most network protocols, LoRaWAN also has some vulnerabilities [6]. The most relevant works about vulnerabilities in the network are from Avoine, G. [7], Zulian, S.[8] and Yang, X.[9]. However, these vulnerabilities have either been patched in LoRaWAN version 1.1 or are not relevant for this study.

2.8.1 Unauthorised activation of end-devices

Avoine [7] published a work on the LoRaWAN specification v1.0. In this work a combination of vulnerabilities is discussed, namely: the enforcement of re-using sessions keys, the replay of Join-Request/Join-Accept messages, and the misuse of shared root keys. The combination of these vulnerabilities could be used by a third party to activate end-devices. However, since this vulnerability is mitigated in the LoRaWAN specification v1.1 [1], which is the version used in this project, it will not be discussed further.

2.8.2 Random numbers

In 2016, Zulian [8] published an article describing weaknesses in the Random Number Generator of the hardware that is used in most LoRaWAN hardware. Furthermore Zulian describes an issue related to nonce generation in the Over the Air Activation, which allowed for impersonation of servers when setting up a new device. However, just like Avoine's vulnerabilities, these issues have been mitigated in v1.1.

2.8.3 Beaconing vulnerability

Another article about LoRaWAN vulnerabilities, published in July 2017, made by Yang [9], found two new weaknesses. The first is an ACK spoofing attack, this is possible because there is no indication which message is confirmed. This weakness has again been mitigated in v1.1.

The second vulnerability is to eavesdrop beaconing frames that are broadcast by gateways and used by Class B end-devices to calculate ping slots for downlink traffic. This causes the end-devices to calculate a wrong beacon slot, which causes a Denial of Service (DOS) for traffic from the server to the end-device.

This vulnerability is *not* mitigated in v1.1 of the specification. However, this attack is of little importance in this study for two reasons. First of all, there are currently no plans for downlink traffic that could not be transmitted during the Class A downlink timeslot in this study. Secondly, the hardware used in this project only allows for Class A and Class C, which leaves this attack vector useless.

2.8.4 Complete Denial of Service

Finally Van Es [6] found a vulnerability in v1.1 that allows for a complete Denial of Service of a certain end-device. In it, an attacker eavesdrops and then replays an older network frame during the end-device registration process, which causes the end-device to calculate a wrong security context, which in turn causes this specific end-device to be unable to communicate with the server in any way.

Currently the only way to avert this vulnerability is to manually check whether after Over the Air Activation the device is transmitting data to the server. If it is not, the OTAA has to be restarted, which, if it completes successfully, will restore communications. This is the only vulnerability that is currently still in existence but poses no real risk to this Proof of Concept.

Chapter 3

Scenario outline

ReRa Solutions[1] has developed a solar sensor that measures the light intensity across a broad spectrum.

These sensors are deployed near solar panels to be able to determine the efficiency of these panels and detect possible defects. This is done by having the sensor measure the amount of light hitting it (and the solar panels), then by calculating the theoretical amount of energy that solar panels should produce under the measured circumstances and then comparing this theoretical energy yield to the actual energy yield.

Once enough sensors have been deployed over a large enough surface area, (for instance, evenly spread across a country) these sensors could also aid in making more accurate weather predictions.

Unfortunately, the sensor currently transmits its collected data over WiFi which limits its deployability, as further explained in Section 3.1. Therefore ReRa Solutions wants to find an alternative to WiFi that is more flexible but still allows for their data to be transmitted to their server.

3.1 Limitations of the current setup

ReRa Solutions is looking to expand the deployment of the sensor throughout the country. This would allow them to do more with the data collected by each sensor.

That in itself is not a problem. Unfortunately, these sensors currently communicate their data through WiFi. Safely using a WiFi connection requires a router protected with a password which has to be configured in the sensor manually and if this password happens to change over time, it has to be manually reconfigured in each sensor.

Another problem that arises by having the system work over WiFi is that these routers have to be placed near the sensor to be able to be connected to it, since WiFi has a very limited range. On some occasions it might even be required to adapt the structure where the sensor is placed at to be able to get a router close enough. Both of these points could cause extra costs, which makes the sensor less attractive for companies to invest in.

[1] `https://www.rerasolutions.com/`

Figure 3.1: The ReRa Solutions solar sensor installed next to a solar panel. Photo by ReRa Solutions.

Because of the problems arising when using WiFi, ReRa Solutions wants an alternative to WiFi.

3.2 Solution requirements

Below are the requirements set by ReRa Solutions for a solution for their WiFi sensor problem.

3.2.1 Functional requirements

What ReRa Solutions wants to achieve is their solar sensor working in such a way that it can still transmit its data, at least once every 10 minutes to their server (for which network latency is of less importance).

The data consists of a few variables with a total size of eight floats (32 bytes). This data size should be allowed to grow to 40 bytes because hooking up a smart meter to the sensor would add two floats of data. The specifics of the smart meter is discussed in more detail below in Section 3.3.

3.2.2 Operational requirements

Furthermore they want this to be done as simple as possible so that sensors are still at least as easy to deploy as they are in the current WiFi-solution. The new

solution would have to be operable without human-intervention for at least as long as the current solution, which is at least 3 years, and has to cost roughly as much as the current solution.

3.2.3 Security and privacy

At the moment encryption of the data is not yet required. The data does not yet contain anything sensitive; just the intensity of the light at the sensor's location. However, in the future ReRa Solutions might want to use power measurements from the Dutch Smart Meter, which offers detailed insight in power usage and generation of a household[2]. The data from these smart meters however is sensitive and from that point on, the transferred data should be encrypted. The non-WiFi solution should have the possibility to switch to encrypted transmission, or have encrypted transmissions from the start.

