Østfold University College

BACHELOR'S THESIS

Covid-19 vaccine temperature monitoring by transmitting measurement data using LoRa and LoRaWAN technology

07.06.2021

B21E05
Mathias Karlsvik
Anders Wiig
André Larsen
Daniel Muri

Bachelor electrical engineering
Fredrikstad

Abstract

The project goal was to develop a system for monitoring Covid-19 vaccine temperatures during storage using LoRa and LoRaWAN technology. The project based its temperature range criteria on Pfizer's vaccine. The monitoring of this temperatures is important to make sure the Covid-19 vaccine keep the correct temperature while in the ultra-cold freezer, and therefore keeping the expected life expectancy before the next step in the vaccine process.

To solve this task there were implemented a LoRaWAN end device for temperature measurements. The end device transmits its data to a back-end system for data storage, data visualisation, and user notification when temperatures reach crucial levels.

Measuring ultra-low temperatures was done successfully by using a K-type thermocouple. These measurements are successfully transmitted to the back-end system. The measurements were visualised in a way that suits the measurement type, and the user was successfully notified by SMS and email when the temperature reached a crucial level.

The project did not have a pre-study form, but topics that needed study became clear as the project progressed. For development, the project team used the rapid prototyping approach.

The project is concluded as a good foundation for further development and it fulfils what the project set out to do.

Contents

1 Introduction

This project aims to solve the task of monitoring Covid vaccine temperatures during shipment and storage. The temperature criteria for the vaccines are based on Pfizer's Covid-19 vaccine. The project team wanted to provide a solution to the issue where vaccines are unusable if they are not stored in a low enough temperature. This issue is addressed with the development of a prototype for temperature measurement and data communication with the help of LoRa and LoRaWAN technology. In addition to the prototype end device, the project also implemented a back-end system for a private LoRaWAN network with data storage, data visualisation and notification for critical temperature level. The development is done using rapid prototyping, and the project has a study of relevant literature during the project process. This means that the project has a practical and a theoretical approach to solving this task.

1.1 Team members

Anders Wiig
anderswi@hiof.no
+47 93461796

Daniel Muri
danielmu@hiof.no
+47 47390806

Mathias Wexsahl Karlsvik
mathiawk@hiof.no
+47 95473637

André Larsen
andrelar@hiof.no
+47 90672125

1.2 Client

Østfold University College department of electrical engineering is located at campus Fredrikstad. The electrical-engineering education is a comprehensive, profession-oriented, and research-based education. The study plan has been prepared with the business community and is adapted to the working life's need for basic engineering competence [1]. The study forms the basis for further development in the practice of the profession. Our project advisors are associate professor Manikandan Palanichamy and assistant professor Reidar Nordby.

1.3 Project background

The project began with a presented topic for the thesis by Manikandan Palanichamy on behalf of Østfold University College. The topic presented was LoRaWAN technology in the context of Internet of Things (IoT) and how to create "smart" things. This technology can have several different use-cases where the amount of data to be transferred is small, i.e., such as that from sensory systems.

With the main topic for thesis ascertained, the project team had to come up with ideas and thoughts on how to define the project in greater detail. The use-case that the team found interesting was that of which had a valuable connection with fighting Covid-19, and the distribution of vaccines. Therefore, the team decided to develop the project in the direction of monitoring Covid-19 vaccines storage parameters.

1.4 Use case

The use case is based on the temperature criteria for the Pfizer Covid vaccine. This vaccine has the following specific requirements for storage:

- *Before mixing, the vaccine may be stored in an ultra-cold freezer between -80^{o}C and -60^{o}C* [2].

The goal was to monitor the vaccines before the vaccines are mixed and prepared to ensure that they held the correct temperature while in the ultra-cold freezer, and therefore keeping the expected life expectancy before the next step in the vaccine process. The system is designed to monitor the temperature by having an end device attached to the vaccine-freezer or shipment. This then sends data to a back-end system that shows the temperature in given intervals. This provides temperature monitoring to check if the freezer or shipment temperature rises above of falls below critical level.

1.4.1 Vaccine temperature monitoring during storage and transport (Logistics)

LoRaWAN nodes can be moved easily between locations and gateways and still connect to the back-end system. This is because there is no hand-over between gateways [3]. This means that when a shipment of vaccines moves from for example a warehouse freezer to

a truck (or other means of transportation), the nodes surveying the shipment can seamlessly connect to the next gateway. This is further discussed in chapter “5.6 LoRa and LoRaWAN Network coverage”.

During storage

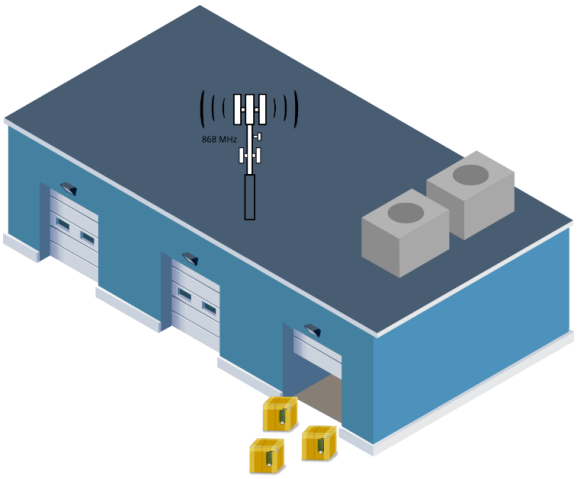

Figure 1 One gateway provides LoRaWAN coverage for the warehouse [4]

During transportation

Figure 2 One gateway provides LoRaWAN for all shipments inside the truck [4]

1.5 Thesis statement

How to utilize LoRaWAN for monitoring Covid-19 vaccine temperature, during transportation and storage?

1.5.1 Partial thesis statements

- How does a LoRaWAN system work?
- How to measure very low temperatures?
- How to develop a prototype to measure very low temperatures and transmit measurements via LoRa and LoRaWAN.
- How to implement a back-end LoRaWAN system for receiving and conveying measurement data.
- How to view and store the measurement data that arrives at the back-end of the LoRaWAN system.
- How to notify when the measurement reaches a critical level?

1.6 Thesis limitations and definitions

The project is limited to the geographical region of Norway and EU, in terms of allowed frequencies and frequency use. It is also limited to the temperature criteria of Pfizer's vaccine only. The project limits itself to the development of a prototype end device and not a finished product.

1.7 Project goals

The overall goal for this project was to learn about LoRa and LoRaWAN and develop a LoRaWAN solution with a node and gateway connected to a back-end system for data processing. This made for a project that combines both a theoretical approach as well as a practical one. Therefore, this project has both process goals and product goals.

1.7.1 Process goals

The process goals in this project were mainly learning goals. The learning process was ongoing under the whole project. This is because the development of a prototype revealed topics that required some insight and helped forming the theoretical part of the project. Therefore, the theoretical part of the project was not in a pure pre-study form. The short timeframe of the project also demanded starting with development as soon as possible.

1.7.2 Product goals

The products were a LoRaWAN prototype and this bachelor thesis, which is a project report. The main goal for development was to develop an end device and a gateway wireless solution using LoRaWAN. The end device generated data and transmitted it using LoRaWAN, and the gateway- and end-system received and processed the data. A sub product goal was setting up a LoRaWAN system with data storage and processing capabilities.

1.8 Project implementation system overview

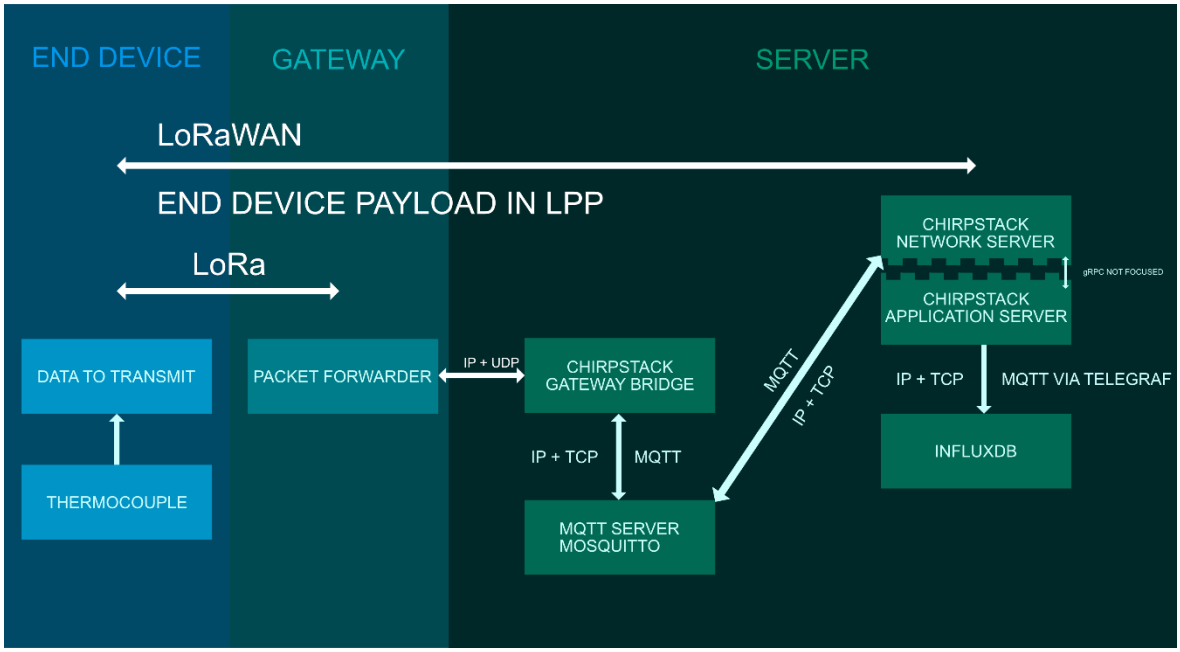

Figure 3 The different parts in the implementation [5].

The project implements a complete LoRaWAN system consisting of an end device, which is a prototype developed by the team, and an implementation of a back-end system that is a server solution with different components. "Figure 3 The different parts in the implementation" shows how all the components connect with each other to form an entire system solution. This figure will be referred to in several of the following chapters to provide an overview of the parts covered. This will be done by highlighting the given element in the figure.

1.9 Why LoRaWAN? – LoRaWAN compared to mobile networks?

LoRaWAN was originally developed to be for industrial IoT Sensors. The devices in a LoRa network can communicate over long distances, even if the devices are in separate buildings. A substantial lower power consumption makes for longer lasting batteries than the 4g/5g units. Whilst 4g/5g needs a vast infrastructure, LoRaWAN runs on smaller and more affordable devices. This makes for a reliable, affordable network for our use-case. [6]

Further discussion around the use of LoRaWAN in comparison with mobile networks are covered in chapter “5.5.1”.

1.9.1 Existing solutions

There are solutions that are comparable to the project. Pfizer’s solution is simple, while the Orion M2M solution is similar to what the team wanted to develop. This is compared to this project’s solution in more detail in chapter “5.4”.

2 Methodology

The project utilizes literature study for the relevant theory. During development, some relevant theoretical topics presented itself as needing further study, hence the project did not have a pure pre-study but an ongoing study of relevant topics. With some limited possibilities for testing, the project relies on data sheets providing correct information.

This project can be divided into two main categories: product development and writing of the report. Everyone was given the necessary components to build a prototype and the team members worked individually and had digital meetups for discussions about the development of prototypes and writing the thesis report. The report was divided into smaller parts and chapters and divided evenly among team members.

2.1 Rapid prototyping – iteration-based development

The development of a LoRa / LoRaWAN prototype was done with the use of the rapid prototype approach, which is an iteration-based methodology. The process covers the steps from concept to product, which is based around designing, building, testing, and

analysing the prototypes. Rapid prototyping allows for early functionality testing and is a fast and effective way to produce an end product. The production for the end device and designing of a backend system has been implemented by testing, analysing, and redesigning a prototype. [7]

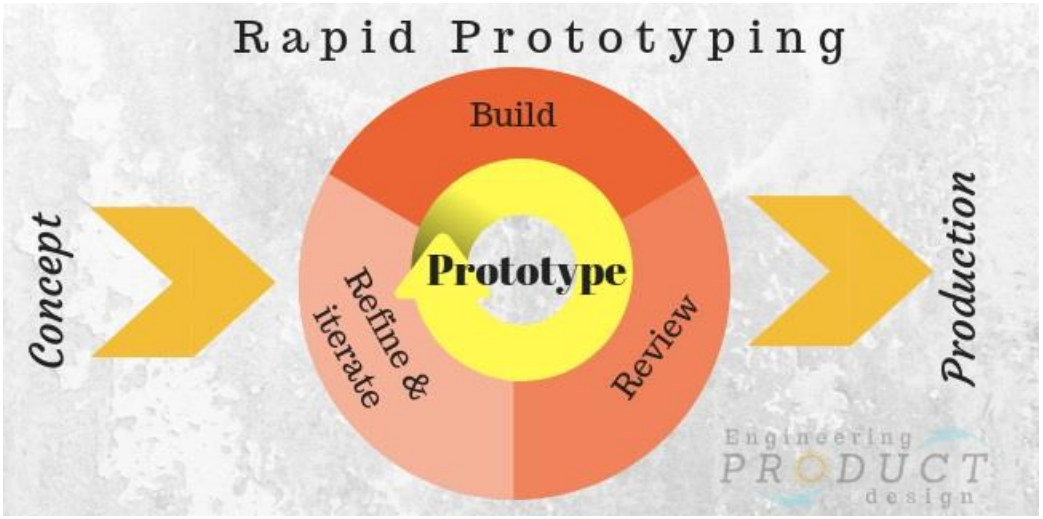

Figure 4: Rapid Prototyping [7]

3 Theory

This chapter covers the different theoretical parts needed in this projects implementation of a complete LoRaWAN-system. It also covers how to measure very low temperatures and power consumption for the microcontrollers used in the project. The different subchapters cover the OSI model, what LoRa is, which protocols are used, and theory around the parts needed for implementation of a back-end system.

Understanding the fundamentals of the OSI-model is helpful in visualizing a LoRaWAN network as different parts of the systems are operating on different layers. For example, LoRa operates on the physical layer whilst LoRaWAN is operating on both the data link and network layer. An explanation of the OSI-model will be given further in 3.1.

The 3.2 LoRa chapter covers what LoRa is, its specifications and the LoRa modulation technique. It also provides theory of why it is a useful and relevant technology.

The 3.3 protocol sub-chapters explain the LoRaWAN protocol, how MQTT works, and how the payload can be encoded with the Cayenne Low Power Payload protocol so that the back-end system can understand what the data received is.

The chapters 3.4, 3.5 and 3.6 explain the different components in the back-end system, what they are and how they are connected to create a complete back-end system. Chapter 3.4 covers the components needed for the LoRaWAN back-end part. The chapters 3.5 and 3.6 are about Telegraf and InfluxDB describe how these components work with the regards to data sourcing, storage, and visualisation.

The measuring of ultra-low temperatures and technology for such measurements are covered in chapter 3.7. The thermocouple was the chosen method to measure the ultra-low temperatures that Pfizer's vaccine requires, after being evaluated against some other methods.

3.1 OSI-model

An introduction of the Open Systems Interconnection (OSI) model is provided here and used as a reference point throughout the theory and implementation chapter of this report.

Figure 5: OSI-model [8]

The OSI-model is a reference model used in data communication. The model is defined by the International Organization for Standardization (ISO), and it represents the building blocks of a network communication system. The model consists of seven layers: application, presentation, session, transport, network, data link and the physical.

Application layer: Refers to the end user layer with high-level APIs including resource sharing and remote file access. Examples: HTTP DNS and FTP.

Presentation layer: The data translator for the network. Its job is to ensure that the data sent from one application layer is readable by the receivers' application layer.

Session layer: Services contains mechanisms for managing communication sessions between end-users. For example, opening, closing, and managing a semi-permanent dialogue.

Transport layer: Provides a host-to-host server between application servers. Examples on services included connection-oriented communication, flow control and multiplexing. Examples are TCP/IP and UDP.

Network layer: Provides packet forwarding systems like routing. Its functionality includes connectionless communication, host addressing, message forwarding etc.

Data Link layer: Refers to the data transfers between network entities. This may also include error detection and error correction.

Physical layer: The lowest layer in the OSI-model. Refers to the actual physical mediums that the signal is sent through. [8]

3.2 LoRa

LoRa is a wireless transmitting system that uses radio waves for communication between end device and gateway. LoRa is a spread spectrum modulation technique derived from the Chirp Spread Spectrum technique (CSS) and is purely a physical layer implementation, as defined by the OSI-model. LoRa is a wireless transmitting system that uses radio waves for communication between end device and gateway. The LoRa technology provides a trade-off between sensitivity and data rate. This is described as the spreading factor, where a low spreading factor increases the data rate while lowering the reception sensitivity and vice versa. This allows an adaptive optimization for LoRa-devices where the goal is to preserve end node battery life. [9]

3.2.1 LoRa specifications

The LoRa technology operates in the license free ISM band. This increases availability for developers and users to design and implement LoRa technology for their systems without a license cost. Traffic regulations on these license free bands are a necessity, and in Europe, this is done by the ETSI (European Telecommunications Standards Institute). Mainly targeting regulation of duty cycle and maximum power transmitted.

3.2.2 LoRa Modulation

Lora technology uses a modulation technique called Chirp Spreading Spectrum. To fully understand the motivation behind developing the LoRa modulation technique, some extra theory about the Shannon - Hartley Theorem and Spread Spectrum principles is provided in appendix 1. Mainly the LoRa modulation technique provides strong resistance against interference and can reach far with low power. The standard preamble of a LoRa-signal consists of 8 symbols, consisting of 6 up-chirps and 2 down-chirps. This is followed

by the data of the signal [10]. An example of a modulated LoRa-signal is in the figure below.

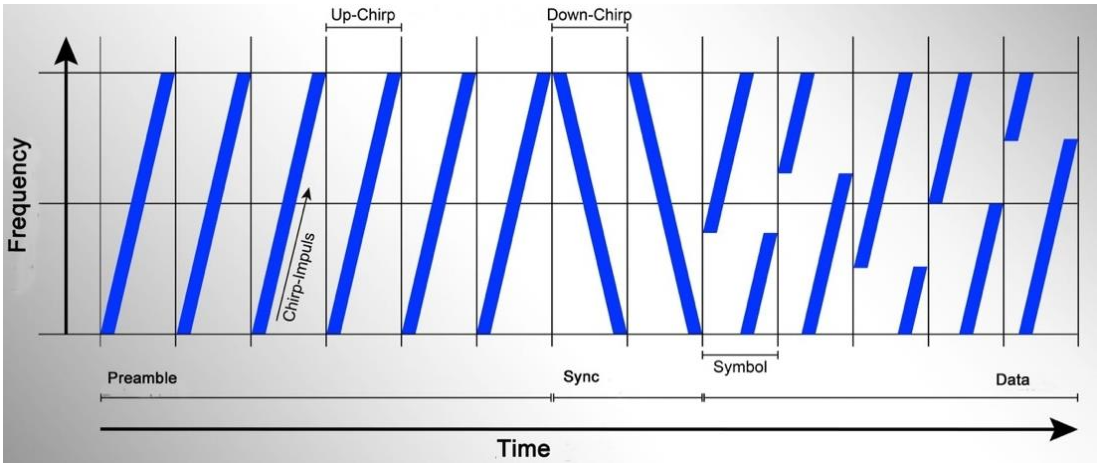

Figure 6: Example of a LoRa-signal [11]

3.2.2.1 Chirp Spread Spectrum

Chirp Spread Spectrum (CSS) is a modulation technique that is commonly used in sonar and radar technology. "Chirp" stands for Compressed High Intensity Radar Pulse and is a simple signal with constant amplitude that varies in frequency. A up-chirp goes from a low frequency which increases at a constant rate to a high frequency as seen in the figure below. A down-chirp goes from high to low. [12]

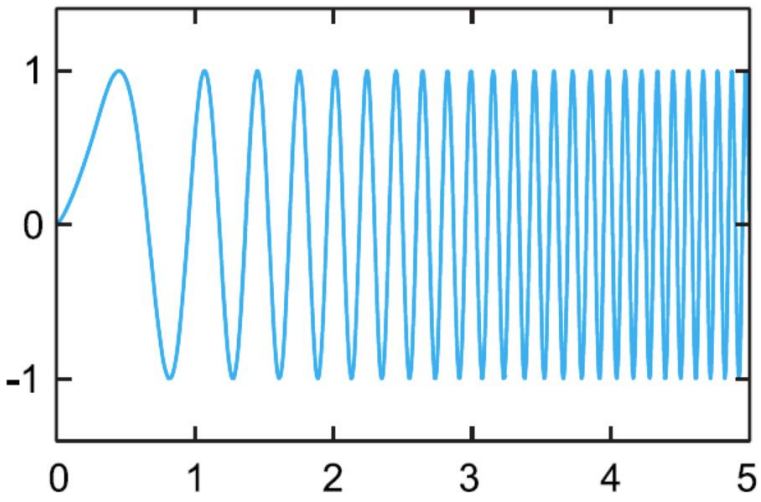

Figure 7: Chirp Signal [9]

3.3 Protocols

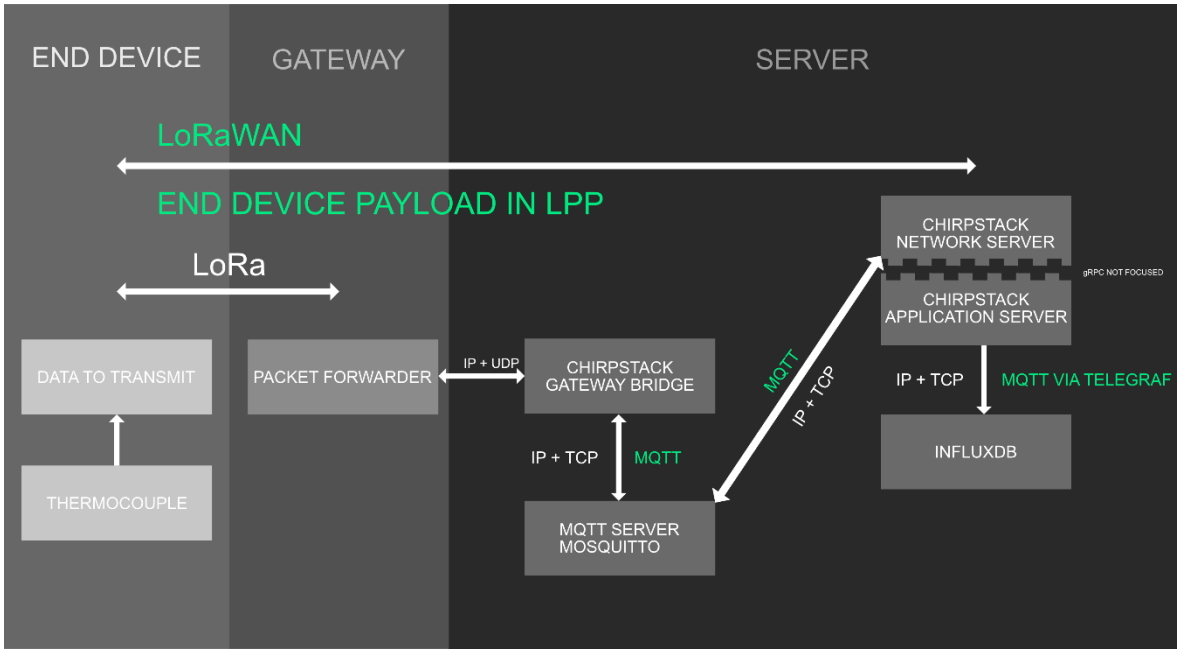

Figure 8 Protocols used in project implementation [5].

This project relies on three main protocols. These are LoRaWAN, Message Queuing Telemetry Transport (MQTT) and Cayenne Low Power Payload (LPP). LoRaWAN is used all the way from the end device to the application server. MQTT is used internally in the ChirpStack server system implementation. Here MQTT is used for transporting data from the gateway bridge to the network server and from the application server to an InfluxDB database via Telegraf. The data that the end device is transmitting is constructed in an array that is created using the LPP format. The three protocols are presented in the following chapters.

3.3.1 LoRaWAN

Figure 9 LoRaWAN Logo [13]

3.3.1.1 What is LoRaWAN

Long Range Wide Area Network (LoRaWAN) technology is an open network protocol that uses LoRa as its physical layer and is used for device administration and communication. It was designed to allow secure bi-directional communication between low-powered devices and internet-connected applications over long-range wireless connections. The protocol is standardized and maintained by the LoRa alliance, which is an open association of collaborating members.

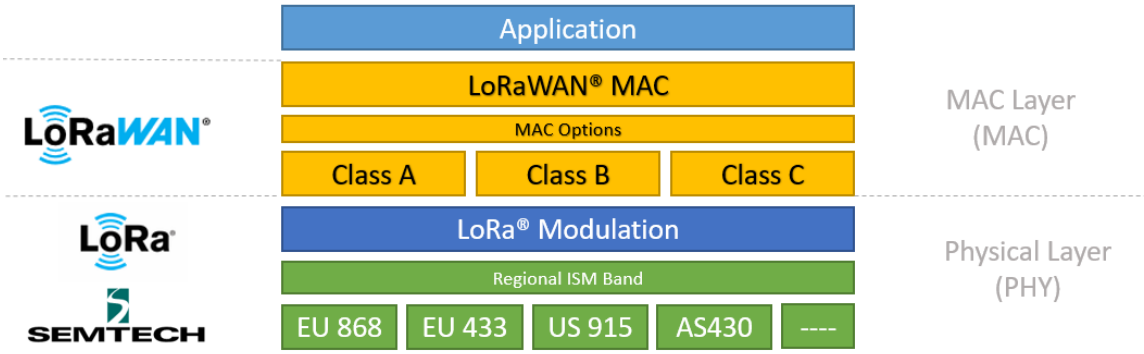

Figure 10 LoRaWAN technology stack [9]

The LoRaWAN layer in "Figure 10 LoRaWAN technology stack" which includes the MAC layer, and the different classes of end devices defines the communication protocol and system architecture for the network. In reference to "Figure 5: OSI-model [8] " the LoRaWAN layer from the technology stack can be mapped to the data link layer and the Network layer, which is the second and third layer, respectively. [9] [14]

3.3.1.2 LoRaWAN function and network elements

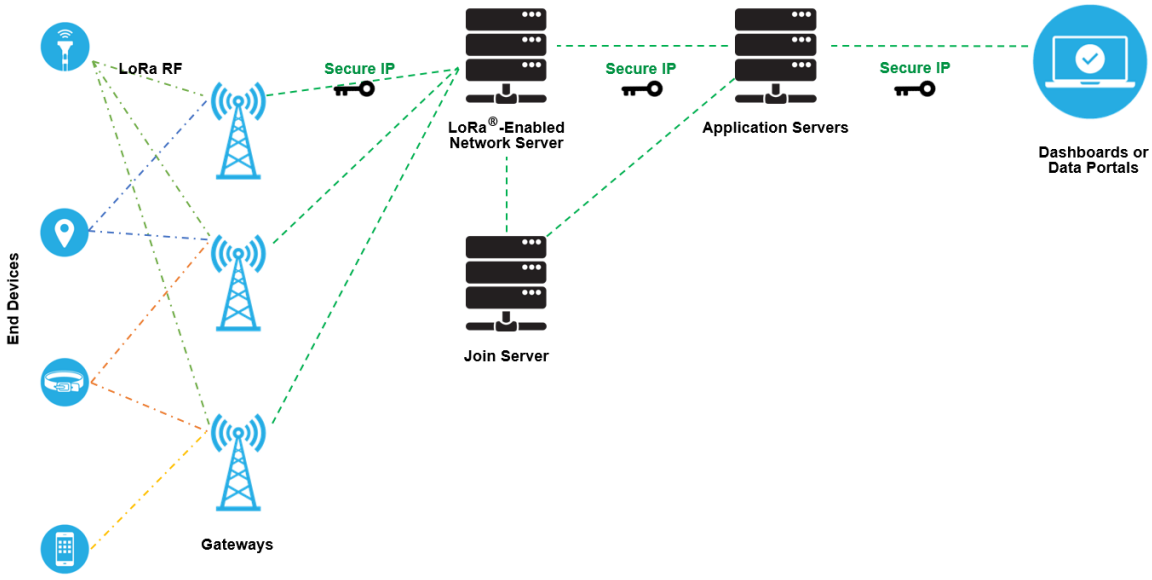

Figure 11 Typical LoRaWAN network implementation [9]

The figure above shows a typical LoRaWAN network implementation and consists of end nodes, gateways, a network server, a join server, and an application server. In a typical LoRaWAN network, nodes are not linked to a specific gateway, instead the transmitted data is received by multiple gateways. The data will then go through a redundancy check where the incorrect data will be dropped. Then each of the gateways will forward the correct received data to the network server via Internet. A critical function for moving nodes is that there is no need for handover from gateway to gateway. [14]

3.3.1.2.1 End device

A LoRaWAN-enabled end node can be an actuator or sensor and is connected wirelessly to a LoRaWAN network through LoRa RF modulation via gateways. End devices are often an autonomous battery powered device that controls physical objects like a water valve or reads environmental data such as temperature and humidity.

When manufactured, the devices are assigned unique identifiers that is used to administer and activate the device. This is done to warrant secure transport of data packets over both private and public networks. [9]

Device classes

End devices in a LoRaWAN network can operate in three classes. The classes specify how a device communicates with the network. In this project class A was implemented, but all classes are described. All devices need to support Class A. Class B devices will have to support both Class A and Class B, and a Class C device needs to support all three classes. [9]

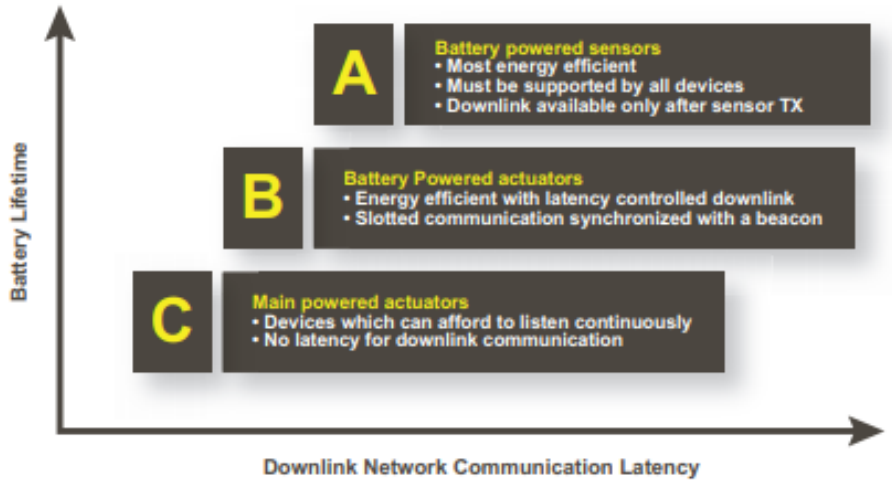

Figure 12 Device classes [14]

Class A

Class A devices spends most of its time in idle state. The device wakes up to initiate an uplink and transmits the data if there is any change in whatever the device is set to monitor. It will then listen to a response from the network in a short receive window of around one second. It will go back to idle state for a brief period if it does not receive a downlink, waking up shortly after to again listen for a response. if there is still no response, the device will go back to idle state until it has new information to report back. [9]

Class B

Class B builds on Class A making it possible to provide regularly scheduled receive windows as well as having the ones from whenever a Class A uplink is sent. For the communication of Class B to work, a beaconing prosses is required. The beaconing process is that the network periodically broadcast a time-synchronized beacon to end devices so they can align their internal timing reference. The end devices can open receive windows periodically based on the timing reference of the beacon. This requires that the gateways in the network have a built-in GPS timing source, to ensure that they will be synchronized to the precise beacon timing. [9]

Class C

Class C devices are always “on” as they are main powered, meaning they will always listen for downlink messages, except for when they are transmitting an uplink. This results in low latency for communication between the server and the device. It also makes use of the same receive windows as Class A but will not stop the second window until it sends new information to the server. [9]

3.3.1.2.2 Gateway

The gateway receives messages from any end device within reachable distance and forwards the received data to the LoRaWAN network server (LNS). The gateway is connected to the LNS through an IP backbone link. Since end devices can be served by multiple gateways, all uplink data sent by an end device will be received by all the gateways accessible. This notably reduces the error rate of data packets and will both allow low-cost geolocation and reduce battery consumption for mobile sensors.

IP traffic to the network server from a gateway can be backhauled trough either Ethernet, Wi-Fi, or cellular connection. LoRaWAN gateways can be looked at as LoRa radio message forwarders as it only operates on the physical layer. They check the integrity of the received data and if the cyclic redundancy check (CRC) is incorrect, the message will be dropped. And if its correct, the gateway will forward the message to the LNS alongside some metadata such as the received signal strength indicator (RSSI) and an optional timestamp. A gateway will without any interpretation execute the transmission request

for LoRaWAN downlinks. The LNS will perform data de-duplication and delete all copies if multiple gateways receive messages from the same end device. The network usually selects the gateway that received the message with the best RSSI since that gateway is most likely the closest to the end device. [9] [15]

3.3.1.2.3 Network server

A LoRaWAN network server manages the entire network and to adapt to new changes, it will dynamically control the network parameters. It will also establish secure connections that is encrypted with 128-bit Advanced Encryption Standard (AES) for end-to-end data, which is the encryption standard used in LoRaWAN. The LNS will ensure both the integrity of every message as well as the authenticity of all the sensors on the network. [9] [15]

Some features all LoRaWAN networks have in common:

- Address checking for devices.
- Frame counter management and authentication.
- Adapting data rates.
- Responding to MAC layer requests from devices.
- Acknowledging received messages.
- Forwarding Join-requests and accepts to the join server and devices.
- Forwarding uplink payloads to the right application servers.
- Queuing downlink payloads from the application servers to the connected devices. [9]

3.3.1.2.4 Application server

The application server's responsibility is to securely manage, handle and interpret sensor data, and they will also generate the application layer downlink payloads for the end nodes connected to the network. [9]

3.3.1.2.5 Join server

A join server manages over-the-air-activation (OTAA) processes for the end devices requesting to join the server. The join server processes the frames of uplink join-requests and is generating the join-accept frames. It decides what application server an end device should be connected to through signaling the LNS and it also performs the derivation of the application and network session key encryption. The join server then communicates the device application session key to the application server and network session key to the network server. [9]

For every end device, the following information must be contained from the join server:

- DevEUI (end device serial unique identifier)
- AppSkey (Application session key)
- NwkSkey (Network session key)
- Application server identifier
- End device service profile

[9]

3.3.1.2.6 Security

Two layers is utilized for security in LoRaWAN, one for the network that ensures authenticity of the node and one for the application that makes sure the network does not gain access to the end users application data. [9] [15]

The two layers of encryption defined in LoRaWAN:

- A 128-bit Network session key that is shared between the end device and the network server.
- A 128-bit Application session key that is shared at the application layer for end-to-end data.

[15]

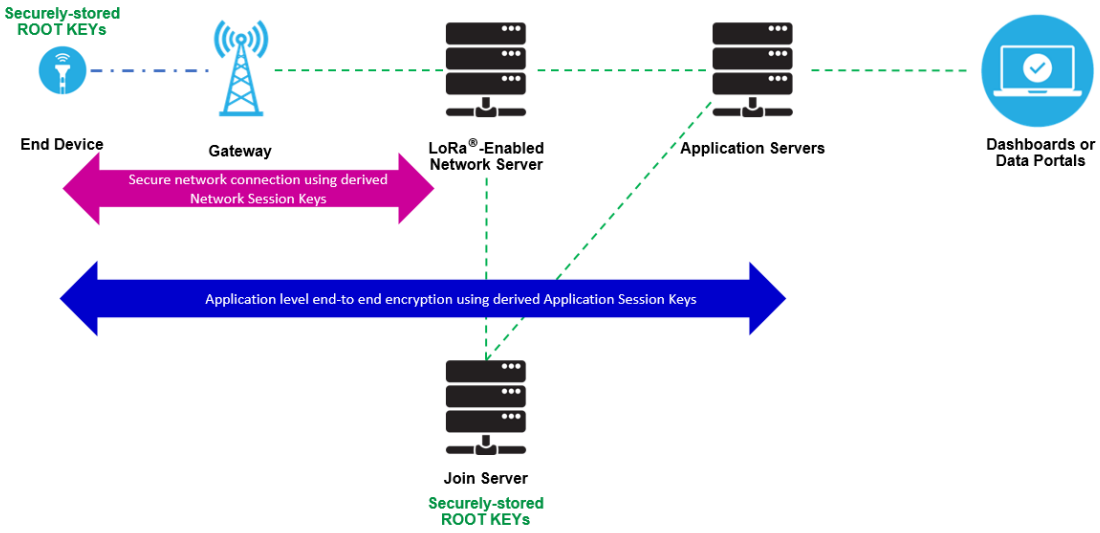

Figure 13 Secure transmission of data packets [9]

Data in a LoRaWAN network is encrypted twice, once by the node and once more by the LoRaWAN protocol. The data will only be sent to the gateways after these encryptions. After the gateway then forwards the data to the LNS, the network will then use the network session key and decrypt the LoRaWAN data. This data will then be sent to the application server which uses the application session key to decrypt the sensor data. This is important to make sure other gateways in the proximity do not gain access to other people's data since end devices send data to all nearby gateways within proximity. [9] [15]

At the start of operation, devices need to be activated and joining the network to ensure good quality of service, billing, and security. It is two types of activation specified in LoRaWAN, it is over-the-air-activation (OTAA) and activation by personalization (ABP). OTAA is the preferred one. [9]

For OTAA, the network (join server) and the joining device will exchange a 128-bit Application key (AppKey). This AppKey will be used to create a Message Integrity Code (MIC) when the device sends a join-request, the server will then check the AppKey with the MIC. And if this check is valid, the server will create two new keys, the 128-bit AppSkey and the 128-bit NwkSkey. The join server will send these keys back to the device making use of the AppKey as an encryption key and the device will install the two session keys as soon as they are received. The AppSkey is used for end-to-end encryption all the

way from the application server to the end device and the NwkSkey is used to warrant the integrity of a message from the end device to the LNS. [9] [15]

ABP is the second method to join the network and will directly link the device to a specific network. With this method, the user will insert the device session key which may lead to security issues. [15]

ABP	OTAA
A less secure and simplified commissioning process	The essential provisioning parameters is generated autonomously by the manufacturers
At fabrication, keys and IDs are personalized	Can renew secure keys routinely
Skips the join procedure which makes devices functional immediately after powering up	Can securely and dynamically switch networks as it can store multiple identities
Since the NetID is a part of the device network address, the device is bound to a specific network	High-grade security with tamper-proof options

Table 1 A comparison of the activation types OTAA and ABP [9]

3.3.1.2.7 Frame structure

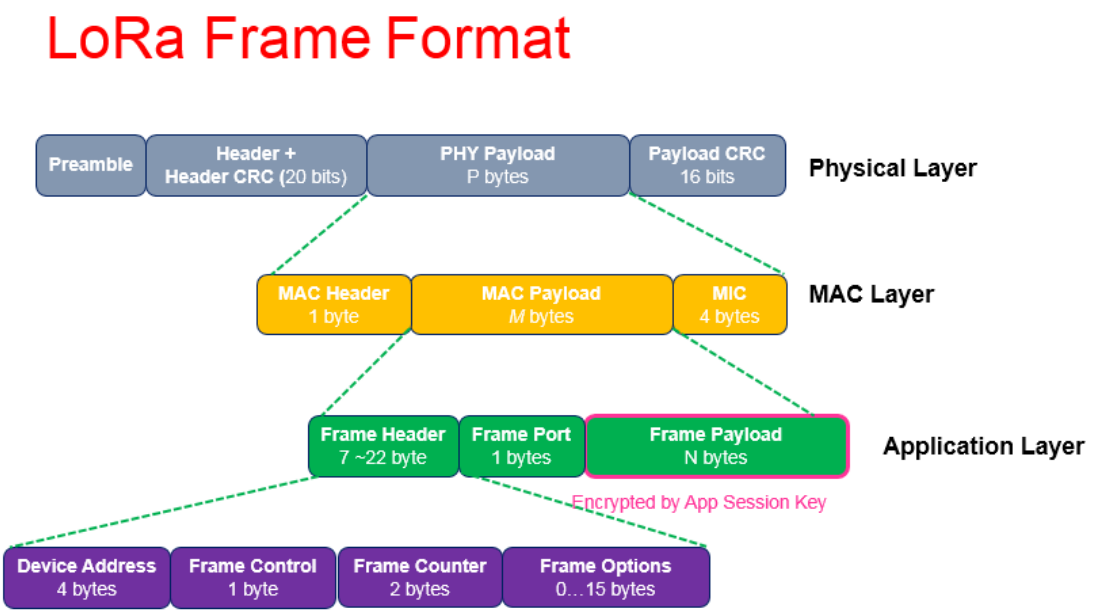

Figure 14 LoRa and LoRaWAN frame format [16]

The physical layer frame starts with a preamble, which is responsible for the synchronization function and defines the modulation scheme for the packet and usually has a duration around 12.25 ms. After the preamble comes the Header and a Header CRC, these two together are 20-bits long. The header contains information such as payload length and if the payload 16-bit CRC is present. The payload CRC is only present in uplink frames for LoRa networks. And as seen from the figure above, the PHY payload contains the MAC layer frames.