Other than encryption, two other security aspects are important for the sensor; data integrity and device authentication. Data integrity means that the sent data has not been modified between sender and recipient. If the data is changed, it will be noticed. Device authenticity is assurance that the data is sent from the device it claims to be from. This prevents false users/devices sending fake data to the server.

3.3 Why LoRaWAN would be a better solution

If the sensor could use LoRaWAN instead of WiFi, most of the current problems described above could be fixed, while also meeting all the requirements that ReRa Solutions has set. Most of these points are further explored in Chapter 4.

3.3.1 Fulfillment of the functional requirements

Normally, one of the downsides of using LoRaWAN is that it does not allow for large amounts of data to be transmitted at once (up to a maximum of 51 bytes per packet) and not very often (only one packet per two to ten minutes). However, this should not pose a problem as the sensor only sends only 32 to 40 bytes per packet and only sends one packet every 10 minutes.

3.3.2 Operational fulfillment

It is easy to deploy the LoRaWAN-sensor on a large scale without the need of extra equipment since all settings (including network keys) are coded into the sensors and automatically change if network settings happen to change. Because transmitting data through LoRaWAN consumes very little energy, the sensor should be operable for at least 5 years without user intervention, but probably even longer. Also, the Netherlands (the country where ReRa Solutions is located) already has a national scale LoRaWAN network available. The sensor could immediately be deployed anywhere in the country without having extra costs in hardware.

[2] `https://en.wikipedia.org/wiki/Smart_meter`

3.3.3 Security and privacy

Encryption of the data is done using AES and data integrity is achieved through CMAC and is automatically enforced in LoRaWAN. AES is also used for authentication as every device receives its own key which automatically authenticates it whenever the device transmits data to the network. For more information on the security aspect of LoRaWAN, see Section 4.3.

Chapter 4

Comparison between the LoRaWAN- and the WiFi-solution

4.1 Comparison LoRaWAN and WiFi in general

In this section the differences between LoRaWAN and WiFi in general are compared.

4.1.1 Range and connectivity

LoRaWAN has a much longer range, but has for instance no connectivity in Faraday cage-like buildings, like the Huygens building. It also does not have good connectivity in buildings made out of reinforced concrete. This is the case for most electromagnetic waves, but especially for low energy or large (long range) waves, like the ones LoRaWAN uses [10].

Even though WiFi has a much shorter range, it does allow for a lot more data transferred in a certain timespan. Where LoRaWAN has a maximum data-transfer of 5 kbps (if used at maximum power) [5], WiFi allows for data transfers of up to 866 Mbps [11]. (Which is more than 160 thousand times more data per second!)

4.1.2 Energy consumption and material costs

Consumption and costs of the network

A WiFi-router and a LoRaWAN gateway consume about the same amount of power. However, you get about five kilometer coverage with one gateway, whereas a router gives only coverage in a 15 meter radius, so you would need fewer gateways than routers to cover the same area, which would mean that in the end LoRaWAN gateways would consume less energy [3].

The cost of a router however, is much smaller. Where a gateway costs about €300 [1], a router costs only about €60. That means that you could place several

[1] `https://shop.TheThingsNetwork.com/index.php/product/the-things-gateway/`

routers for the price of only one gateway. However, a gateway can be made by hand and requires very few materials, and costs therefore only about €60. [2]

It should be noted that with the LoRaWAN solution it is possible to use an already existing network and you are not required to purchase or set up any gateways. With WiFi it is sometimes possible to use an existent network, but because of the limited range, this would limit the use and placement options of the sensor, so it might be required to install extra routers.

Consumption and costs of the end-device

If you compare the prices of a microprocessor with a WiFi antenna and a microprocessor with a LoRa antenna, you find practically the same price for both, about €40.

Finally, power-consumption by both of these microprocessors. Mahmoud showed in a study published in 2016 [3] that a microprocessor that works with WiFi requires a constant power supply and would not be able to work with an AAA-battery, whereas a Class A or B LoRaWAN microprocessor would.

4.2 Settings and passwords

With a WiFi-router, every router has different settings and a different password to connect to it. In the current solution, this password has to be set manually for every sensor for every router it connects to. If settings or passwords change, the sensor loses connection and has to manually be reconnected.

With a LoRaWAN solution you just have to select a network and the network settings are automatically configured with Over the Air Activation. When settings change, these are automatically updated on every device connected to the network. This gives more ease of use and ease of deployment with the LoRaWAN-solution.

4.3 Security Analysis

To assess the security of these networks, three subjects are taken into account: Confidentiality, Integrity and Authenticity. Here confidentiality means that the sent data can only be read by the sender and the intended recipient(s). Integrity guarantees that the sent data has not been modified between sender and recipient. If the data is altered, it will be noticed. Authenticity is assurance that the data is sent from the source it claims to be from. This involves some proof of identity.

4.3.1 Required security

Confidentiality

The current version of the sensor only transmits the intensity of the light hitting it. Since this is not sensitive data, confidentiality is not yet an issue. However,

[2] A tutorial on how to build a gateway yourself. `http://cpham.perso.univ-pau.fr/LORA/RPIgateway.html`

in the future the sensor might be connected to a smart meter to help compare the solar panel generated energy with the energy consumption of the household. Once that happens confidentiality has to be incorporated in the solution and is therefore a requirement.

Integrity

This is the most important factor when looking at security in this system. The data collected by the solar sensor is useless if it is not completely intact when it is received by ReRa Solutions' server. Because it is this important, ReRa Solutions already has an integrity test built into the data transmitted by the sensor that allows the server to verify whether the received data has not been altered by the transmission.

Authenticity

It is important that only the deployed sensors are able to transmit data to the server and for the server to know which sensor sent what data. Third parties should be unable to insert their own fake data into these transmissions.

4.3.2 Provided security

In this section the security provided by both the current solution as the LoRaWAN solution are discussed and compared. A description of the current solution can be found in Chapter 3.