The frames in the MAC layer are the MAC Header, the MAC payload, and the Message Integrity Control (MIC). The MAC Header is 1-byte and defines the message type and the protocol version. Examples of this is whether it should be acknowledges or not, or if it is transmitted in downlink or uplink. If an end device is in a join process, the payload may be switched out with a join request or join accept message. The MAC payload contains the application layer frames. To compute the 4-byte MIC value, both the MAC payload and the MAC Header are used in its entirety with a network session key. The use of the MIC value is to authenticate the end device as well as preventing any forgery of messages.

The application layer consists of the Frame Header, Frame Port and Frame payload. The information in the Frame header which is between 7 to 22 bytes are the device address,

Frame control, Frame counter and Frame options. Device address is 4 bytes and is in two parts where the 8 first bits identify the network, and the other bits are dynamically assigned during the join- and identifying process. Followed is the 1-byte Frame control used for network control information. The Frame counter is 2 bytes large and used for sequence numbering and last in the Frame Header is the Frame options which is used to send commands to change transmission power or data rate and is between 0 to 15 bytes. The Frame ports value is decided by the application type. The value of a Frame payload is encrypted with an AppSkey. [16]

3.3.1.3 Advantages of LoRaWAN

- An open standard: LoRaWAN is an open standard technology which allows for bi-directional communication and adaptive data rate. The reasoning for it being an open standard is to get wide deployment and acceptance.
- Open frequencies and no licensing: LoRaWAN operate within the frequency spectrums of 433Mhz, 868Mhz and 915Mhz which is available worldwide, is free to use and requires no licensing. In Norway, the 868Mhz frequency band is the standard but the 433Mhz frequency band is also available if needed.
- Low power consumption: The power consumption of end devices is low since the end devices in a LoRaWAN network are asynchronous and only communicate whenever they have data ready to be sent. This protocol type is often referred to as the Aloha method.
- Security: Security is one of the main focus points for LoRaWAN, that is why two-layer security are utilized. One layer is for the network while the second layer is for the application where AES encryption is used.
- MQTT compliant: As the gateway needs to forward the data from the end devices to the network server it is dependent on a way to communicate with the server. And in many cases MQTT is the selected protocol for this task and is a publish-subscribe protocol that transports messages between the devices.
- The LoRa Alliance and global reach: One big advantage of LoRaWAN is that it has a lot of backing in the LoRa Alliance which has existed since 2015 and consist of

more than 500 members worldwide. The members include some technology leaders such as Cisco, HP, IBM and Semtech as well as some leading product manufacturers such as Bosch and Schneider and many other.

- A single gateway is designed to handle thousands of end devices.
- Wide covered area with its long range as it can reach up to 15km outdoors, this is seen as both an advantage and a disadvantage.
- Fully bi-directional communication.

[9] [14] [15]

3.3.1.4 Limitations of LoRaWAN

- Signals may be unstable in urban areas due to densification of LoRaWAN networks. This may be a problem both for open frequencies as you may get interference and low data rate due to heavy traffic on the frequency and because the end devices send information to all the gateways in its radius.
- It is not made for large data loads as the payloads are limited to 100 bytes and the data rate is low ranging between 250bps to 50kbps.
- While the 863-868 MHz frequency band in Norway is open for use, there is limitations to effect, occupied bandwidth and transmission time.

[9] [15] [17]

3.3.2 Message queuing telemetry transport protocol (MQTT)

Figure 15. MQTT logo [18]

3.3.2.1 What is MQTT?

3.3.2.1.1 General

“Message Queuing Telemetry Transport” is a protocol used for Internet of Things (IoT) and machine to machine (M2M). MQTT is located in the application layer in the OSI-model (See "Figure 5: OSI-model [8]") [19]. It is a messaging protocol that is designed in a way that requires few resources. It is open, simple, and easy to implement which makes it ideal for many situations. The design is also prepared for operating on unreliable networks, networks with low bandwidth and networks with low response time. The design principles of MQTT are about minimizing the network requirements of bandwidth and device resources, whilst still proving reliable with a certain level of deliverance. This makes it a reliable protocol where there is limited bandwidth, battery capacity or power consumption. [18] [20]

MQTT was invented by Dr. Andy Standford-Clark of IBM, and Arlen Nipper of Arcom (now Eurotech), in 1999. The protocol is, since its release, used by a wide variety of industries. The current version of MQTT is version 5 (v5.0) which is an open OASIS-standard. MQTT v3.1.1 is also a certified ISO-standard. [18] [20]

3.3.2.1.2 MQTT v3.1.1 and MQTT v5

MQTT v3.1.1 was the first OASIS standard version of MQTT, and it was also standardized as ISO/IEC 20922:2016. With the release of MQTT v5 there was added several new features to MQTT while keeping most of the core in place. These major functional objectives are:

- Enhancements to scalability and large-scale systems

- Improved error reporting
- Formalize common patterns including user capability discovery and request response.
- Extensibility mechanisms including user properties
- Performance improvements and support for small clients

[21]

3.3.2.2 How does MQTT work?

3.3.2.2.1 The publish and subscribe model.

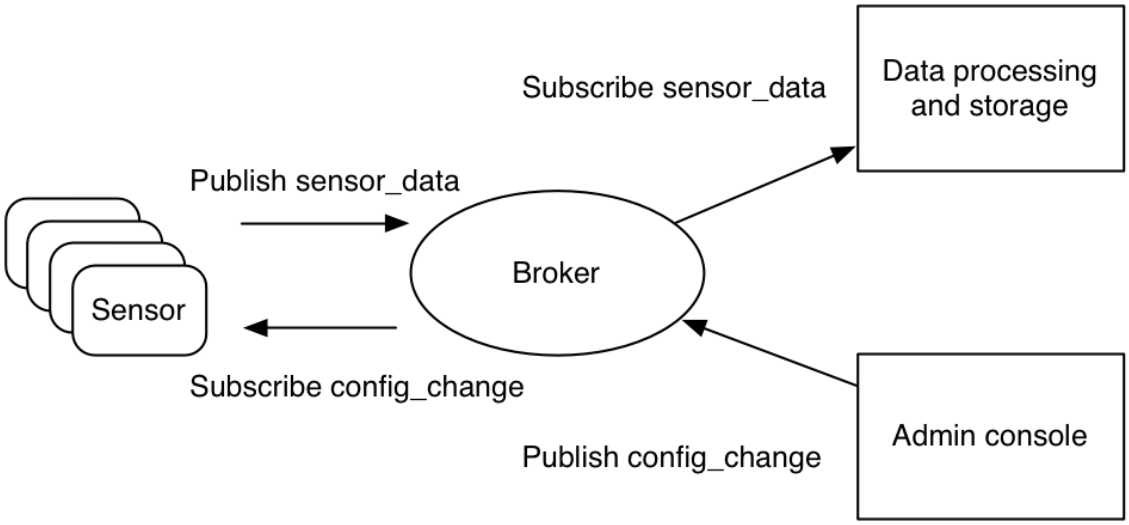

Figure 16. The MQTT publish and subscribe model for IoT sensors. [22]

A key feature of the MQTT protocol is its publish and subscribe model, where it decouples the publisher and consumer of data. In the MQTT protocol there is defined two types of entities in the network, a message server (formerly known as broker) and a number of clients. The server receives and routes messages from the clients to the correct destination. A client is anything that can interact with the server to send and receive messages. A client could be anything between an IoT sensor in the field or an application that process data.

This is done in a certain way:

1. The client attempts to connect to the server. Where it can subscribe to any kind of

“message-topic” from the server. This connection can either use TCP/IP connection for the normal messages or it can use the Transport Layer Security (TLS) connection for more sensitive messages.

2. The client will then publish messages under a certain topic by sending both the message and the topic to the server.

3. The server will then forward the message to every client that has subscribed to the given topic.

Messages sent with MQTT is organized by topic, which gives application developers the possibility to state that only certain clients can interact with certain messages. As an example, sensors will publish readings under the topic “sensor_data” and subscribe to the topic “config_change”. Then applications that process data and save sensor data into a backend database will subscribe to the topic “sensor_data”. An admin console application could receive system admin’s commands to adjust sensor configurations, such as sensitivity and sample frequency, and then publish those changes to the "config_change" topic. [22]

3.3.2.2.2 Understanding the MQTT protocol

The MQTT protocol is a wire protocol, which specifies how data bytes should be arranged and transferred over the TCP/IP network [22]. Developers do not need to understand the wire protocol, but rather need to know that each message has a command and data payload. Those commands specify the message type (for example, a connect or subscribe message). Every MQTT library and tools provide ways to operate those messages with simple course of action directly and can automatically inhabit/fill some required fields, such as message and client-ID.

As described in "3.3.2.2.1 The publish and subscribe model., the client will have to connect to the server first by dispatching a CONNECT message. The CONNECT message will request to set up a connection from the client to the server. The CONNECT message has the following content parameters.
[22]

Parameter	Description
cleanSession	This flag specifies whether the connection is persistent or not. A persistent session stores all the subscriptions and potentially missed messages in the server
username	The authentication details of the server
password	The authentication details of the server
lastWillTopic	If the connection is dropped unexpectedly, the server will automatically publish a “last will” message to a topic
lastWillQos	The “last will” message’s QoS
lastWillMessage	The “last will” message itself.
keepAlive	The time interval the client needs to ping the server to keep a connection alive

Table 2. CONNECT message parameters [22]

The client will obtain a CONNACK message from the server. The CONNACK message has the following parameters.

Parameter	Description
sessionPresent	Indicates whether the connection already has a persistent session. Which is if the connection already has subscribed topics and will receive the delivery of missed messages.
returnCode	0 indicates success. Other values identify the cause of the failure

Table 3. CONNACK message parameters [22]

After an established connection, the client sends one or more SUBSCRIBE messages to the server, where it indicates which topics it wants to receive messages from. That message can have one or multiple repeats of the following parameters.

Parameters	Description
QoS	The QoS (quality of service) flag indicated how consistently the message under this topic needs to be delivered to the clients. It has certain values: **Value 0**: Unreliable, the message will be delivered almost instantly, where the client will miss the message if its unavailable

	Value 1: the message should be delivered at least once. **Value 2:** the message should be delivered exactly once
topic	A topic to subscribe to. A topic can have multiple levels separated by the slash character.

Table 4. SUBSCRIBE message parameters [22]

When a client has subscribed to a topic, the server will return a SUBACK message with one or more returnCode parameters.

Parameter	**Description**
returnCode	Every topic in the subscribe command has a return code. The value of the return code is: **Value 0-2:** Success with the corresponding QoS level **Value 128:** Failure

Table 5. SUBACK message parameters [22]

Corresponding to the SUBSCRIBE message, the clients can also UNSUBSCRIBE from one or more topics.

Parameter	**Description**
topic	Can be repeated for multiple topics

Table 6. UNSUBSCRIBE message parameters [22]

The client can send PUBLISH messages to the server. The message contains a topic and payload. The message is then forwarded to all the clients that has subscribed to that topic. [22]

Parameter	**Description**
topicName	Decides what topic the message is published under.
QoS	The quality-of-service level of the message delivery.
retainFlag	Indicates whether the server will hold on to the message as the last know message of the topic.
payload	The actual data in the message. Whether it's a text string or binary data

Table 7. PUBLISH message parameters [22]

[22]

3.3.2.2.3 Structure of an MQTT control packet

The MQTT protocol works by exchanging series of MQTT Control Packets in a certain way. This section will describe the format of these packets.

A MQTT Control Packet consists of up to three parts, which are always in the following order: Fixed header, which is present in every control packet. Variable Header, which is present in some control packets. And last the payload, which is present in some control packets.

Fixed Header

Each MQTT Control Packet contains a Fixed Header as shown below.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
Byte 2	Remaining length							

Table 8. Structure of an MQTT Control Packet [21]

MQTT Control packet type

Positioned in byte 1 of the fixed header, bits 7-4

Name	**Value**	**Direction of flow**	**Description**
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgement
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgement (QoS 1)
PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 Delivery part 2)

PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS 2 Delivery part 3)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgement
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgement
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

Table 9. MQTT Control Packet types [21]

Flags

Bit 3 to 0 in the fixed header contains flags that is specific for every MQTT control packet. Flags that are linked to the Publish-Control Packet from MQTT v5 are set to decide three different parameters, DUP, QoS and Retain. Flags from the other types of Control Packets are reserved for use in the future and need to be set to the value that is specified in the documentation.

DUP: If the DUP flag is set to 0 it indicates that this is the first case of the client or server is trying to send the Control Packet. If the flag is set to 1 it indicates that it could be a new attempt to send a Control Packet that was attempted sent earlier.

QoS: This flag decides what Quality of Service level is present.

Retain: If the Retain flag is set to 1 in a publish control packet that the client is sending to the server, then the server needs to change pre-existing messages that is kept for the applied topic and save the message so it can be delivered to future subscribers of the topic. If the Retain flag is set to 0 in a control packet that is published, then the server will not keep the message and it will not change the pre-existing message. [21]

Variable Header/Remaining length

The Remaining Length is a Variable Byte Integer that represent the number of bytes remaining within the current Control Packet, including data in the Variable Header itself and the Payload. However, it does not include the bytes used to encode the Remaining Length itself. The packet size is the total number of bytes in a Control Packet, which is equal to the total length of the Fixed Header and the Remaining Length.

Not every control packet includes a variable header, and the content is dependent on what kind of packet it is, but the packet-identification is common for different types of packets. The header includes two fields, those being packet-identification and properties.

The Packet Identifier is often included in many of the MQTT Control Packet types and includes a Two Byte Integer Packet Identifier field. These control packets are PUBLISH where QoS > 0, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE and UNSUBACK.

The field for properties consists of an identifier that defines the use-case of the packet and the datatype, followed by a value. [19]

Payload

Some MQTT control packets contain a payload as the last part of the packet. This is the application message. Control packets that contain a payload is listed below.

MQTT Control Packet	**Payload**
CONNECT	Required
CONNACK	None
PUBLISH	Optional
PUBACK	None
PUBREC	None
PUBREL	None
PUBCOMP	None
SUBSCRIBE	Required
SUBACK	Required
UNSUBSCRIBE	Required

UNSUBACK	Required
PINGREQ	None
PINGRESP	None
DISCONNECT	None
AUTH	None

Table 10. MQTT Control Packets that contain a Payload [21]

[19]

3.3.2.2.4 Quality of Service (QOS)

MQTT has three different levels/values of Quality of Service. Those levels are as follow:

Value 0 – The message is delivered after the capability of the network. No response is sent from the receiver to the sender and there are no attempts from the sender to send the message again to the client. The message will either get to the receiver once, or not at all.

Value 1 – Ensures that the message is received at least once.

Value 2 – The highest Quality of Service level. Which is used when neither loss nor duplication are acceptable.

For example, MQTT messages that defines commands that enable certain operations, value 2 would be the way to go. This would secure that the command gets through, and that the operation will not be enabled more than it is supposed to (duplicates).
[21]

3.3.2.2.5 Security

For clients and units connected to a network, security is one of the most important factors, therefore also for MQTT. The MQTT-server is recommended to have a system for authentication of clients and users, with a client-ID, certificates, and passwords. It is also recommended that that all communication is encrypted, and that there are methods to

uncover attempts at uncertified connections to the MQTT-system, or in any other way of trying to manipulate the dataflow between clients and servers.

MQTT can use SSL/TLS to secure communication between units and give the opportunity for encryption. TLS and SSL are two protocols for secure communication and is located in the TCP/IP- layer. It is possible to utilize TLS/SSL together with other reliable communication safety- protocols, but not with unreliable like for example UDP. TLS v1.0 was released in 1999 and were built on SSL v3.0. TLS v1.3 is from 2018 and is the newest version of the TLS protocol. There is two main parts of TLS, to reach a secure connection there is one “handshake” protocol and one “record” protocol. The handshake protocol is when the client and server is authenticating by sharing certificates and agreeing on encryption keys. The record protocol takes the data that is going to be sent, fragmenting them in smaller parts then encrypting these with help from the encryption keys, and lastly sending the data. In the opposite end the data is decrypted and put back together. [23]

The different ports MQTT is communication with, depending on if its unencrypted, encrypted or with encryption and certificates:

1883: MQTT, unencrypted

8883: MQTT, encrypted

8884: MQTT, encrypted, client certificate required
[23]

3.3.2.3 Advantages of MQTT

- **Packet agnostic:** Any type of data can be transferred in the payload carried by the packet. As mentioned earlier the data could be text or binary. As long as the receiving end know how to interpret the data it does not matter.
- **Reliability:** There are several Quality of Service values that can be enabled to make sure the message is delivered.
- **Scalability:** The publish and subscribe model scales well in a power efficient way.

- **Decoupled design:** There are several elements to the design that decouple the device and subscribing server, which results in a more though communication plan.
- **Time:** A device can publish its data regardless of state of the subscribing server. Which enables the subscribing server to connect and receive the data whenever it is able to. This decouples both ends of the communication on a time basis. This allows devices to remain in a sleepy state while not worrying about if the subscriber is able to receive the data when it wakes up to communicate.
- **Space** (delivery details)**:** The publisher needs to know the server IP address and how to communicate with it, but it does however not need to know anything about the subscribers, nor do the subscribers need to know anything about the network connection details of the publisher. This makes for smaller communication overheads on the device side. It also frees both ends to be able to operate independently. Subscribers can be upgraded and changed, and additional subscribers can be added without any change needed on the publishing device end.
- **Synchronizing:** Neither side must pause what they are doing to communicate or receive a message. This process is asynchronous and there is no need to interrupt any work in progress to publish a message.
- **Security:** MQTT offers the option of TLS/SSL encryption of communication
- **Bidirectional:** A device can be both a publisher and a subscriber. In this way, it can receive commands by subscribing to a topic on the server. It could also receive data from other devices that could be input into the data it publishes.
- **Maturity:** MQTT was invented by IBM with the first protocol draft in 1999. MQTT was released royalty free in 2010 and has since then been donated as an open source project to the Eclipse foundation. It is in use by millions of connected products today.

  [24]

3.3.2.4 Limitations of MQTT

- **Operating over TCP:** TCP was designed for devices with more memory and processing power than many lightweight, power constrained IoT devices. TCP requires more “handshakes” to be able to set up communication links before any messages can be exchanged. That will increase wake-up and communication times, which again will affect long term battery consumption. TCP connected devices tend to keep sockets open for each other with a persistent session. This adds to power and memory requirements.
- **Centralized server can limit scale:** The server can affect a scalability since there is need for additional overheads for each device connected to the server. The network can only grow as large as the server can support. This puts limits on expansion for each hub and spoke group.
- **Server single point of failure:** The server can also be a single point of failure in the network. Commonly the server is a device that is plugged into a wall socket with several battery powered publishing devices. If there is a power failure the server will be offline whilst the publishing devices will still be operating, rendering the network useless until the power is restored.
- **Security:** By default, MQTT is unencrypted. This makes it natively unsecure and requires additional steps and some overhead absorption to make sure TLS/SSL is implemented.

  [24]

3.3.2.5 Mosquitto – an open source MQTT server

Figure 17. Mosquitto logo [25]

Mosquitto is a message server for MQTT protocols v.5.0 and v.3.1.1. It is open source, requires few resources and can be utilized on all devices from microcontrollers to full servers. Mosquitto uses a C-library for implementing MQTT clients. [25]

The MQTT server Mosquitto is a vital part, as it handles almost all the communication in the back-end system.

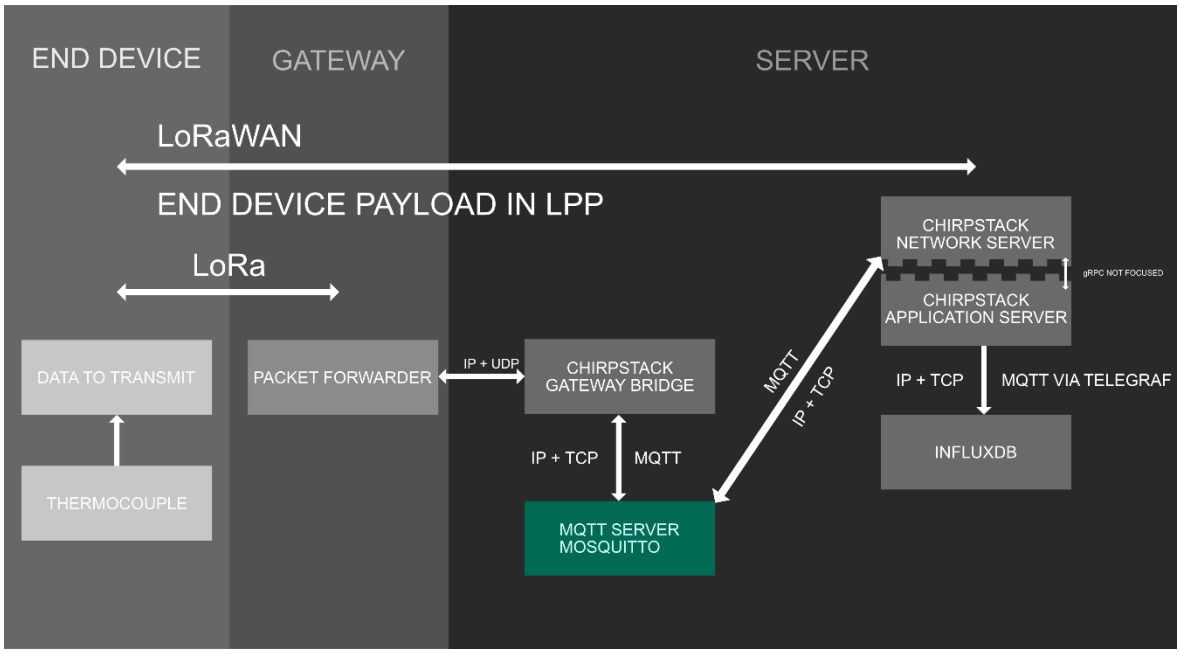

Figure 18 Mosquitto working as MQTT server

Mosquitto allows research to be done directly related to the MQTT-protocol, such as performance and investigating the use of authenticators. It also supports other research activities, like a block that is useful when larger systems are being built, and it is also evaluated for use in smart cities. [26]

3.3.5 Cayenne Low Power Payload (LPP)

The Cayenne Low Power Payload (LPP) provides a structured way of transmitting data. This protocol is used in the transmission of payload data in the project implementation in chapter ”4.2.2 Prototype 2”. Using this protocol provides the back-end with the capability to ascertain what kind of data the payload contains. The protocol is compliant with payload size restrictions and can be lowered down to 11 bytes. LPP allows for transmitting multiple sensor data at one time. It is also possible to send different sensor data in different frames. This is done by prefixing each sensor data with two bytes. These two bytes are:

- The data channel, which uniquely identifies each sensor in a device across frames. For example, “outdoor sensor” or any sensor defined to belong to a specific channel.
- Data type, which specifies the type of data that is being transmitted. For example: temperature, humidity, pressure or other data types.

The structure of the payload is as follows:

1 byte	1 byte	N bytes	1 byte	1 byte	M bytes	Continues
Data 1 Channel	Data 1 Data type	Data 1 Data	Data 2 Channel	Data 2 Data type	Data 2 Data	→

Table 11 Cayenne Low Power Payload structure

Each data type can use one or more bytes for transporting data. For example, the payload can look like this:

1 Byte	1 Byte	2 bytes	1 byte	1 byte	1 byte
Data channel 1	Temperature sensor	Data	Data channel 4	Humidity sensor	Data
0x01	0x67	2 bytes data	0x4	0x68	1 byte data

Table 12 Example of payload structure

Here the temperature sensor uses 2 bytes of data, because this is specified in the LPP. The different identifiers for the data types are specified to conform to IPSO Alliance Smart Objects Guidelines [27]. Here each data type is identified with an Object ID. These object

IDs have been converted in the LPP to fit in a single byte. The way the LPP defines this conversion is “LPP_DATA_TYPE = IPSO_OBJECT_ID – 3200”. [28]

The following table specifies these LPP data types:

Type	IPSO	LPP	Hex	Data Size	Data Resolution per bit
Digital Input	3200	0	0	1	1
Digital Output	3201	1	1	1	1
Analog Input	3202	2	2	2	0.01 Signed
Analog Output	3203	3	3	2	0.01 Signed
Illuminance Sensor	3301	101	65	2	1 Lux Unsigned MSB
Presence Sensor	3302	102	66	1	1
Temperature Sensor	3303	103	67	2	0.1 °C Signed MSB
Humidity Sensor	3304	104	68	1	0.5 % Unsigned
Accelerometer	3313	113	71	6	0.001 G Signed MSB per axis
Barometer	3315	115	73	2	0.1 hPa Unsigned MSB
Gyrometer	3334	134	86	6	0.01 °/s Signed MSB per axis
GPS Location	3336	136	88	9	Latitude : 0.0001 ° Signed MSB
					Longitude : 0.0001 ° Signed MSB
					Altitude : 0.01 meter Signed MSB

Table 13 LPP data types [28]

The LPP protocol is understood by both The Things Network (TTN) [29] and ChirpStack [30] and is therefore suitable for this project.

3.4 ChirpStack - An open source LoRaWAN server solution

Figure 19ChirpStack logo [31]

To get a better understanding of how a LoRaWAN system works behind the gateway (the back-end system), it is relevant to look at a LoRaWAN server system. This chapter will look at the ChirpStack LoRaWAN server stack. Chapter “4.3” will look at a practical implementation. The ChirpStack project is open source [32] and supported by a large community [33]. The ChirpStack project provides components for LoRaWAN networks. It offers a solution which is ready to use. All of its components are also licensed under the MIT license and can be used for commercial purposes [32].

The application server part has a user-friendly web-based interface for managing devices and application programming interfaces (APIs) for integration. The ChirpStack project has modular architecture which makes it possible to integrate with other existing infrastructures.

The components of the ChirpStack are:

- ChirpStack Gateway Bridge: handles the communication with the LoRaWAN gateways
- ChirpStack Network Server: a LoRaWAN network server implementation
- ChirpStack Application Server: a LoRaWAN application server implementation

- ChirpStack Gateway OS: Linux-based OS to run the (full) ChirpStack stack on a Raspberry Pi based LoRa gateway
  [32]

The relation between the components can be viewed in “Figure 20 The ChirpStack architecture overview ”.

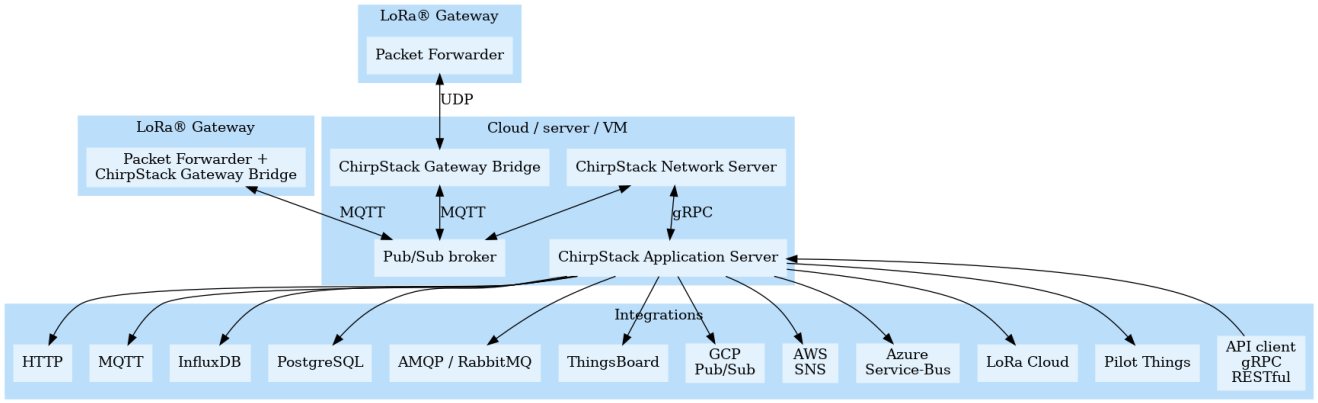

Figure 20 The ChirpStack architecture overview [34]

The gateway communicates with the network server via a packet forwarder, the gateway bridge and the MQTT protocol. The gateway bridge can be deployed in different ways (see chapter “3.4.1.1”) and depending on the type of bridge deployment there are two different ways the packet forwarder communicates with the gateway bridge. If the gateway bridge is installed on the server, the packet forwarder sends LoRaWAN frames to the gateway bridge via the UDP protocol. If the gateway bridge is directly installed on the gateway itself the UDP protocol is no longer needed and the UDP communication of LoRaWAN frames is replaced by TCP and the MQTT protocol.

The network server communicates with the application server via gRPC. gRPC is an open source Remote Procedure Call (RPC) framework. It connects services in across and across data centres [35]. gRPC is not a focus in this thesis. The application server can be integrated with several services for storing or processing the data.

3.4.1 ChirpStack gateway bridge

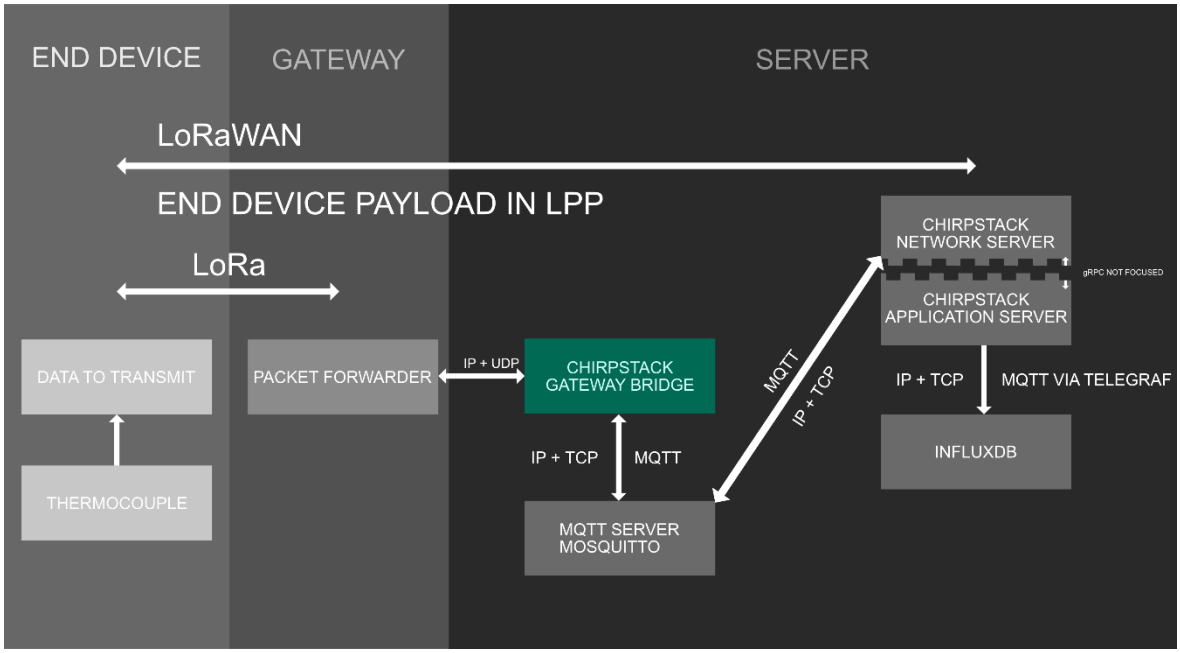

Figure 21 Gateway bridge - communicates with gateway and network server via Mosquitto and MQTT [5].

The ChirpStack gateway bridge is a part of the ChirpStack LoRaWAN network server stack. It sits between the packet forwarder and an MQTT server [34]. The gateway bridge is a service that converts LoRa packet forwarding protocols into a data-format which the ChirpStack network server uses, either JSON [36] or Protobuf [37]. The gateway bridge supports three different packet forwarders. These are ChirpStack Concentratord, Semtech UDP Packet Forwarder and Basic Station Packet Forwarder. The gateway in this project uses the Semtech UPD packet forwarder. There is also support for integration of other services such as MQTT, Google Cloud IoT Core and Azure IoT Hub MQTT bridge.

[38]

3.4.1.1 Deployment

There are several different ways in which the ChirpStack gateway bridge can be deployed. It can be installed as a single instance, on multiple servers or it can be installed on each gateway. In this project, the gateway bridge is deployed as a single instance. For details on the other possible methods of deployment, see appendix 2.

For a single instance, the ChirpStack gateway bridge is installed on one server and all gateways connect to the bridge. This is the easiest way to deploy the bridge, but it is also the least secure. This is because of the UDP protocol that most gateways use, does not support any authorization and there is no way to check if the data that is received is authentic [39].

3.4.1.5 Dependencies

ChirpStack gateway bridge uses MQTT for publishing events and receiving commands. The bridge requires a MQTT server to be installed, for example by using Mosquitto. Other options could work as long as they support MQTT v3.1.1.
[40]

3.4.2 ChirpStack network server

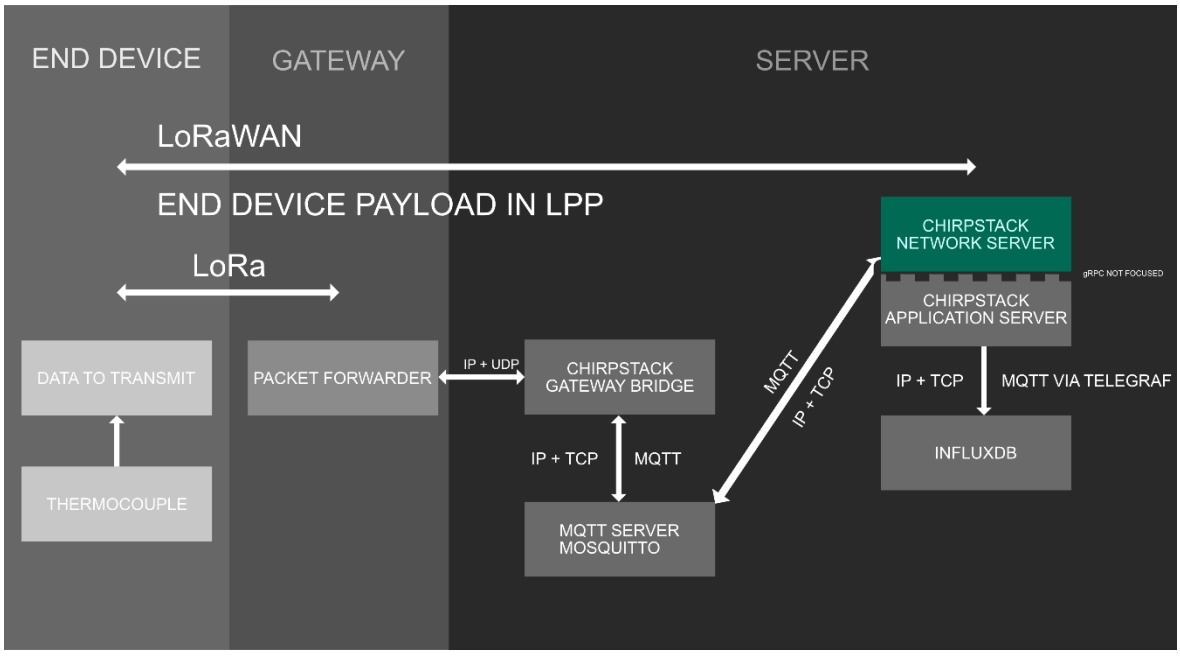

Figure 22 Network server - part of the system implementation [5].

The ChirpStack network server is a LoRaWAN network server, which is responsible for managing the network. This means it has several responsibilities. The server has knowledge of device activations and handles join-requests when devices want to join the

network. The network server is also responsible for de-duplicating data from LoRaWAN frames received by LoRa gateways. [41]

The LoRaWAN frames handled by the network server:

- Authentication
- LoRaWAN MAC-layer (and MAC-commands)
- Communication with the ChirpStack Application Server
- Scheduling of downlink frames

  [41]

The network server has a lot of features. The features utilized in this project are device activation (OTAA), adaptive data rate (ADR), device classes (class A), device profiles, gateway management (gateway location by GPS), LoRaWAN versions (version 1.0.X), LoRaWAN regions (EU) and service profile (features for devices). Details on these features and the other supported by the network server are covered in appendix 3.

3.4.2.18 ChirpStack network server dependencies

The network server requires an MQTT server for publishing and receiving application payloads. It has the same requirements as the Gateway bridge which requires MQTT v3.1.1 or above.

For handling the gateway data, the ChirpStack network server also requires two different types of databases. A PostgreSQL database for gateway data, and a Redis database for storing device-session data and non-persistent data, such as distributed keys, deduplication sets and meta data. PostgreSQL above v9.5 and Redis above v2.6.0 is required. For any installation of the server on Ubuntu or Debian OS, the compliant versions are Ubuntu v18.04 and Debian v10.
[42]

3.4.3 ChirpStack application server

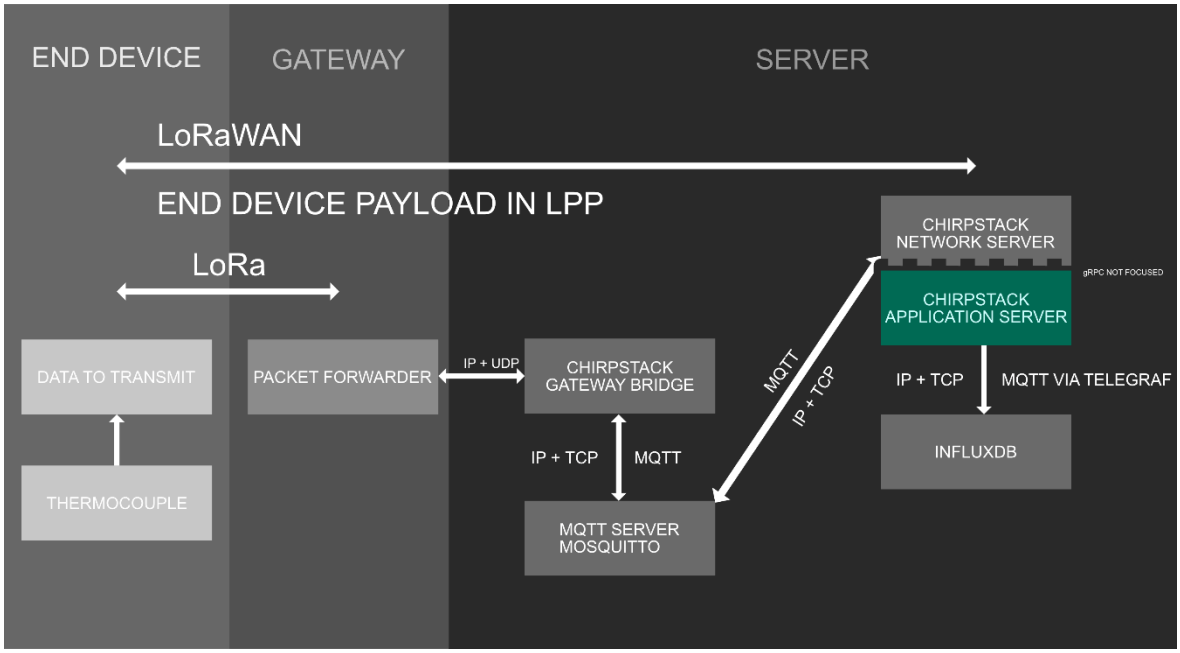

Figure 23 The application server part of the system [5].

The ChirpStack application server is part of the open source LoRaWAN network server stack. It is responsible for the devices in a LoRaWAN infrastructure, handling join requests and encryption of application payloads. The application server has a web-based interface where it is possible to manage users, organizations, applications, and devices.