Between sensor and the host server

Both solutions use the Advanced Encryption Standard (AES) specification [12] for encryption purposes. The AES specification is a symmetric-key algorithm, which means that the same key is used for both encryption and decryption. The specification is used to ensure confidentiality in the transmission of the data. Integrity is achieved for both solutions by using an AES based Cipher-based Message Authentication Code (CMAC) [13].

In the current solution a session key is used whenever a connection is made to the Particle Cloud. This key is safely transmitted from the server to the sensor using RSA public/private keys [14] that are set in the factory. In the LoRaWAN solution the symmetric key is either coded into the software (when using ABP) or set through the OTAA protocol when connecting the sensor to the network.

Authenticity is achieved in the current solution through the same factory set RSA keys that set up the AES communication. In the LoRaWAN solution authenticity is achieved through a device ID which is hardcoded and uses when communicating with the server. These keys or this device ID could only be acquired by a third party through physical access to the sensor.

Between host server and application server

Current Solution The current solution uses OAuth [15] to authenticate data access. When setting up a server for an application, it receives a secret key from the Particle Cloud which will authenticate the server. This key will also be used when authenticating users login in on the server and is also used to secure

the communication between the host and the application server. (In this case between Particle Cloud and ReRa Solutions.) OAuth ensures all three security requirements are met.

LoRaWAN solution In the LoRaWAN solution the data is immediately sent from the host server (KPN-Things) to the application server through a (TLS secured) POST-message. The problem is that while using the free developer version, KPN still has access to the AES-keys. This means that KPN is effectively a Man-in-the-Middle at the Network Server, which breaks all three security requirements.

However, in the paid enterprise version, the version the solution will be in once it is taken into production, the keys remain under your own control, providing end-to-end security between end-device and application server.

Chapter 5

LoRaWAN Proof of Concept

In this chapter an attempt at a Proof of Concept is described for ReRa Solutions' solar sensor to work with LoRaWAN instead of WiFi. However, since the proof is not completed, it also describes where things went wrong and what changes should be made in future attempts at a Proof of Concept of this solution.

At the end of this chapter (Section 5.2) the motivation to choose the Seeeduino LoRaWAN for this study is described, along with its specification, the motivation why this device was picked, what alternatives there are to this device and why picking a different device is recommended for future attempts.

5.1 Toy Example

The full log of the taken steps can be found in Appendix B.
The goal here was to build a simple "Hello Word"-like program using LoRaWAN. First by getting the microprocessor to work, then by having it send some data to a server or another device.

5.1.1 Hello world locally

To be able to communicate with the Seeeduino board you are required to install the Arduino IDE and install additional boards, namely the Seeeduino SAMD board. Then you can connect the board over micro-USB to your computer and use example code available in the IDE to have the board transmit "Hello World!" over the cable. You can check in the Serial Monitor of the IDE whether you can actually receive this.

Encountered problems

Not a member of the proper group To be able to communicate with the Seeeduino via the USB-port, the user first has to either have enough rights, or be added to the correct group. The groups used for the communication are

`dialout` and `uucp`. A user can be added to a group through
`usermod -a -G examplegroup username`

Service blocking communication When you are using Ubuntu to program your Seeeduino, you will notice that after taking the required steps you are still not able to communicate with the board. This is because Ubuntu has the service ModemManager installed by default.

ModemManager is a service made to communicate with mobile broadband cards (e.g. GSM through SIM-cards). This means that whenever a serial device is connected, this service will attempt to connect to a mobile network as if it were a SIM-card. This causes the Seeeduino to be continuously blocked by the data transmitted to it by ModemManager, which effectively disables the board for personal use. To prevent this service from interfering with the Seeeduino, execute the following command:
`systemctl disable --now ModemManager.service`

5.1.2 Data over LoRaWAN

To send data over LoRaWAN, you first have to select a network that supports your data. In this PoC we have used both the publicly available The Things Network[1] as well as the commercial KPN-Things network[2]. After registering at one of these networks, you can create an application and request keys for it. Then you have two options to connect to the network. Either through Authentication By Personalisation (ABP) or through Over The Air Authentication ($OTAA$).

When using ABP you manually enter the keys, the network identifier and application identifier. The Seeeduino will then try to connect to the network using these credentials.

On the other hand, if you use OTAA, you only need to enter the network and application identifier and the network and the board will automatically negotiate a key. This is especially useful when you want to connect several devices at once using the same code.

Encountered problems

Too little coverage by TTN When actually attempting to connect using the Seeeduino, we were unable to get a connection running through TTN. After we applied optimisations to the code (Appendix B.5) we could still not even find the network. We have tested the board both in- and outside and after checking the coverage of TTN, we concluded that there is simply not enough coverage in the city we were working from (Nijmegen). This is why we decided to switch to KPN-Things.

No coverage inside The second problem we encountered when attempting to connect was that even the stronger KPN-Things network with its better coverage could not connect to our Seeeduino while we were inside. If you want to connect or transmit data, you need to be outside.

[1] `https://www.TheThingsNetwork.org/`

[2] `https://www.kpn.com/zakelijk/grootzakelijk/internet-of-things/lora-netwerk.htm`

OTAA not working The final problem we encountered was that even with the KPN network, while being outside, the Seeeduino was unable to connect to the network over OTAA. You have to manually enter the keys and connect through ABP. Instructions on how to do this can be found in Appendix B.4.

5.2 Device Choice

For this attempted Proof of Concept, we used the Seeeduino LoRaWAN. Below is a description of the device specification and costs, a motivation for why this device was chosen and what alternative devices could be used.