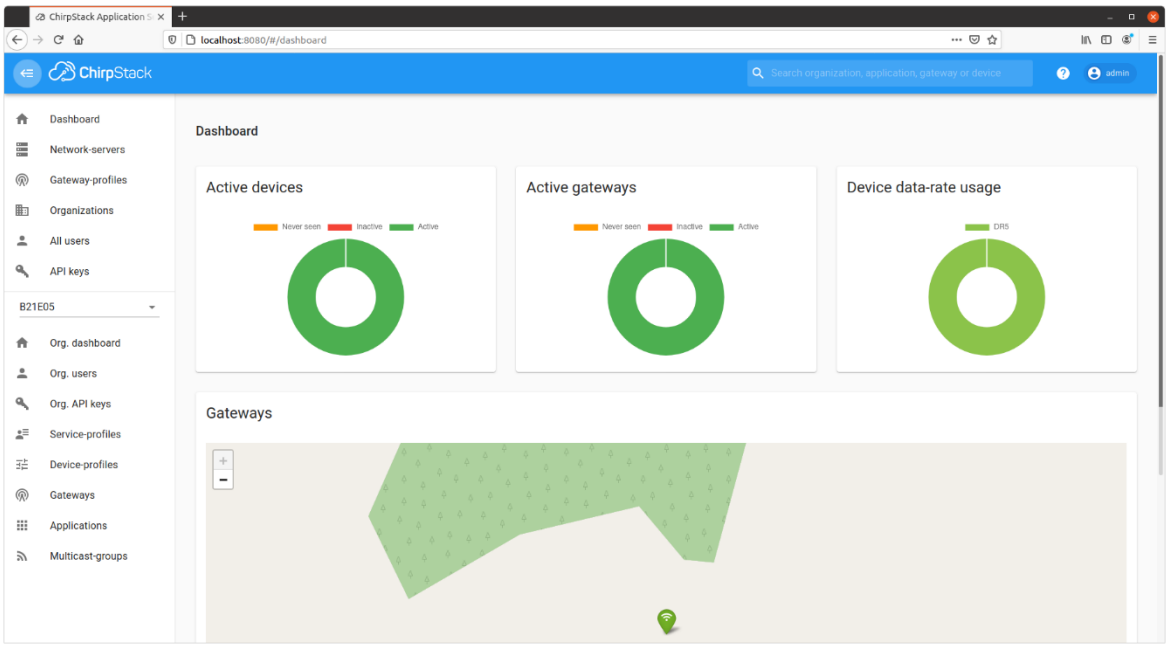

Figure 24 This is what the web interface for the application server looks like

The application server has integrations with external services via the use of gRPC and RESTFUL API [43]. Data from the application server can be sent or received by using MQTT or HTTP and it can be written to a InfluxDB database.
[44]

The application server has features which are listed in the following chapters. Of these features only gateway discovery and firmware update over the air was not used. Gateway discovery was not used due to only implementing one gateway, and firmware update over the air is only available as an experimental feature [45].

3.4.3.1 Payload encryption and decryption

The application server handles encryption and decryption of the application payloads. The application key for each device and the join-accept when Over The Air Activation (OTAA) is used. The payloads will be sent decrypted to the integrated services but are not decrypted before the network server. The network server does not have access to these payloads.
[46]

3.4.3.2 User authorization

The ChirpStack application server makes it possible to grant users global administrative rights, making them the admin or assigning them view-only permissions within an organization. This makes it possible to run the application server in an environment where there are several different user-categories, and where each organization or team only have access to their own applications and devices.
[46]

3.4.3.3 Application programming interface (API)

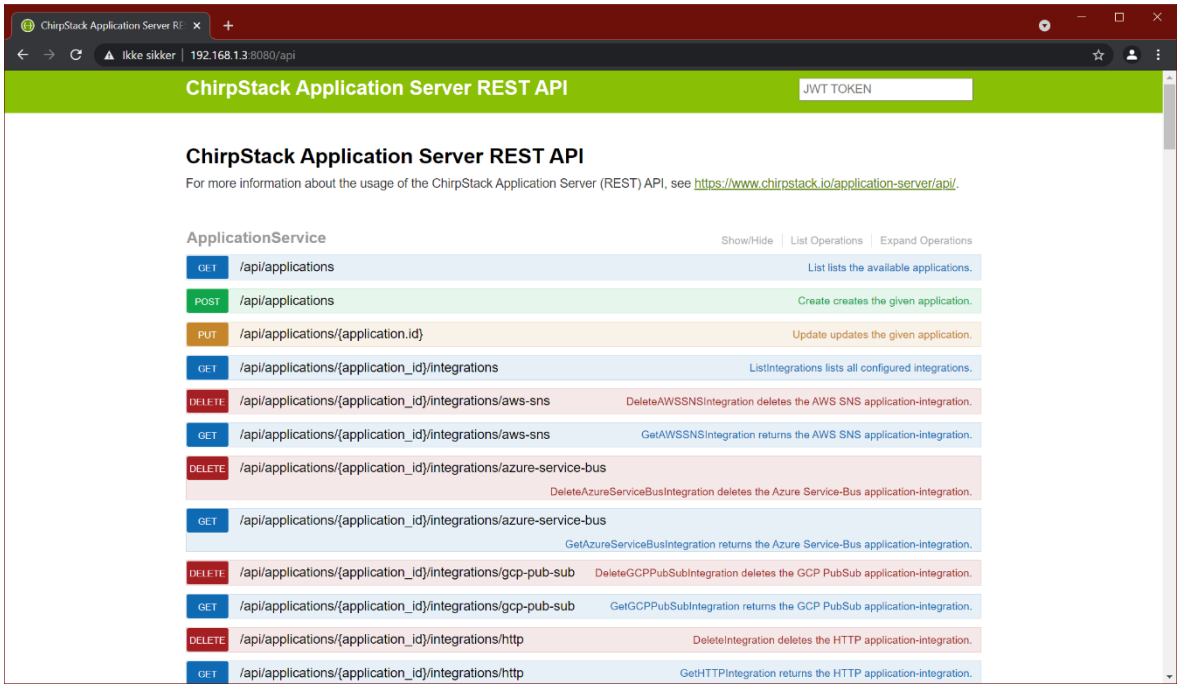

Figure 25 Access to API through the web interface

For integration with external services the application server provides the use of gRPC and RESTFUL API, which has the same functionality as the web interface.
[46]

3.4.3.4 Gateway discovery

In networks that contain several gateways it is necessary to test the network for gateway coverage. The application server can periodically send out pings through each gateway to discover how well these pings are received by other gateways in the same network. This way the application server can map out the network. Then the application server can display this information as a map in the web-interface. This feature can be enabled through configuring the network server.
[46]

3.4.3.5 Live frame logging

It is possible to inspect all raw, encrypted LoRaWAN frames per gateway or device. When viewing the LoRaWAN frames option in the application server for a gateway or for

devices, one can see all frames that pass in real-time. It also allows for inspection of the (encrypted) content of each LoRaWAN frame.
[46]

3.4.3.6 Live event logging

From the web interface on the application server it is possible to inspect all events without the need of an MQTT client or build-integration. From the “live event logs” menu option under detailed page for devices, one can see all uplink, acknowledgement, join or error events in real-time.
[46]

3.5 Telegraf

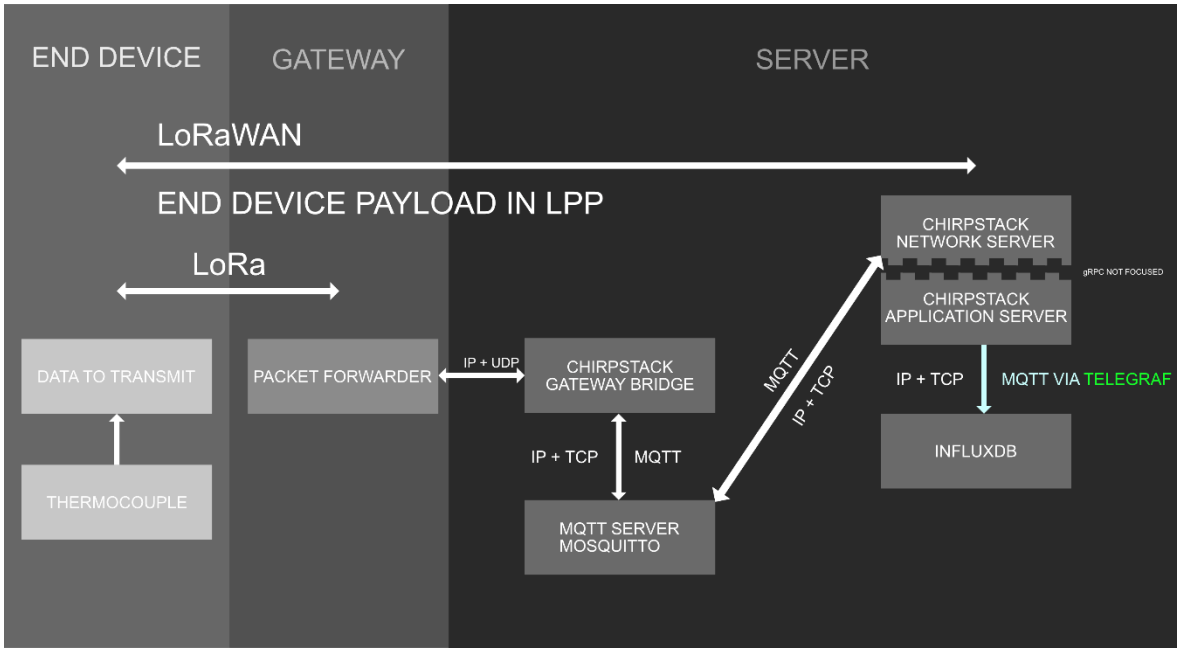

Figure 26 Telegraf used for conveying data from the application server to InfluxDB [5].

Telegraf is a server agent for collecting and sending metrics and events from systems, databases and IoT sensors. Telegraf is plugin driven, meaning that it allows for new inputs and outputs to easily be added. Telegraf is used in this projects implementation for sending measurement data from the ChirpStack application server into InfluxDB, via an MQTT plugin (see chapter “4.5.1”).

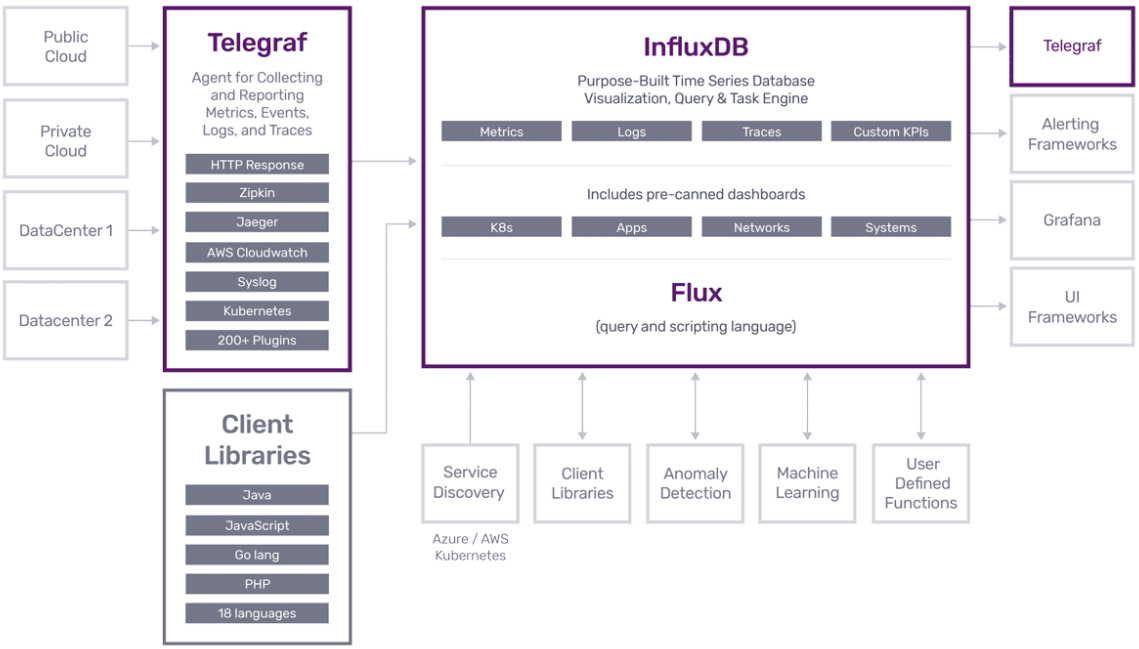

Figure 27 Overview of how Telegraf relates to other services [47]

There are many input integrations for various metrics, events, and logs from popular systems. The output plugins allow sending metrics and a variety of different datastores, services and message queues. Some of these are InfluxDB, Graphite, OpenTSDB, MQTT, NSQ and several others.
[47]

Key features of Telegraf are:

- It is written in Go. It compiles into a single binary and have no external dependencies.
- It has very little memory usage.
- The plugin system allows adding new inputs and outputs for many services, and many plugins exist for popular services.
  [48]

3.6 InfluxDB

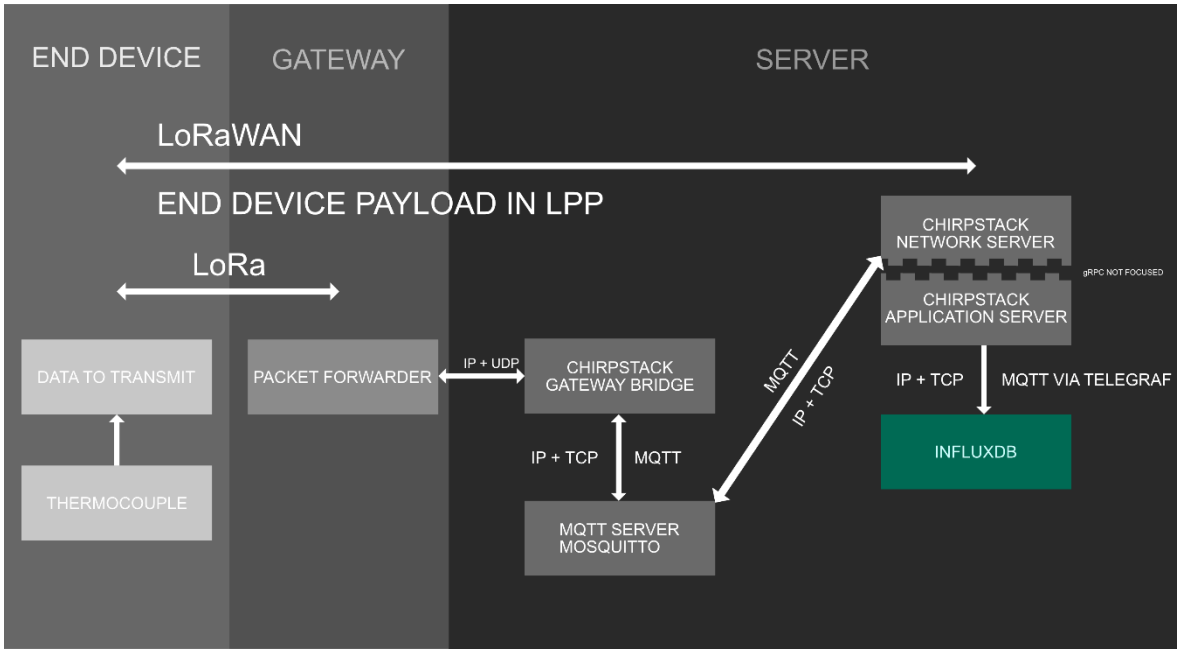

Figure 28 InfluxDB as component in the back-end [5].

InfluxDB is a time series platform. Timeseries data means a sequence of data that consists of successive measurements made from the same source over a period of time [49]. InfluxDB comes in three different versions, InfluxDB cloud, InfluxDB Enterprise and InfluxDB Open Source. This project implements InfluxDB open source for data storage, visualisation, and processing. InfluxDB open source can be used for free. It is different from a traditional database as it can handle new types of workloads, millions of data points, and hundreds of data sources. Flux, which is the scripting and query engine, is designed to transform this timeseries data into information and enable real-time decisions. [50] Flux is designed for data scripting, monitoring, and alerting [51].

Timeseries data collected is stored in buckets. Each bucket can collect from multiple sources. There are no limitations on the number of sources data is sourced from, the buckets job is only to store the data. To identify the source of data, a tag can be applied for the data points for identification. [52] Buckets have data retention periods, meaning each bucket can be defined to hold on to its data for a specified amount of time. The time for data retention is calculated from the data timestamp. [53] [54]

InfluxDB has a web-based user interface that includes a data explorer, dashboard tools and script editor. The data explorer can be used to browse through metric and event data

collected, and common data transformations ca be applied. [51] Data can be processed an analysed using tasks in the InfluxDB task engine. Tasks are scheduled Flux queries to input data streams for analyzation, modification and acted upon accordingly. [55]

The user interface also provides tools for visualizing the data. This is done by creating dashboards with cells that have many different types of visualisations, such as band, gauge, graph, heatmap, histogram and others. This makes it possible to visualize the data in the way that makes the most sense for the application. [56] [57]

Data can be monitored, and alerts can be sent based on checks defined in a query. A check queries data and assigns a status with a level argument based on the specific conditions. Notification rules can be assigned, and notifications can then be created based on these rules. [58] The notification rules send a message to a notification endpoint [59]. The possible notification endpoints are Slack, PagerDuty and a HTTP endpoint [60]. It is also possible to set up a task for sending an email alert [61].

3.7 Measuring ultra-low temperatures

In this chapter, the measurement of ultra-low temperatures (ULT) will be looked at since the vaccines has storage demands of -70°C with a margin of ±10°C. There are several methods to ensure that such low temperatures could be measured:

- **Diode Thermometer:** Diodes can be used as thermometers as silicon diodes have a linear ratio between the temperature and the voltage.
- **Capacitance thermometer:** These are used for measuring ultra-low temperatures but is most suited for places with high variations in the magnetic fields.
- **Semiconductor temperature sensors:** These sensors are different from manufacturer to manufacturer but share multiple characteristics. These characteristics are low-cost, linear output and small size. Some disadvantages are that they require calibrating, and that the semiconductor chip often has bad thermal contact with an outside surface.
- **Metallic Resistance Thermometers:** The working principal of these are that the resistance change in the metallic conductor when the temperature changes. The

temperature may be calculated by utilizing the measured resistance variation. These are usually expensive.

- **Thermocouple:** A thermocouple is a simple sensor able to measure both ultra-low and ultra-high temperatures. It is both low cost and very solid, making it easily replaceable if damaged.
  [62] [63]

The thermocouple was the option that was implemented in this project.

3.7.1 Thermocouple technology

A thermocouple is made up by two wires where each of the wires are made from different metals. A thermocouple consists of a least two junctions, a reference junction (cold junction) and the measurement junction (hot junction). The cold junction is the point where the wires are connected to the measuring instrument. In this point the metals change from the thermocouple metals to copper or similar metals which is commonly used in most measuring instruments.

The output from a thermocouple is a voltage proportional to the temperature difference between the cold - and the hot junction. The generated voltage is both small and nonlinear.

Most wires used in thermocouples are alloys and their homogeneity is vital to the measuring systems accuracy. [62]

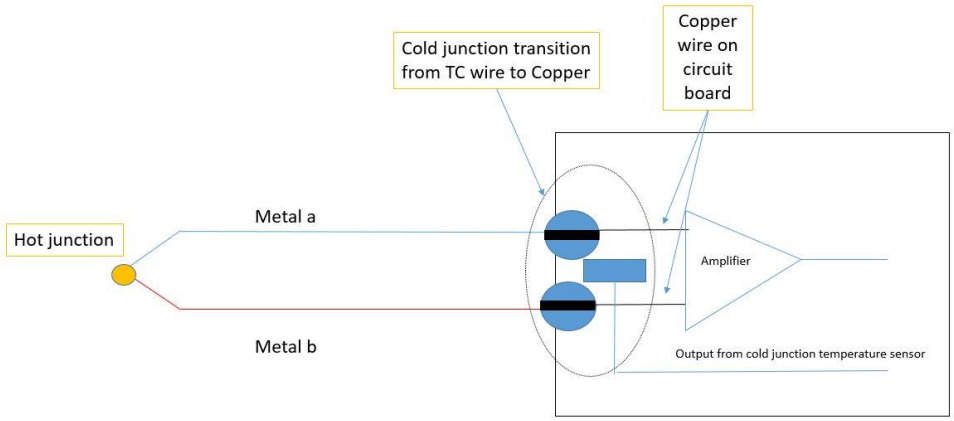

Figure 29 Thermocouple connected to an amplifier [62]

3.7.2 Thermocouple types

There are two basic groups of thermocouples, it is those constructed from noble metals and those from base metals.