5.2.1 Specifications and costs

The Seeeduino LoRaWAN can be purchased from several webshops for about €40. We purchased it online at Kiwi Electronics[3].
These are the Seeeduino LoRaWAN's specifications:

Arduino/Processor

Microcontroller	ATSAMD21G18, 32-Bit ARM Cortex M0+
Operating Voltage	3.3V
Digital I/O Pins	20
PWM Pins	All but pins 2 and 7
UART	2 (Native and Programming)
Analog Input Pins	6, 12-bit ADC channels
Analog Output Pins	1, 10-bit DAC
External Interrupts	All pins except pin 4
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
EEPROM	None
Clock Speed	48 MHz
Lenght	68 mm
Width	53 mm
Weight	19.6g(without GPS), 19.9(with GPS)

Table 5.1: Specifications of the Ardruino and the Processor

LoRaWAN Module RHF76-052

- 1.45uA sleep current in WOR mode
- High link budget of 160dB. -140dBm sensitivity and 19dBm Output power.
- Dual band, 434/470MHz and 868/915MHz
- 19dBm@434MHz/470MHz
- 14dBm@868MHz/915MHz

[3] `https://www.kiwi-electronics.nl/seeeduino-lorawan`

- Support LoRaWAN protocol, Class A/C
- Upgradeable firmware
- Small size: 23mm X 28mm with 33 pin SMT package

5.2.2 Motivation for this device

We chose the Seeeduino LoRaWAN because we required an Arduino that had a LoRaWAN module built in, preferably with the option to become a Class C device in case we would want to implement over-the-air software updates.

The advantage that the Seeeduino has over some other Arduinos is the option to use both the 434MHz as the 868MHz network. Because at the time of purchase we did not yet know what network we were going to use, this seemed to be the safest choice.

Seeeduino antenna too weak While conducting the research we noticed that the Seeeduino has trouble transmitting data to a gateway and even more trouble receiving data. Therefore we suggest that in future research a different board is used.

5.2.3 Alternative device

As an alternative to the Seeeduino LoRaWAN, we suggest the Sodaq One, available at: `https://shop.sodaq.com/sodaq-one-eu-rn2483-v3.html`. Even though it is more expensive than the Seeeduino (€95 instead of only €45), it has a LoRa chip of a different manufacturer which might make a difference for its range.

Furthermore, this one has an equally strong processor and a built in accelerometer, which is able to detect when it moves. This could be useful when automatically mapping the sensor based on GPS information, for instance the location could be automatically updated once it stops moving again.

There are not many other complete LoRaWAN devices, you have much more choice when browsing solely for LoRa modules, which you would have to manually plug onto your own (Arduino) chipset.

5.2.4 Alternative power usage

During our research we powered the board with the power coming from the micro-USB cable. However, when actually deploying the board, an alternative method to supply power is required. Below are some suggestions.

Connected to a constant power supply Without having to change anything to the current design, the board could be powered with mains electricity. This could be achieved by plugging the micro-USB into an adapter and plugging this adapter into an outlet.

This method is however, not the preferred one as it requires an nearby outlet, which is both unlikely and unpractical to have outside.

Battery Another relatively easy to implement option is hooking the board up to a battery. Because of the low power consumption, the board should be able to last for about ten years on one battery.

Through solar panel(s) Since the sensor is made to detect the effectiveness of solar panels, the sensor will always be in the close proximity of panels. Therefore the sensor could theoretically be hooked up to the power generated by these panels. Practically this was possible and has been tested, however because of the high voltage power produced by the panels (up to 1000V), this option would require certifying the device under applicable norms, which is expensive.

Through its own solar panel (with a chargeable battery) Hooking it up to a full sized solar sensor was not possible, but as an alternative power source, it could be hooked up to a panel specially designed to exactly provide the board's required power. Other than that the panel stores its excess power in a chargeable battery which would allow it to remain powered during the night or when it does not receive enough sunlight for an extended period.

The board suggested in Section 5.2.3 also has a built-in solar charge controller that allows for just this when you plug a solar-cell into it.

Chapter 6

Future Work

This chapter lists further research ideas and possible continuations of this Proof of Concept. Section 6.1 lists the steps that still need to be taken to complete the Proof of Concept. Section 6.2 contains possibilities of continuing with the sensor after the proof is completed. In Section 6.3 some other ideas are suggested for LoRaWAN related research and Section 6.4 describes other Low Power Networks that could be used instead of LoRaWAN.

6.1 Completing the Proof of Concept sensor

Because the Proof of Concept has not been completed, a future thesis could get better hardware, then quickly move through the problems encountered during this research and continue to connect the Arduino to ReRa Solutions' server. Once the server is able to process data sent by the Arduino, the Arduino can be connected to the sensor so that it will be able to send actual data over LoRaWAN .

6.2 Continuation of this project

Further continuations of the Proof of Concept are listed here.

6.2.1 Build a gateway

During the research a connection issue surfaced several times. The main reason that this problem occurred was that the antenna on the Seeeduino is not sensitive enough to connect to a network while it's inside a building. This problem could be solved by building your own gateway, placing it inside the building and connecting it to a network. That way you would have more coverage nearby which would help in the development process.

6.2.2 Self sustaining sensor

Currently the sensor is still battery-powered. This battery has to be replaced every five to ten years. Ideally the sensor would be self-sustainable, meaning that it would generate its own energy. This could be achieved by connecting a solar

panel and a solar power controller, along with a chargeable battery to the sensor. In Section 5.2.3 a suggestion is given for an Arduino that has a solar power controller built in. This controller automatically switches between charging and depleting the battery, depending on the amount of energy generated from the solar panel.

6.2.3 Energy consumption

One of the advantages of LoRaWAN is that is consumes very little power while still being able to transmit data at very long ranges. But how much less energy does the LoRaWAN-based sensor consume in comparison to the WiFi-based sensor?