The base metals are typically lower cost elements such as copper, nickel, or iron alloys. The ratio of metals in the alloys may differ from manufacturer to manufacturer as these are not defined, however the voltage/temperature response curves are.

In noble metal thermocouples both the voltage/temperature response curves and the purity levels are defined due to the use of more expensive elements such as Platinum and rhodium. [62]

Thermocouple type	Use range in Celsius (excluding limitation of wire insulation)	Wire composition
Type K	-270 to 1370	Ni-Cr (+ve) and Ni-Al
Type N	-270 to 1300	Ni-Cr-Si (+ve) and Ni-Si-Mg
Type T	-270 to 400	Cu (+ve) and Cu-Ni
Type J	-210 to 1200	Fe (+ve) and Cu-Ni
Type E	-270 to 1000	Ni-Cr (+ve) and Cu-Ni
Type R	-50 to 1768	Pt13% Rh (+ve) and Pt
Type S	-50 to 1768	Pt10% Rh (+ve) and Pt
Type B	0 to 1768	Pt30% Rh (+ve) and Pt

Table 14. Some commonly used thermocouple types and their use range [62]

3.7.3 Thermocouple instrumentation

To read the measurements of a thermocouple, an amplifier is needed as the voltage can be as low as 4µV/C (microvolt per degree Celsius). As the generated voltage is not completely linear, a reference voltage in the amplifier is used for compensation. There are different kinds of amplifiers such as digital and analog, where the analog is the easiest and the digital is the most accurate. [62]

3.8 Power consumption

This chapter will give an overview of the power consumption issues when designing a microcontroller system. There are loads of factors that contributes to the overall power consumption, with the two major ones being:

- Operating supply voltage (V_{DD})
- System clock frequency

The product design in this project is built on an Arduino Uno with a LoRa-shield on top. The figure below shows current consumption of an Arduino Uno CPU based on these two factors.

Parameter	Condition	Symbol	Min.	Typ.[2]	Max.	Units
Power supply current[1]	Active 4MHz, V_{CC} = 3V	I_{CC}		1.5	2.4	mA
	Active 8MHz, V_{CC} = 5V			5.2	10	mA
	Active 16MHz, V_{CC} = 5V			9.2	14	mA
	Idle 4MHz, V_{CC} = 3V			0.25	0.6	mA
	Idle 8MHz, V_{CC} = 5V			1.0	1.6	mA
	Idle 16MHz, V_{CC} = 5V			1.9	2.8	mA
Power-down mode[3]	WDT enabled, V_{CC} = 3V				44	µA
	WDT enabled, V_{CC} = 5V				66	µA
	WDT disabled, V_{CC} = 3V				40	µA
	WDT disabled, V_{CC} = 5V				60	µA

Figure 30 Arduino Uno Power Consumption [64]

The LoRa-shield will also contribute to the overall power consumption of the product. The figure below provides an overview.

Symbol	Description	Conditions	Min	Typ	Max	Unit
IDDSL	Supply current in Sleep mode		-	0.2	1	uA
IDDIDLE	Supply current in Idle mode	RC oscillator enabled	-	1.5	-	uA
IDDST	Supply current in Standby mode	Crystal oscillator enabled	-	1.6	1.8	mA
IDDFS	Supply current in Synthesizer mode	FSRx	-	5.8	-	mA
IDDR	Supply current in Receive mode	*LnaBoost* Off, band 1 *LnaBoost* On, band 1 Bands 2&3	- - -	10.8 11.5 12.0	- - -	mA
IDDT	Supply current in Transmit mode with impedance matching	RFOP = +20 dBm, on PA_BOOST RFOP = +17 dBm, on PA_BOOST RFOP = +13 dBm, on RFO_LF/HF pin RFOP = + 7 dBm, on RFO_LF/HF pin	- - - -	120 87 29 20	- - - -	mA mA mA mA

Figure 31 LoRa Shield Power Consumption [65]

The prototypes of this project are designed with an Arduino running with 16MHz and 5V which gives a constant current consumption of 14mA plus the LoRa shield power for consumption.

4 Implementation and results

This chapter will cover the implementation of a prototype and the parts of the back-end system that together creates a complete LoRaWAN system. This chapter will also present results during the implementation process, as different results follow the different parts of the implementation.

Firstly, the hardware utilized will be described. Then the development of prototypes is covered. After that, the prototypes and the gateway are connected to the back-end system. The back-end system is implemented with data storage, data visualisation and notifications when the measurements reach a critical level.

4.1 Hardware

The hardware in this thesis is based on Dragino's IoT Development Kit [66] with the Arduino Uno as a base. The table below shows the hardware used in this project:

Hardware:	Prototype 1:	Prototype 2:
Gateway	LG01-N Single Channel LoRa IoT Gateway	LPS8 Indoor LoRaWAN Gateway
Microcontroller	Arduino Uno	Arduino Uno
LoRa Shield	Dragino LoRa Shield v1.4	Dragino LoRa Shield v1.4
Temperature measurement	DHT11 temperature and humidity sensor	24 AWG K-type thermocouple
		AS8495 K-type thermocouple amplifier

Table 15 List of all the hardware used for prototype 1 and prototype 2.

Prototype 1 was a learning experience to get to know how LoRaWAN works and therefore the hardware was based on a simple development kit from Dragino.

Some of the hardware used in prototype 1 was not deemed fit to reach the requirements for the use-case, therefore a new gateway and temperature sensor was acquired for prototype 2. The gateway was changed as the LG01-N only is a single channel gateway designed for a private LoRa protocol and was not recommended for LoRaWAN use by Dragino themselves [67]. And the temperature sensor was changed to a thermocouple with an amplifier to reach the low temperatures required in the use-case (see chapter 1.4 Use case).

For more in-depth information about the hardware, see appendix 4.

4.2 Prototypes

The prototypes developed are on the Arduino platform. This is mainly because the development of a prototype started with the Dragino IoT kit.

4.2.1 Prototype 1

Prototype 1 is based on Dragino's IoT Development Kit and was used as a learning experience for the team to get to know LoRaWAN and learn the basics.

The prototype is a LoRa Shield combined with an Arduino Uno with a DHT11 temperature and moisture sensor connected to read the surrounding temperature. This data would be sent to the LG01-N gateway that uses The Things Network (TTN) as its network server. As the temperature readings is only shown as a payload in TTN, the application server Cayenne Low Power Payload (LPP) "myDevices" was implemented. See appendix 5 for all the steps taken to setup the prototype.

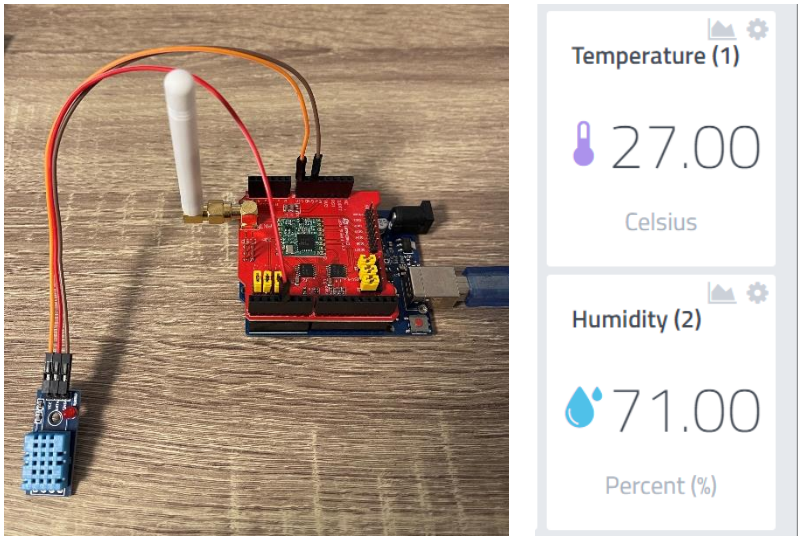

Figure 32 Prototype 1

The prototype however had a few problems regarding the requirements of the use-case:

- Problem 1 is that the gateway was not suited for the use-case (see chapter 5.1 The development of a second prototype)
- Problem 2 is that the DHT11 temperature sensor could not reach temperatures lower than 0°C which is not low enough for the project's requirement.

4.2.2 Prototype 2

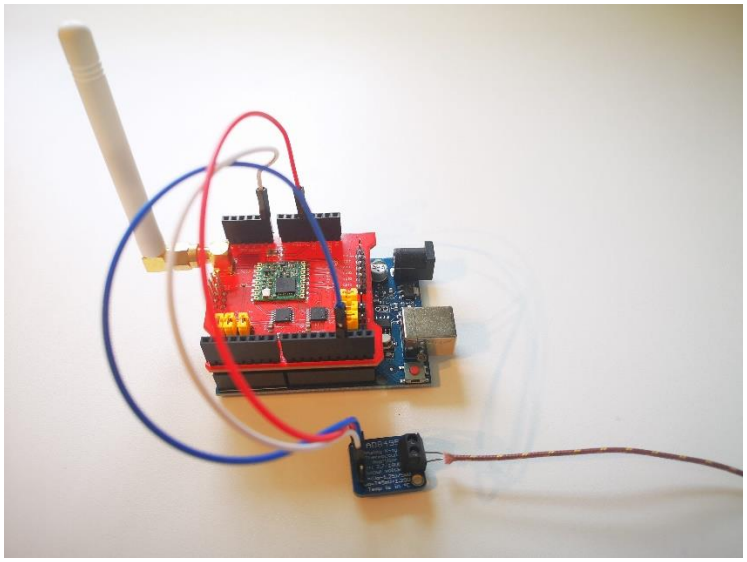

Figure 33 The LoRaWAN prototype with thermocouple

This prototype is based on the hardware for prototype 2 in "Table 15 List of all the hardware used for prototype 1 and prototype 2.", and the Arduino LMIC LoRaWAN library [68], which is compatible with all Arduino boards. It supports LoRaWAN 1.0.2 and 1.0.3 Class A devices, using the Semtech SX1272 or SX1276 radio modules. It can be used in the several LoRaWAN regions. This implementation uses the SX1276 in the EU regional settings.

This prototype is the one used for the rest of the project implementation and will be referred to as “the prototype” or “the device” in following chapters.

4.2.2.1 Pin mapping of the SX1276 LoRa radio module

To connect the SX1276 radio module to the Arduino, there must be a mapping of pins. The digital pins (DIO) of the SX1276 are used in the LMIC library to get status information from the transceiver. The transceiver pins that the library needs are DIO0, DIO1 and DIO2. The other pins can be left disconnected but should be marked “LMIC_UNUSED_PIN”. These transceiver DIO pins can be connected to any of the Arduinos I/O pins.
[69]

In LoRa mode, the transceiver DIO pins are configured like this:

- DIO0: TxDone and RxDone
- DIO1: RxTimeout

For example, when a LoRa transmission starts, the DIO0 pin is configured as a TxDone output. When the transmission is complete, the DIO0 pin is set high.
[69]

In the sketch for the prototype the pin mapping is set like this:

```
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
```

C / C++ code snippets 1 Pin mapping SX1272

The different pins are defined:

- nss is active low slave select, Arduino pin 10.
- rxtx is unused.
- rst is reset – mapped to Arduino pin 9.

dio are the transceivers digital input/ output pins 0 to 2 – mapped to Arduino pins 2, 6 and 7.

[69]

4.2.2.2 Adding the prototype to a back-end system

The device is an OTAA device. Therefore, to add the device to a LoRaWAN network server, the device EUI and the application key must be set according to the keys generated by the network server. The device EUI must be set in little endian format, and the application key is set in big endian. These are set in arrays in the sketch.

```
static const u1_t PROGMEM DEVEUI[8]={ 0x8e, 0xa7, 0x1f, 0xed, 0x92, 0x67, 0x80, 0x33 };

static const u1_t PROGMEM APPKEY[16] = { 0x2e, 0x1c, 0xc4, 0xab, 0x16, 0x76, 0x16, 0xf9, 0x8a, 0xe8, 0x73, 0x8a, 0x60, 0x02, 0x6f, 0x1f };
```

C / C++ code snippets 2 Device EUI and application key

Connecting the prototype to the back-end system is covered in chapter “4.4.2”.

4.2.2.3 Measuring temperature

The function “readTemp” reads the analog input on the Arduino pin A0 with the Arduino “analogRead” function.

```
const int temperatureSensorPin = A0;

analogValue = analogRead(temperatureSensorPin);
```

C / C++ code snippets 3 Pin mapping and reading analog in

Then this value is converted to a voltage. The reading has to be timed with the reference voltage, and since the reading has a resolution of 1024, the value has to be divided by 1023.

```
voltage = analogValue * 5.0/1023.0; // To turn the analog reading into voltage. Uno has 10bit resolution
```

```
float temperature = (voltage - 1.25)/0.005;
  int16_t tempDataSend = temperature*10; // Times 10 because the
  tempsensor data is 0.1 C ref Cayenne Low Power Payload (LPP)
```

C / C++ code snippets 4 Converting analog value to voltage and temperature

4.2.2.4 Storing the data

The sketch contains an array the data can be stored in. This array is constructed based on the Cayenne Low Power Payload (see chapter “3.3.5 Cayenne Low Power Payload (LPP)”). Where the positioning of the data in the array defines the data channel, data type and the data with N bytes.

```
static uint8_t mydata[4] = {0x01,0x67,0x00,0x00}; //0x01 =
datachannel1 0x67 = data type 67= temp sensor 0x00 0x00 = data, 2
bytes
```

C / C++ code snippets 5 Array for storing data

Here, the first position in the array specifies data channel 1. The second position has the value of 0x67 HEX = 103 decimal, which specifies that the data type is a temperature sensor. The final two bytes are the data for a temperature sensor, as the LPP specifies two places for this data type.

The temperature function stores data in this array by feeding the positions 2 and 3 with float data converted to integer data.

```
float temperature = (voltage - 1.25)/0.005;
int16_t tempDataSend = temperature*10; // Times 10 because the
tempsensor data is 0.1 C ref Cayenne Low Power Payload (LPP)
// Feed the mydata array with data- The temp sensor have 2 bytes
data ref LPP
    mydata[2]= tempDataSend>>8; // Data shift 8 places
    mydata[3]= tempDataSend;
```

C / C++ code snippets 6 Feeding data array

The data is converted to integer, so that it can be shifted 8 places to the right. This shifting is necessary because the temperature data does not take up more space than one byte but the LPP has space for two bytes of data.

4.2.2.5 Transmitting the data

The MCCI LMIC library has several types of events. On the event “EV_TXCOMPLETE” the previous transmission is completed, and the next transmission can be scheduled. The function “do_send” is called on this event. This function first checks if there is a job running, either sending or receiving. If there is no job running, the data can be prepared for transmission. Here the temperature function “readTemp” is called, it calculates the temperature and the data is stored in the “mydata” array.

```
// Function for sending data
void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    }
    else {
//*** Do the work before sending data via LoRa and LoRaWAN
    readTemp();
```

C / C++ code snippets 7 Function for sending data

This array is then prepared for transmission with the “LMIC_setTxData2”.

```
// Prepare upstream data transmission at the next possible time.
  LMIC_setTxData2(1, mydata, sizeof(mydata), 0);
        Serial.println(F("Packet queued"));
    } // end else
    // Next TX is scheduled after TX_COMPLETE event.
}
```

C / C++ code snippets 8 Preparing transmission

4.2.2.6 Limiting transmissions / compliance with regional parameters

To limit the time between transmission there is a parameter to decide how often to transmit.

```
      // How often to send data in seconds
      const unsigned TX_INTERVAL = 60;
```

C / C++ code snippets 9 Transmission interval setting

This parameter can be used to ensure that the device is compliant with the EU LoRaWAN regional parameters and other regional laws and regulation regarding time on air. This is an area for future development and is other regional laws and regulation in regard to time on air. This is a future development and is discussed in chapter "5.7.1”.

The complete source code for this prototype can be viewed in appendix 6.

4.3 Implementation of a LoRaWAN back-end system

To set up the back-end of a LoRaWAN system, an Ubuntu server installation, with the deployment of ChirpStack LoRaWAN server was deployed. ChirpStack has some dependencies witch also needed installation. These were MQTT, PostgreSQL, and Redis. In addition to this Telegraf and InfluxDB was installed. Telegraf is used for conveying measurement data from the ChirpStack application server to InfluxDB. InfluxDB is used for data storage, visualization, and notifications. All details of the installation and configuration of the back-end system is covered in appendix 7.

4.3.1 Sketch of back-end system setup

The setup of the back-end system can be viewed as a complete system with components that are dependent on each other. To get an overview of how the back-end system was implemented it is easier to view an excerpt of the back-end part of “Figure 3 The different parts in the implementation”:

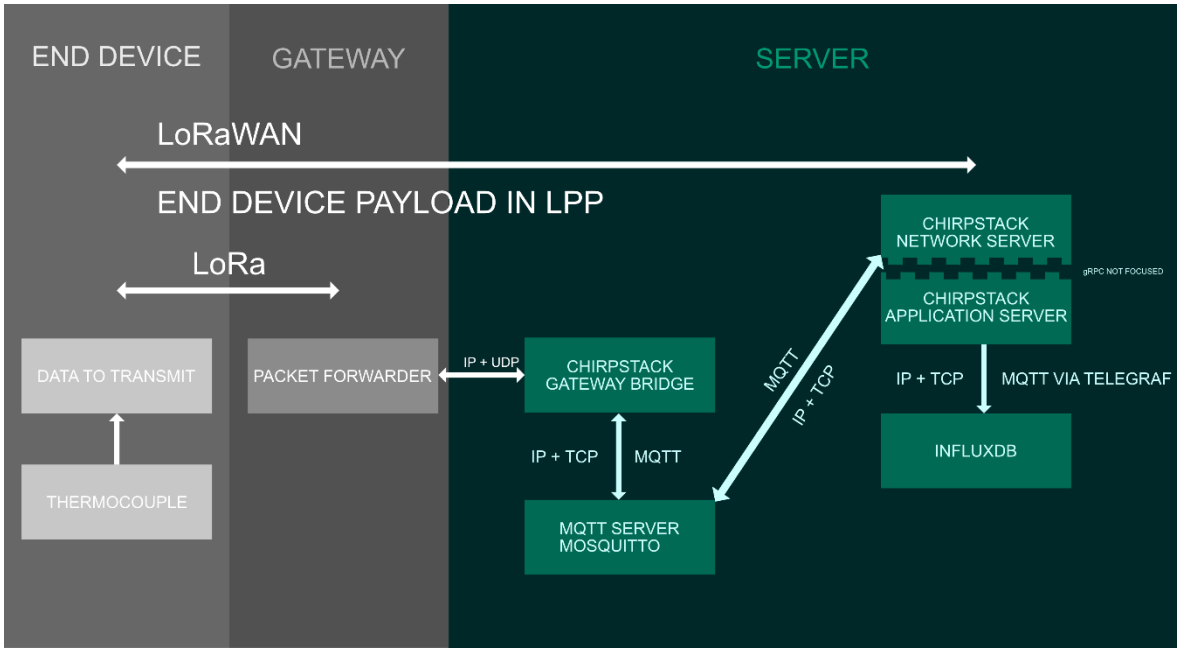

Figure 34 Overview of back-end implementation [5].

The gateway bridge communicates with the ChirpStack network server via MQTT. The network server and the application server are connected as one “entity” and the application server forwards its data to an Influx bucket (similar to database). The data from the application server is forwarded by MQTT via Telegraf.

4.4 Connecting to the back-end system

When the installation of all the components for the back-end was complete, the process of adding a gateway and the prototype to the back-end began. Adding these devices to the back-end system was done via the ChirpStack application servers web-interface. The gateway forwards its packets to the back-end via Semtech UDP packet forwarder.

4.4.1. Connecting the Dragino LPS8 gateway to the ChirpStack LoRaWAN system

Connecting to the gateway requires the creation of a service profile. The service profile was configured with a name and other parameters such as gateway meta data, geolocation, battery level of devices and link margin. It was also possible to set the frequency for requesting end devices statuses. The data-rate also needed to be configured and was set to the range of 0-5 for the EU region [70].