6.3 Other Ideas

During this research, several unanswered questions have been raised, each one of them a good research subject. Below some alternative research subjects related to these questions.

Downstream side of LoRaWAN and software updates In this setup, LoRaWAN is mainly used to upload the collected data from the solar panels to the server. If the downstream transmission side of LoRaWAN would also be used, the sensors could possibly receive software updates while remaining deployed.

LoRaWAN network coverage During the research we noticed that the advertised coverage, especially inside was not realised, even though several sources claim that the range of LoRaWAN should be five to ten kilometers in urban areas. Research could be done on how long the range and how strong the coverage of LoRaWAN gateways actually is.

6.4 Alternatives to LoRaWAN

There exist several long range, low power networks (LPWAN). LoRaWAN is only one of these. The most used LPWANs other than LoRaWAN are NB-IoT and Sigfox. Both implementations use a different chipset and different network protocols. These networks could be compared to find the optimal LPWAN for the solar sensor.

Chapter 7

Conclusions

7.1 Global conclusion

LoRaWAN is very well-suited as a replacement for WiFi for ReRa Solutions' solar sensor. This is because the device sends only few small data packets per hour, which is well within the capabilities of LoRaWAN, as explained in more detail below.

A full description of ReRa Solutions requirements and how LoRaWAN fulfills these can be found in Section 3.1 and Section 3.3.

7.2 Pros and cons of LoRaWAN

In this section both the pros and the cons of LoRaWAN are explained, both in general as well as for ReRa Solutions in particular.

Pros:

- The data-rates allowed by LoRaWAN work great with the requirements of the IoT, like the solar sensor developed by ReRa Solutions; most implementations do not require to send more than 51 bytes at once. (51 bytes being the maximum amount of data per packet.)
- A LoRaWAN device consumes very little energy, which allows these devices to be able to transmit data for several years on only one battery.
- LoRaWAN allows for automatic activation of devices over the air which allows for easy bulk deployments.

Cons:

- LoRaWAN only allows for little data being sent, e.g. data from sensors or GPS and only occasionally. Streaming data or browsing internet is not an option.
- LoRaWAN does not work inside buildings because the low-energy signals are too weak to penetrate reinforced concrete. This would either require the device to be outside, or to have a separate gateway inside. This is

not a problem for ReRa Solutions as their sensors will only be deployed outside.

- The advertised network coverage of the free to use The Things Network is not realised. The Things Network also does not have national coverage which makes it unsuitable for commercial goals.

7.3 Technical conclusions about using LoRaWAN for the ReRa Solutions solar sensor

In this section you can find the more technical conclusions that could be drawn from making the Proof of Concept. These conclusions should also be taken into account when doing future research on either LoRaWAN or ReRa Solutions' solar sensor.

- The antenna and radiochip built into the Seeeduino are too weak for the desired functionality. They do not receive any signal while being inside of buildings and only receive a weak signal when being outside, even when being in an area that has been tested to have a strong signal from the gateway. A different chipset is required to achieve the desired connectivity. A suggestion for a different chipset can be found in Section 5.2.3.

- The Things Network, the network used for the larger part of this thesis has a too unreliable coverage to be used commercially. Since the network is set up and maintained by its users, some areas will have proper coverage, but larger areas will not have any coverage, or too little to properly operate your LoRaWAN devices. This is explained in more detail in Section 5.1.2.

- For the current implementation of the sensor, which requires only uplink traffic, Class A microprocessors seem to work best as these require the least amount of power.
  However, if in the future ReRa Solutions would want to implement periodical software updates for the sensor, the sensor would require a microprocessor with Class C functionality (as the chosen microprocessor allows) so that for these updates it can temporarily switch to Class C because it allows for the largest data-rates. This would cause the battery to drain faster, so this would only be useful for the self sustaining sensors or sensors on a constant power supply. More information about the different device classes can be found in Section 2.4.

- Class B functionality is not required for this sensor since Class B is mainly useful for devices that are not regularly transmitting data to the server. Class B is used to allow the network server to communicate with these devices when they are rarely transmitting (and would in Class A therefore rarely be receiving). Since the sensor transmits data at a set interval, there should be enough time frames for the server to transmit data to the device, without the use of beacon frames.

7.4 Practical problems with the PoC

The hardest part of getting the PoC to work was finding out what the problems we encountered were and what caused them. This was much harder than usual because LoRaWAN is a relatively new system and does not yet have much online support. Whereas with problems with code in Python you can simply look up what is wrong and fix it. With LoRaWAN you only get vague error messages and it is hard to find out what is actually wrong. (In our case the Seeeduino was simply unable to find any network.)

7.5 Changes for future research

Here are some changes that can be made for future research to speed up the development process.

First off, make sure you have a connection to the network you wish to connect to and optimise connectivity by going outside in an area that should be covered and connect to the network via Authentication By Personalisation (ABP).

Secondly, use a commercial network like KPN-Things because it has the most coverage and has the largest chance of working right away. The Things Network (TTN) is simply too unreliable for development.

If however you really want to use TTN because of their practical console or because you want more visibility of what is going over the network, make sure you have your own gateway inside the building you are working from. This would guarantee that you have a connection and it would allow you to see what is actually transmitted through the gateway. This would speed up testing and developing a lot.

Bibliography

[1] N. Sornin, "LoRaWAN 1.1 Specification." LoRa Alliance Technical Committee, October 2017.

[2] C. U. Beser, Nurettin Burcak (Sunnyvale, "Operating cable modems in a low power mode," June 2008. [Online]. Available: http://www.freepatentsonline.com/7389528.html

[3] M. Mahmoud, "A Study of Efficient Power Consumption Wireless Communication Techniques / Modules for Internet of Things (IoT) Applications," *Scientific Research Publishing*, 2016.

[4] "LoRaWAN 1.1 Regional Parameters." LoRa Alliance Technical Committee Regional Parameters Workgroup, January 2018.

[5] X. Vilajosana, "Understanding the Limits of LoRaWAN," *IEEE Communications Magazine*, January 2017.

[6] E. van Es, "LoRaWAN vulnerability analysis: (In)validation of possible vulnerabilities in the LoRaWAN protocol specification." *Master thesis, Open University*, 2018.

[7] G. Avoine, "Rescuing LoRaWAN 1.0," *International Financial Cryptography Association*, June 2017.

[8] S. Zulian, "Security threat analysis and countermeasures for LoRaWAN join procedure," *Master thesis, University of Padova*, 2016.

[9] X. Yang, "LoRaWAN: Vulnerability Analysis and Practical Exploitation," *Master thesis, TU Delft*, July 2017.

[10] J. Chapman, "Mathematics of the Faraday Cage," *Society for Industrial and Applied Mathematics*, 2015.

[11] "IEEE Standard for Information technology–Telecommunications and information exchange between systems - Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation," pp. 1–594, May 2017.

[12] NIST, "Advanced encryption standard (AES)," November 2001.

[13] T. Iwata, JH. Song, R. Poovendran, J. Lee, "The AES-CMAC Algorithm Status," 2006. [Online]. Available: https://tools.ietf.org/pdf/rfc4493.pdf

[14] Ronald L. Rivest, Adi Shamir, Leonard Adleman, "Cryptographic Communications System and Method," September 1983. [Online]. Available: https://patentimages.storage.googleapis.com/49/43/9c/b155bf231090f6/US4405829.pdf

[15] B. Leiba, "Oauth web authorization protocol," *IEEE Internet Computing*, vol. 16, no. 1, pp. 74–77, Jan 2012.

Appendices

Appendix A

Code used in the Proof of Concept

Below you find the code used in the Proof of Concept. How the code became the way it currently is and instructions on how to do it yourself, see Appendix B. The keys in the code have been randomised and have to be set before the code is usable.

```
#include <LoRaWan.h>


unsigned char data[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA,};
char buffer[256];


void setup(void)
{
    pinMode(LED_BUILTIN, OUTPUT);

    SerialUSB.begin(115200);
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on
                                     //(HIGH is the voltage level)
    while(!SerialUSB){
       delay(1000);                  // wait for a second
    }
    digitalWrite(LED_BUILTIN, LOW);  // turn the LED off
                                     // by making the voltage LOW
    SerialUSB.println("Hello world!");

    lora.init();
    lora.setDeviceReset();

    memset(buffer, 0, 256);
    lora.getVersion(buffer, 256, 1);
```

```
SerialUSB.println("Reported version:");
SerialUSB.println(buffer);

memset(buffer, 0, 256);
lora.getId(buffer, 256, 1);
SerialUSB.println("Reported ID:");
SerialUSB.println(buffer);

SerialUSB.println("Setting IDs, keys and device mode");
// void setId(char *DevAddr, char *DevEUI, char *AppEUI);
//KPN OTAA
/*
lora.setId(NULL, "0059AC000000001E", "0059AC0000000001");
lora.setKey(NULL, NULL, "D8F8A93C78461E26465E12341F9A9367");
lora.setDeciveMode(LWOTAA);
*/
//KPN ABP

lora.setId("1420000C", "0059AC000000007B", "0059AC0000000001");
lora.setKey("6454df7d1de3b632527bfde8060e0c71",
    "ccadc20ba16660d4155f8a3bcac51c4a", NULL);
lora.setDeciveMode(LWABP);


// setKey(char *NwkSKey, char *AppSKey, char *AppKey);

SerialUSB.println("Setting data rate");
lora.setDataRate(DR5, EU868);

//SerialUSB.println("Setting adaptive data rate");
//lora.setAdaptiveDataRate(true);

//SerialUSB.println("Setting channels");
//lora.setChannel(0, 868.1);
//lora.setChannel(1, 868.3);
//lora.setChannel(2, 868.5);
//lora.setChannel(3, 867.1);
//lora.setChannel(4, 867.3);
//lora.setChannel(5, 867.5);
//lora.setChannel(6, 867.7);
//lora.setChannel(7, 867.9);

SerialUSB.println("Setting receive window");
lora.setReceiceWindowFirst(0, 868.1);
lora.setReceiceWindowSecond(869.525, DR0);

SerialUSB.println("Setting port");
lora.setPort(1);

SerialUSB.println("Setting duty cycle");
```

```
    lora.setDutyCycle(true);

    SerialUSB.println("Setting join duty cycle");
    lora.setJoinDutyCycle(true);

    SerialUSB.println("Setting power");
    lora.setPower(14);

    //SerialUSB.println("Setting PubNetwKey");
    //lora.setPubNetwKey(1);
    //lora.setPubNetwKey(1);

    /*
    SerialUSB.println("Setting OTAA join");
    while(!lora.setOTAAJoin(JOIN,10)){
        SerialUSB.println("Join not succeeded yet, looping.");
        for(int i=0; i<=3; i++){
          digitalWrite(LED_BUILTIN, HIGH); // turn the LED on
          delay(100);
          digitalWrite(LED_BUILTIN, LOW);  // turn the LED off
          delay(100);
        }
        digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
        delay(1000);
        SerialUSB.println("Setting OTAA join");
    }
    SerialUSB.println("Join succeeded.");
    */
}

void loop(void)
{
  SerialUSB.println("Still alive");
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on
                                   //(HIGH is the voltage level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off
                                   // by making the voltage LOW
  delay(1000);                     // wait for a second

    bool result = false;

    SerialUSB.println("Transfering packet");
    //result = lora.transferPacket("Hello World!", 10);
    result = lora.transferPacket(data, 10, 10);
    SerialUSB.print("Packet sent: ");
    SerialUSB.println(result);
    delay(10000);

  /*
```

```
    if(result)
    {
        short length;
        short rssi;

        memset(buffer, 0, 256);
        length = lora.receivePacket(buffer, 256, &rssi);

        if(length)
        {
            SerialUSB.print("Length is: ");
            SerialUSB.println(length);
            SerialUSB.print("RSSI is: ");
            SerialUSB.println(rssi);
            SerialUSB.print("Data is: ");
            for(unsigned char i = 0; i < length; i ++)
            {
                SerialUSB.print("0x");
                SerialUSB.print(buffer[i], HEX);
                SerialUSB.print(" ");
            }
            SerialUSB.println();
        }
    }
    */
}
```