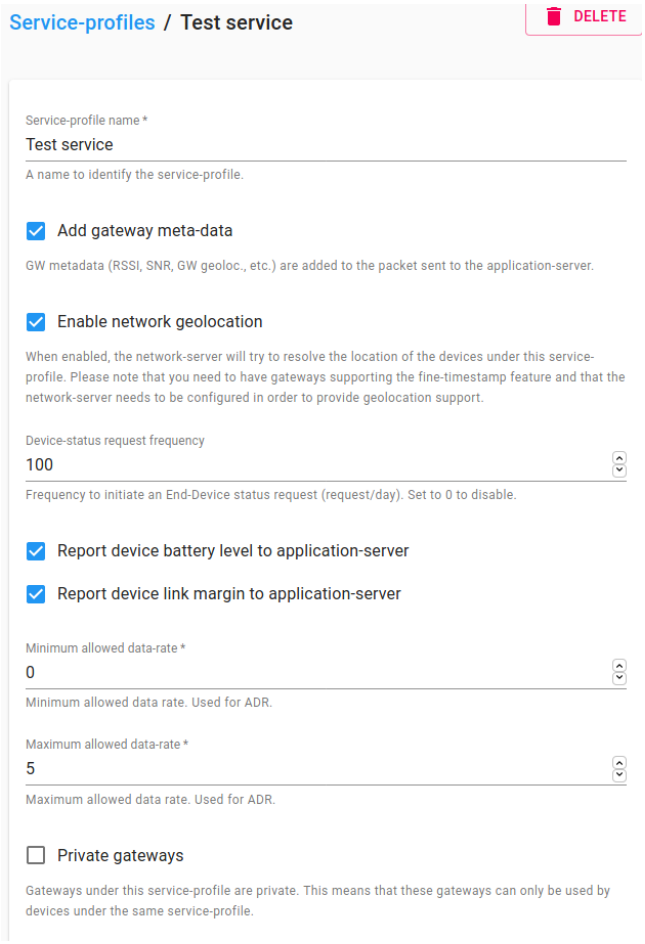

Figure 35 Service profile

When the Dragino LPS8 gateway was added to the system, it is displayed with a status on the application server which provides information about the gateway name, ID, network server and when it was last seen by the system.

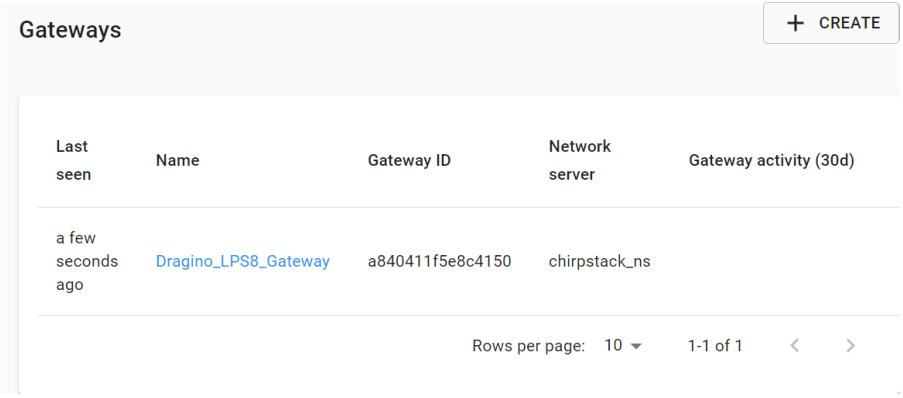

Figure 36 Gateway status in the application servers web interface

It was also possible to view live LoRaWAN frames as the gateway communicated with the ChirpStack system. Here is a LoRaWAN frame that have been sent from the gateway.

Figure 37 Live LoRaWAN frames from the LPS8 gateway

This made it possible to check that there was communication between the gateway and the network and application server.

4.4.2 Connecting the prototype to the back-end

To connect the prototype to the back-end system, it was first necessary to create an application and a device profile for it to be assigned to, in the ChirpStack application server's interface. The application was assigned to the same service profile as the LPS8 gateway. The creation of the application was limited to giving it a name, a description and applying the service profile.

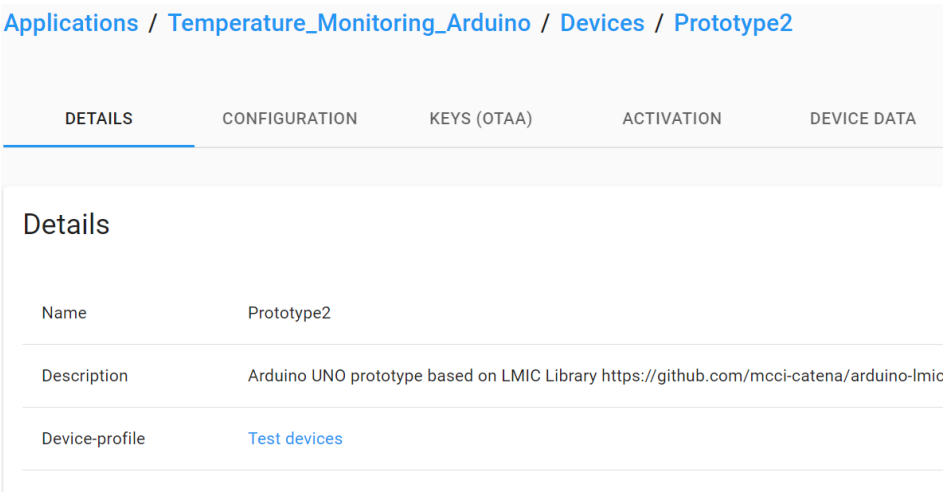

Figure 38 Creating an application for the prototype

Next was the creation of the device profile. The device profiles define a lot of different parameters. In this case the LoRa MAC version that the prototype 1 device supported was 1.0.2. The device class was class A. The ADR algorithm had only a default option. And the interval for the device uploads is set to determine when or if a device is inactive. Further the device profile decides if devices should be activated through OTAA or ABP. In this implementation OTAA was used. EIRP (the transmission power) was not altered.

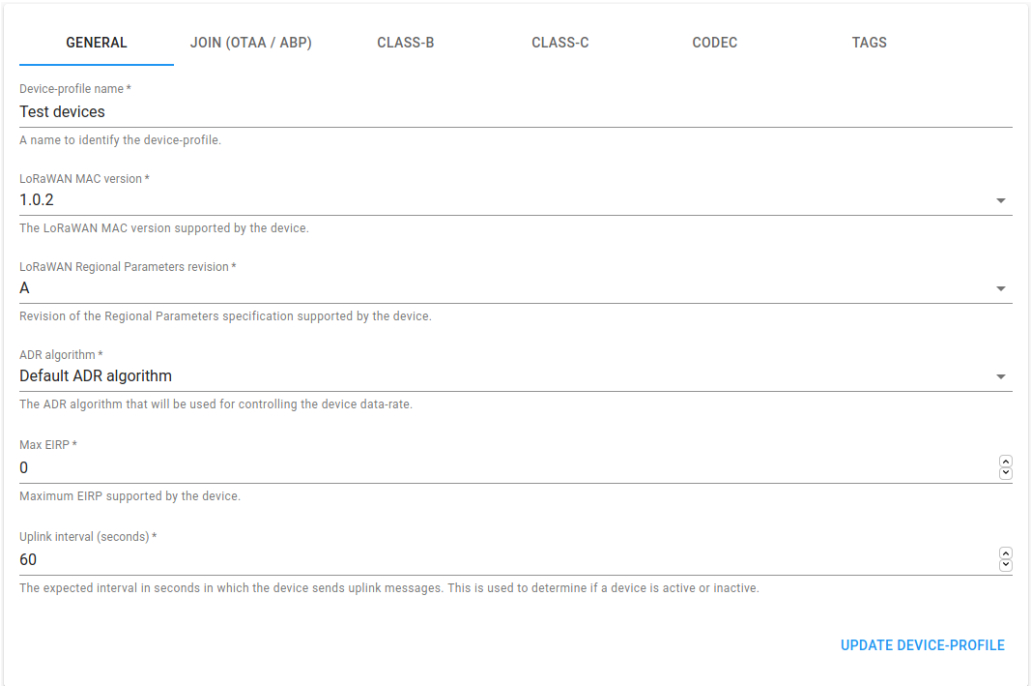

Figure 39 Device profile – defines LoraWAN version, LoRaWAN class, ADR and EIRP

Next was adding the prototype. This was done by assigning a name to the device, a description, the device EUI and assigning it to a service profile. Once the device was added it would be activated by OTAA. It was possible to view the parameters the network server had generated and assigned to the device. These parameters were device address, network session key and application session key. It was also possible to view a count of the device LoRaWAN frame uplink and downlinks.

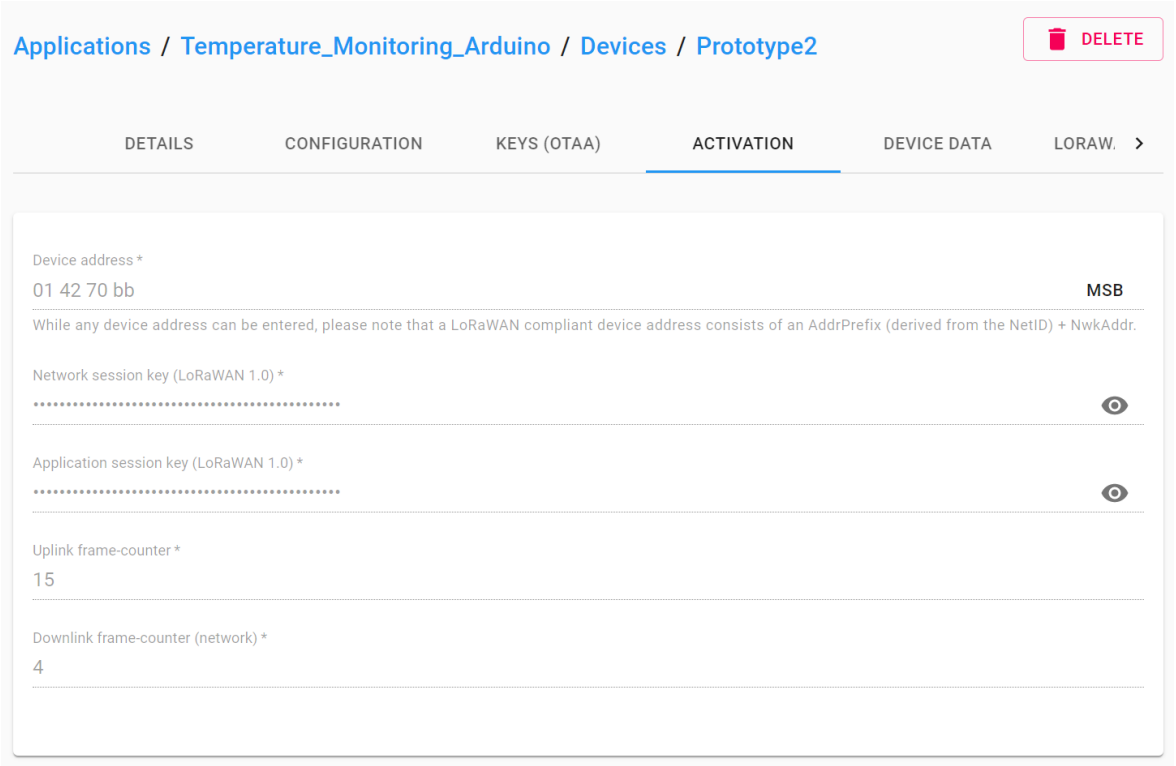

Figure 40 Parameters assigned to the device

When the device was added it had a similar status as the gateway where it was possible to see when the device was last seen.

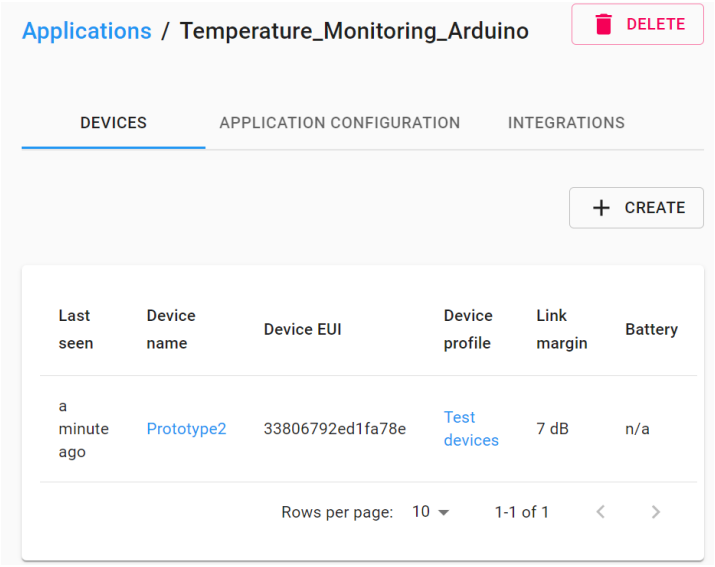

Figure 41 Device status, last seen.

Both live LoRaWAN frames and device data could be viewed under the application in the interface. This was a good way to see if there is any data reaching the application server. In device data, the application payload was recognized as a temperature sensor with its corresponding value. This is because of the LPP and ChirpStacks support for this protocol (see chapters “3.3.5” and “4.2.2.4”)

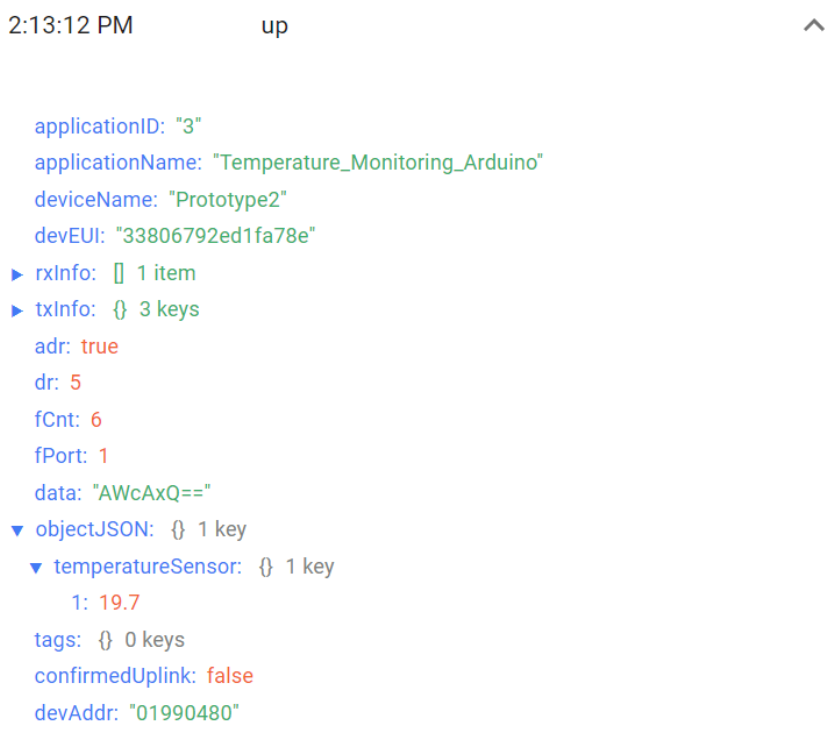

Figure 42 Sensor type and value recognized from payload

When both the gateway and the device were added, their status is presented in an overview under the organization they are connected to.

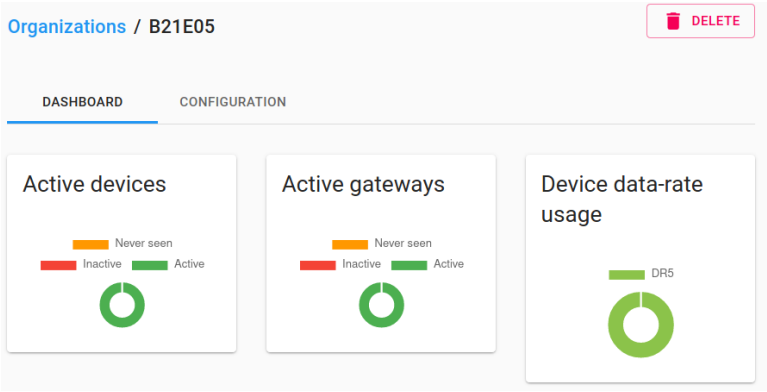

Figure 43 Organizational status overview

This status provides a quick check to see if any devices are considered inactive, never have been seen or active devices.

4.5 Storing measurement data

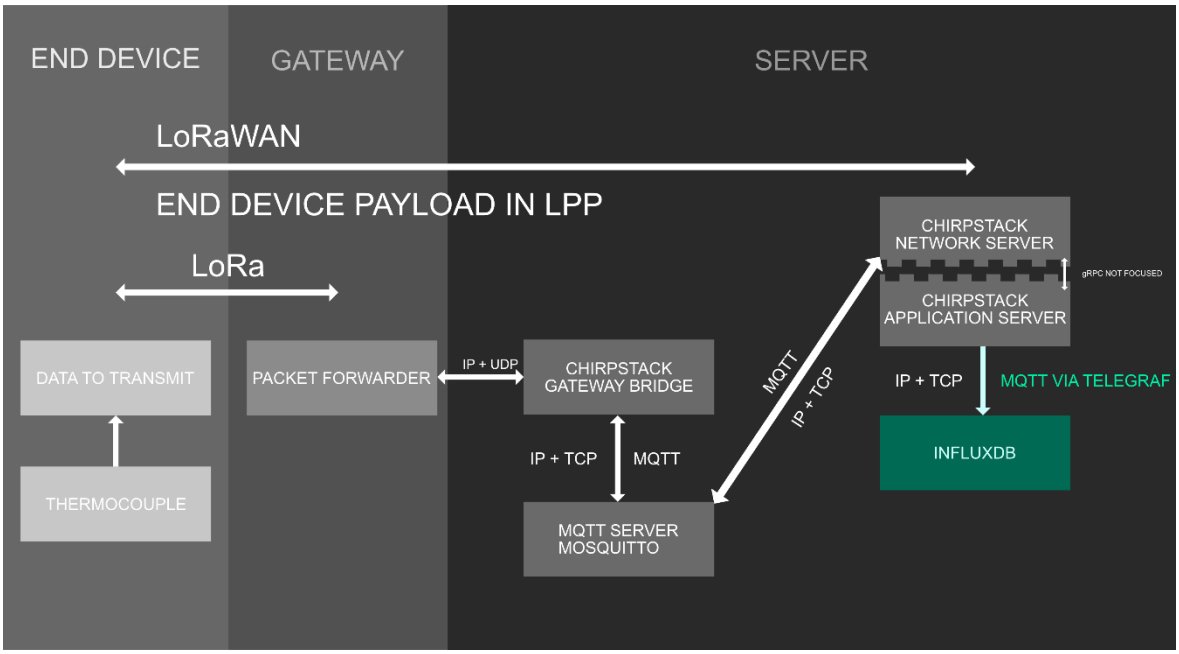

Figure 44 Conveying data and storing it [5]

Data that arrives at the back-end system does not automatically get stored anywhere. To have a data storage where the data can be monitored over time, it is necessary to implement a database system. InfluxDB (see “3.6”) was chosen for this task. Why InfluxDB

was chosen over PostgreSQL is covered in chapter “5.3”. Data is sourced from the ChirpStack application server to InfluxDB. This is done with the help of Telegraf and MQTTs publish and subscribe principle (see ”3.3.2.2.1 The publish and subscribe model.”).

4.5.1 Conveying data with Telegraf

To get measurement data from the ChirpStack application server to an InfluxDB database, Telegraf (see “3.5”) was used as an intermediate. There is an integration option in the ChirpStack application server for connecting directly to an InfluxDB database [71], but this does not yet have implemented support for InfluxDB version 2 [72]. Therefore, Telegraf was chosen to do this task as it is a good option for migration [73]. Telegraf is installed as a service running on the same server as the rest of the back-end system.