Appendix B

Log of PoC procedure

This chapter describes the steps taken to get a Seeeduino working on either the network of KPN or TTN. For the actual final code used in the PoC, see Appendix A.

B.1 Getting the Seeeduino to work

These and later steps up until B.4 can also be found here.[1]

Below you find the steps you need to take to get the Arduino IDE (the IDE you have to work in) working for the Seeeduino. These are the first steps we have taken even before connecting the device to the computer:

1. Download the latest Arduino IDE: `https://www.arduino.cc/en/main/software`
2. Go the preferences/settings and under Additional Board Manager URLs add the following URL: `https://raw.githubusercontent.com/Seeed-Studio/Seeed_Platform/master/package_seeeduino_boards_index.json` If you already have one add the new one separated by a comma
3. Now head to Tools → Board → Boards Manager and install the Seeeduino SAMD board.
4. On Windows youll have to install a Serial driver as well. On my Linux this wasnt necessary
5. Now select in Tools → Board the Seeeduino LoRaWAN board
6. Under File → Examples → LoRaWAN select the OTAA sample

B.2 Hello world locally

In this section you find the code used to test whether you are able to receive messages that are sent over the USB cable from the Arduino to your computer. This is not as trivial as it sounds as you can read in the "*Notes for Ubuntu*

[1] `https://blog.squix.org/2017/07/seeeduino-lora-gps-getting-started-with-lorawan-and-ttn.html`

users".

To have the Seeeduino transmit a visible "Hello World" message, simply add `SerialUSB.println("Hello world!");`
before `lora.init();`.
This can then be viewed using the Serial Monitor (Ctrl + Shift + M or Tools → Serial Monitor).

Note for Ubuntu users: To be able to communicate with the Seeeduino through micro-usb, you first have to disable the *Modem Manager* service. This can be done with the following command:
`systemctl --now disable ModemManager.service` (requires root privileges)
You will also need to give the user port access to be allowed to communicate by adding the user to either the uucp or dialout group. This can be done by:
`usermod -a -G dialout your_username`
or
`usermod -a -G uucp your_username`
Afterwards a system reboot is required. (`reboot`)

B.3 Data over LoRaWAN

In this section the procedure is described that you need to follow to get network keys from your desired network, modify your code to use them and how to see whether data transmission is actually working.

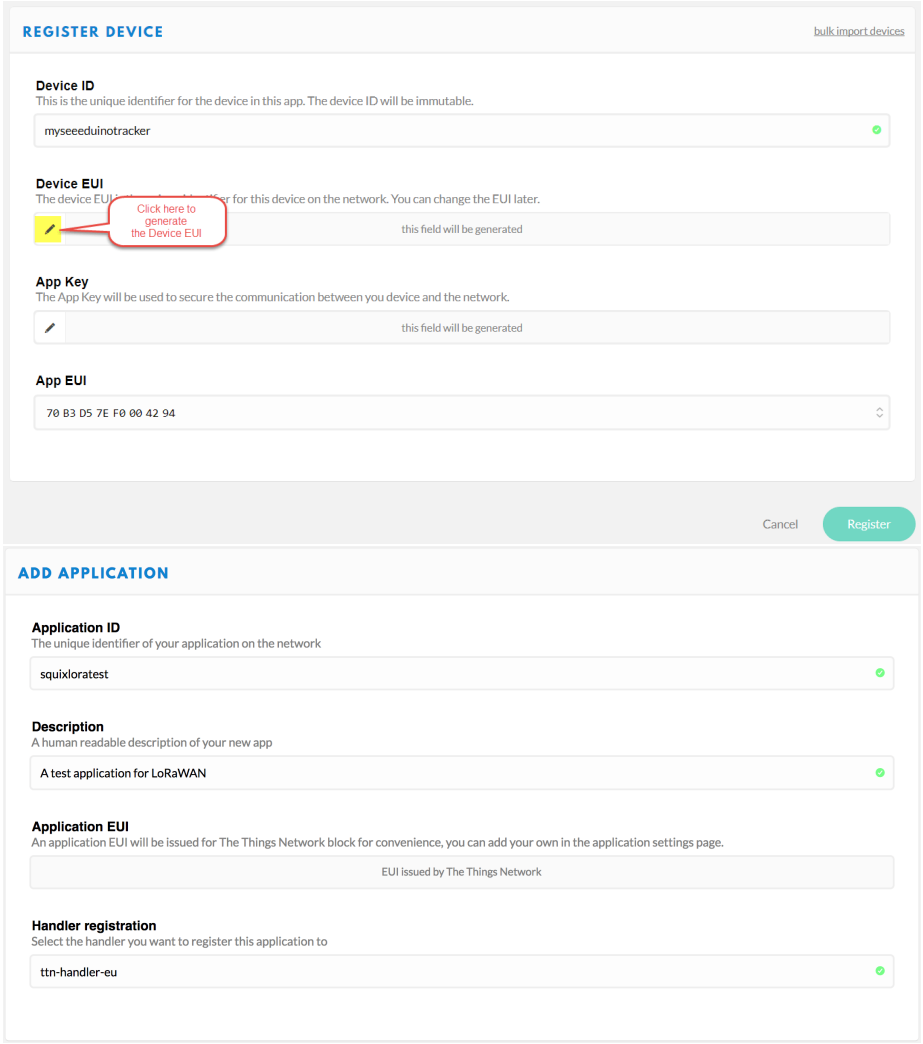
B.3.1 Acquiring network keys

To connect to a network you need to acquire network keys. Below you find a description of how to acquire your keys for The Things Network. This process can be repeated for the KPN-Things network, just from a different website.

TTN

1. Go to `https://www.TheThingsNetwork.org/` and register an account
2. Go to `https://console.TheThingsNetwork.org/` and select Application
3. On the next screen click "+add application"
4. In the next screen pick a meaning full name for your application in lowercase letters for Application ID and write something in Description. Under Handler Registration pick your region, for Europe I have picked ttn-handler-eu
5. On the next screen click "register device"
6. For "Device ID" pick a meaningful name in lowercase letters
7. On the left side of the DevEUI field click on the arrow icons. The icon changes into a pen and this will cause the ID to be generated. This is necessary since the API doesnt offer a way to read out the devices MAC address. See image below

8. In App EUI pick the ID of the application we created earlier



KPN

Connecting to the KPNThings network works in a similar way as connecting to TTN. However use KPN's website instead of TTN's: `https://loradeveloper.mendixcloud.com/login.html`

B.3.2 Adapting the code

Now its time to adapt the OTAA code. Replace the line lora.setKey(..) with the values below. I also copied here the signature of the two methods so you know which value you have to put there. setId(NULL, DevEUI, AppEUI) and setKey(NULL, NULL, AppKey):

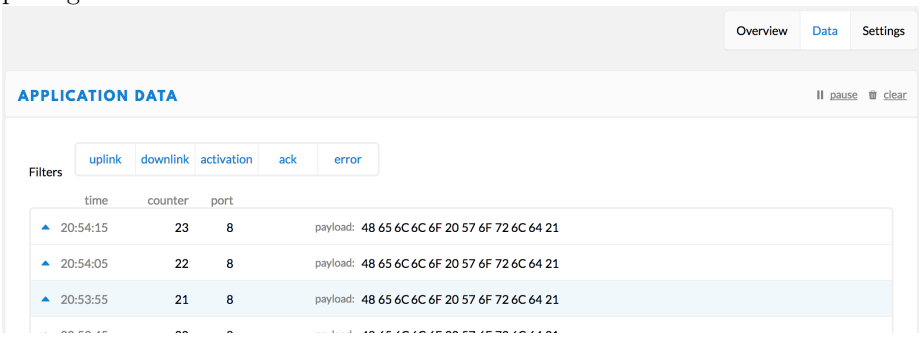
```
// void setId(char *DevAddr, char *DevEUI, char *AppEUI);
lora.setId(NULL, "12409E2345695432", "70B3D57EF0006593");
```

```
// setKey(char *NwkSKey, char *AppSKey, char *AppKey);
lora.setKey(NULL, NULL, "47BDA77B6D7B4DDA7DC182E54295FE4E");
```

Now make sure to select the Seeeduino under Tools ¿ Port and upload the sketch. If you are not owning a LoRaWAN gateway yourself but you think that one should be close then remove obstacles between you and that gateway. LoRaWAN can penetrate walls but how deep into the house the signal is strong enough depends on many factors. Upload the code and open the serial monitor. If everything worked out well your output should look something like this:

```
+VER: 2.0.10
+ID: DevAddr, 26:01:3A:44
+ID: DevEui, 12:40:9E:23:45:69:54:32
+ID: AppEui, 70:B3:D5:7E:F0:00:65:93
+ID: DevEui, 12:40:9E:23:45:69:54:32
+ID: AppEui, 70:B3:D5:7E:+KEY: APPKEY 47 BD A7 7B 6D 7B 4D DA 7D C1 82 E5 42 95 FE 4E
+MODE: LWOTAA
+DR: EU868
+DR: DR0
+DR: EU868 DR0  SF12 BW125K
+CH: 0,868100000,DR0:DR5
+CH: 1,868300000,DR0:DR5
+CH: 2,868500000,DR0:DR5
+RXWIN1: OFF; 3; 0,868100000; 1,868300000; 2,868500000;
+RXWIN2: 869500000,DR3
+POWER: 20
+JOIN: Start
+JOIN: NORMAL
+JOIN: Network joined
+JOIN: NetID 000013 DevAddr 26:01:21:52
+JOIN: Done
+MSG: Start
+MSG: TX "Hello World!"
+MSG: Done
```

Now you can also go back to your TTN or KPN console (website) and see if the packages have arrived there.



B.4 Code changes for ABP connection

This section is meant to clarify what has to change to the code acquired in B.1 in order to have the Arduino use ABP instead of OTAA to connect to the network server.

To connect through ABP several small code changes have to be made.

- Change the OTAA-setting in the (TTN or KPN) console to ABP and manually enter the Device EUI, Application EUI, Device Address, Network Session Key and App Session Key in the designated spots in the code, see the commented code here: B.3.2.
- Change `lora.setDeciveMode(LWOTAA);` to `lora.setDeciveMode(LWABP);`

B.5 Code optimisations

Since we were unable to connect to TTN, we applied some optimisations to the code that were suggested on the TTN-forum.

- Removed both the `lora.setReceiveWindow`'s because these are set by default and setting them manually allows for mistakes.
- Changed the initial datarate to DR3: `lora.setDataRate(DR3,EU868);`
- If you are still unable to find bandwidth, you can change the `join`-loop to add some extra time during which the board will attempt to join the network before giving up. `while(!lora.setOTAAJoin(JOIN, 15)){`