The configuration of Telegraf defines how it should convey data by using input and output plugins. This configuration is done in the Telegraf configuration file “telegraf.conf”. Telegraf sourced the measurement data from the application server via an MQTT plugin [74]. A MQTT plugin was used, as it already runs in the system and connects many of the components in the back-end (see “Figure 34 Overview of back-end implementation”). First the MQTT plugin was configured to connect to the MQTT server (Mosquitto – see chapter “3.3.2.5 Mosquitto – an open source MQTT server”) with address and port number to listen to.

```
[[inputs.mqtt_consumer]]
  ## Broker URLs for the MQTT server or cluster.  To connect to multiple
  ## clusters or standalone servers, use a seperate plugin instance.
  ##   example: servers = ["tcp://localhost:1883"]
  ##            servers = ["ssl://localhost:1883"]
  ##            servers = ["ws://localhost:1883"]
  servers = ["tcp://localhost:1883"]
```

Configuration 1 Setting MQTT server address and port

Then the topics to listen to was defined. The application server publishes data from the given application by its ID number, in this case it is number 3, and the rest of the topic is specified by the event up. This means that it listens to the device uploads, which contain the payload with measurement data.

```
  ## Topics that will be subscribed to.
  topics = [
    "application/3/device/+/event/up"
  ]
```

Configuration 2 setting the topic for subscription using a Telegraf input plugin

To output this data into InfluxDB, Telegraf was configured with the “InfluxDB v2” output plugin [75]. This was set to write the data to the specified bucket on the address and port for InfluxDB.

```
[[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  ## urls exp: http://127.0.0.1:8086
  urls = ["http://localhost:8086"]
```

Configuration 3 setting the address and port for writing data to

Then a token for accessing the bucket is defined. This token is required for write and read permissions.

```
## Token for authentication.
  token = "oHkCIQEIi2LP5iPZkx-v6khWMQ9uu2Xbd3A-
sbZwa5iSHpi1WfviKbWbEEahg4tKI3GP-wv6_Fs1296nHF69_g=="
```

Configuration 4 Setting the token for bucket read / write permissions

Then the organization and bucket to write to is defined. (The bucket is named prototype 1, but is used for prototype 2. Renaming it could have undesirable consequences for data queries and more, therefore its name remains unaltered).

```
  ## Organization is the name of the organization you wish to write to; must exist.
  organization = "B21E05"

  ## Destination bucket to write into.
  bucket = "prototype1"
```

Configuration 5 setting the organization and bucket to write data to

With these plugins configured Telegraf could convey its data to InfluxDB.

4.5.2 Storing data in InfluxDB

Storing the measurement data is done in InfluxDB. The data from Telegraf is written to a InfluxDB bucket. This bucket was created in InfluxDBs user interface.

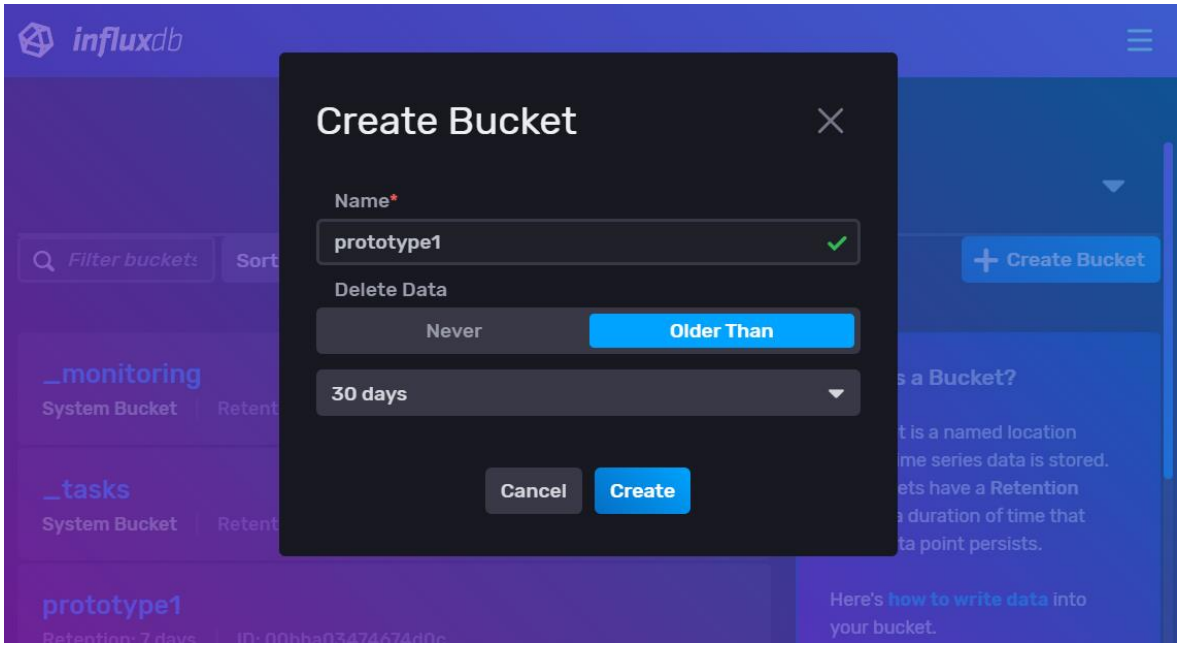

Figure 45 Creating bucket and setting retention period

Here the bucket is named, and the data retention period was set. The data to be fed into this bucket was set earlier in the Telegraf plugin. And when the bucket was created the data explorer was used for checking if any data was reaching the bucket. The option "view raw data" was a way to do this.

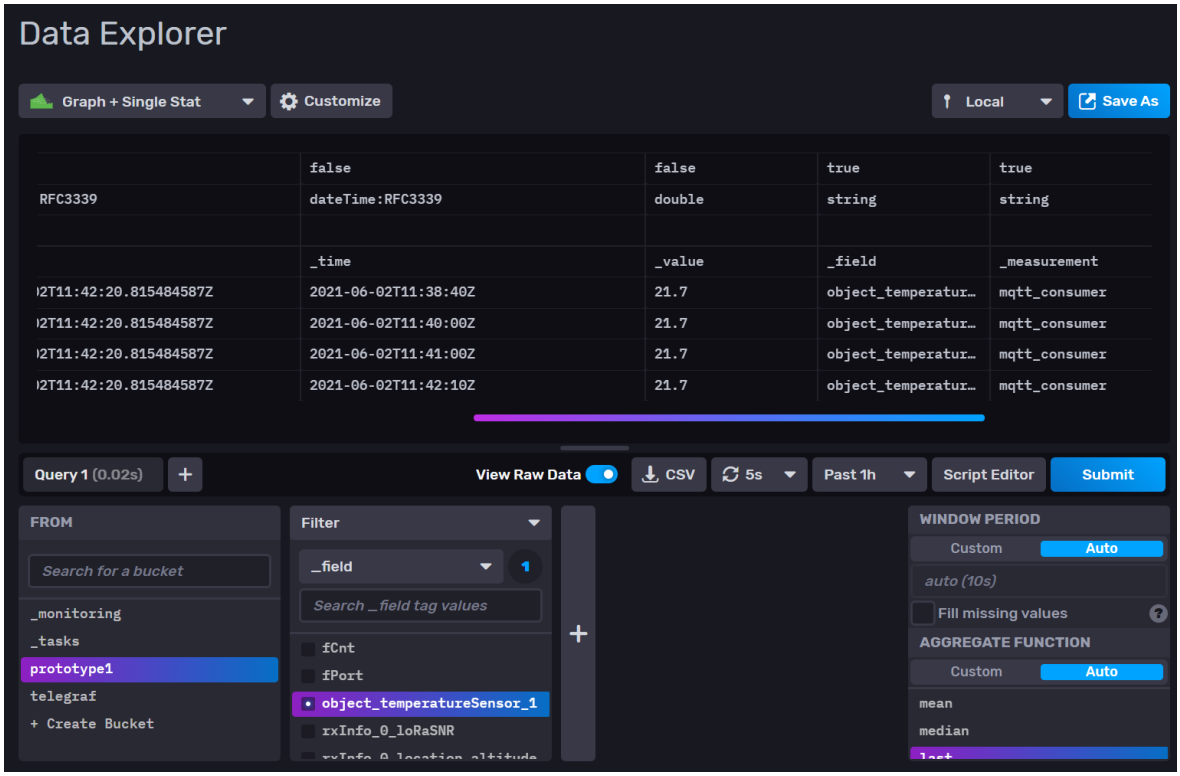

Figure 46 Inspection of raw data

The inspection is set for data for the past hour, to be refreshed every five seconds, and the data is set for the last value. The inspection of data for the bucket prototype1 has the option for filtering by the field “object_temperatureSensor_1”. This means that the measurement field is reaching the bucket, since the measurements are using the field names from the object JSON key [71]. One can also see that the values for the measurements are reaching the bucket under the _value column. When it is verified that data is reaching the bucket the creation of queries was done. The queries were set to the last value for measurements and the measurement data. These queries were saved as data cells and could then be used for data visualisation in the InfluxDB dashboard manager.

4.6 Test of data visualization

Figure 47 Test measurement in freezer. 3 different cells for visualization

The measurement data could be visualised in many ways. This was done via the dashboard in InfluxDB where cells with different ways of displaying data is created. Here, the data is shown in cells for a graph, a gauge, and a single stat. During a measurement test in a freezer, the graph shows that the prototype had been in room temperature before the temperature plummets when the freezer measurement began. The graph shows data for the past six hours, but this time period could be set to any amount of time.

4.7 Setting up back-end notifications

In the monitoring of the temperature, it would be desirable to have a form of notification if the temperature rose beyond a critical value. To solve this InfluxDB can be used with the option for alerts. To set up notifications in InfluxDB, there are three steps to follow. The first step was setting up a check. The check is set as a critical value check and defined to be triggered when the temperature is above -20 C (this due to testing in a normal freezer at this stage).

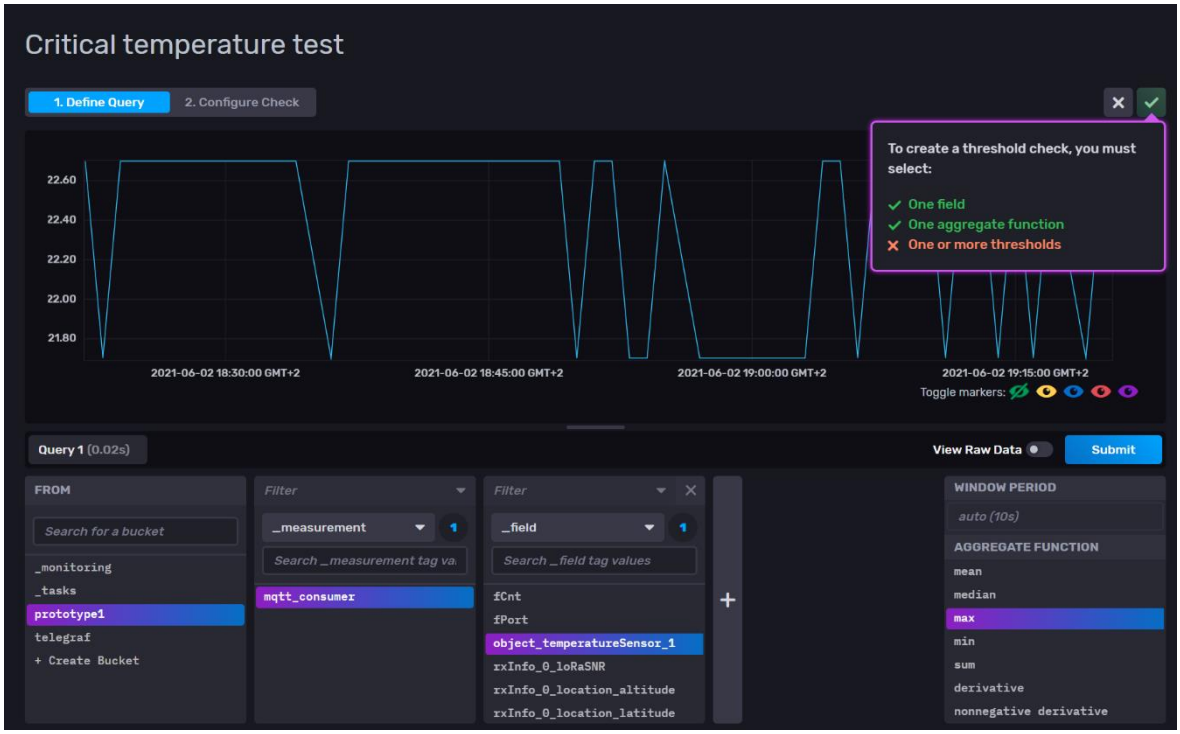

Figure 48 setting the data to check

This was done by selecting which data to check and then set a threshold. The data was set by selecting the bucket and the right measurement. Just like when setting up queries for visualisation of data.

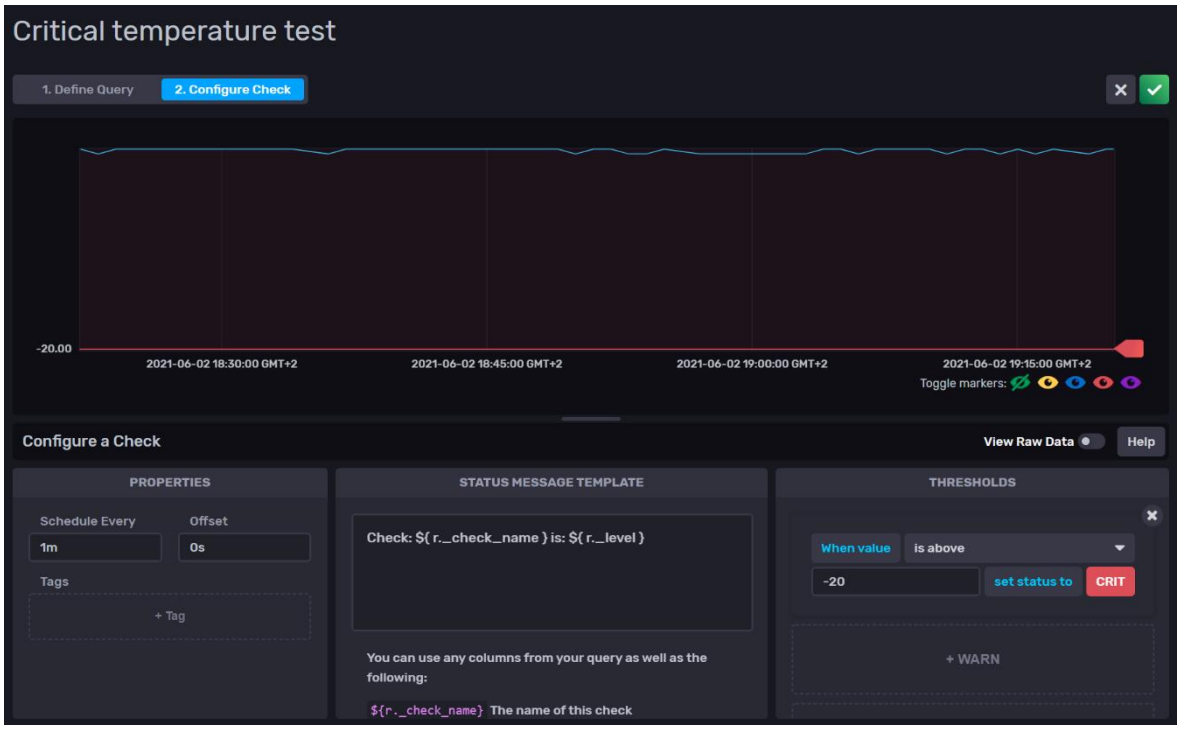

Figure 49 Setting the threshold to monitor

When the check was set, the next step was to create a notification endpoint in InfluxDB. Pagerduty was used for this notification endpoint test. This was done using Pagerdutys events API [76].

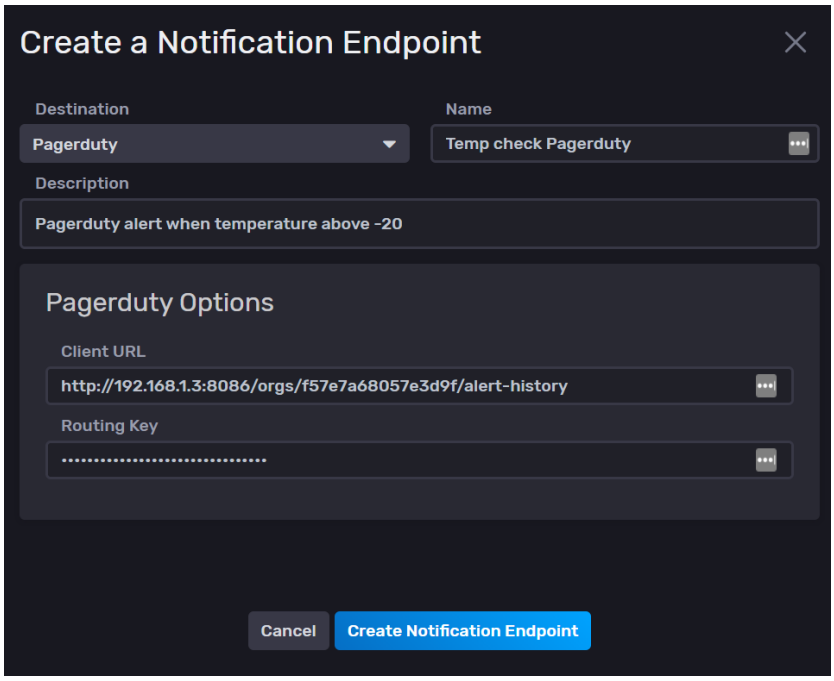

Figure 50 Notification endpoint

The endpoint in InfluxDB was integrated with PagerDuty using PagerDutys guide for InfluxDB integration [77]. With the notification endpoint set, the final step was configuring the notification rules. In the ruleset the notifications are configured with the time interval for when a notification should be sent. Then the conditions for the notification are set. Here it was set for when a status is equal to critical. The status critical was defined in the check to be when the temperature was above -20. In this test of notification, a critical value condition was used. There are other levels of notifications as well. These are info, warn, ok and any. So, there are many possibilities for setting up notifications about status, and early warnings.

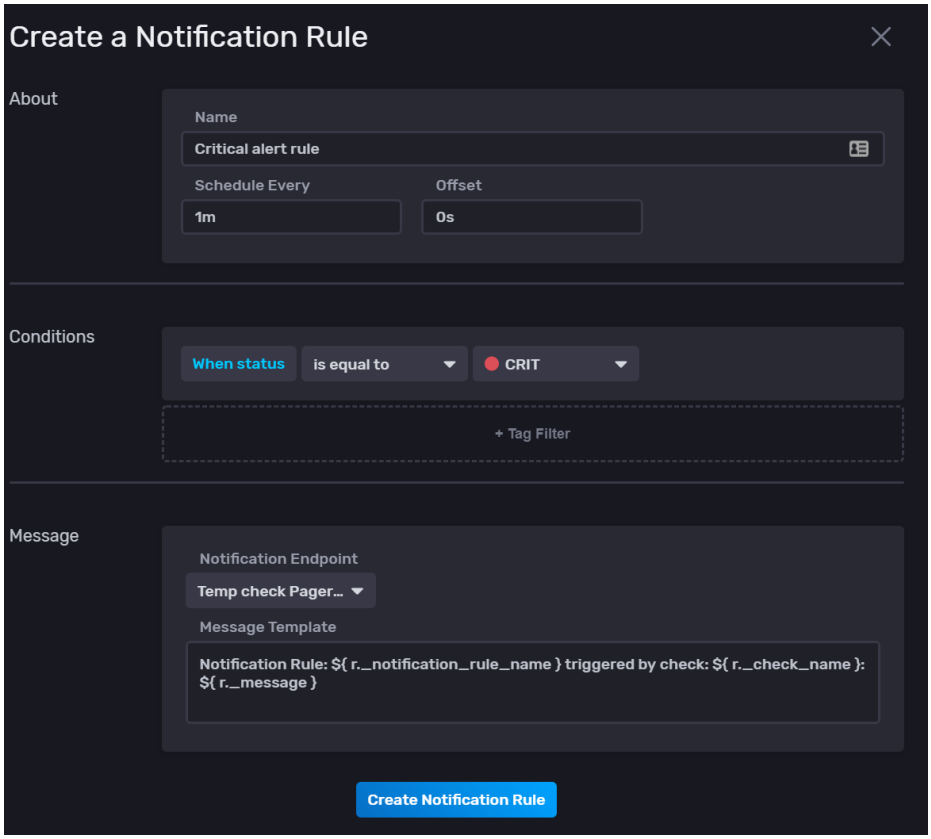

Figure 51 Configuration of notification rule

With these steps in InfluxDB complete and with a trial version of PagerDuty the implementation for notification was tested. PagerDuty was set to notify by email and SMS. In addition, there is an overview of incidents on PagerDutys site.

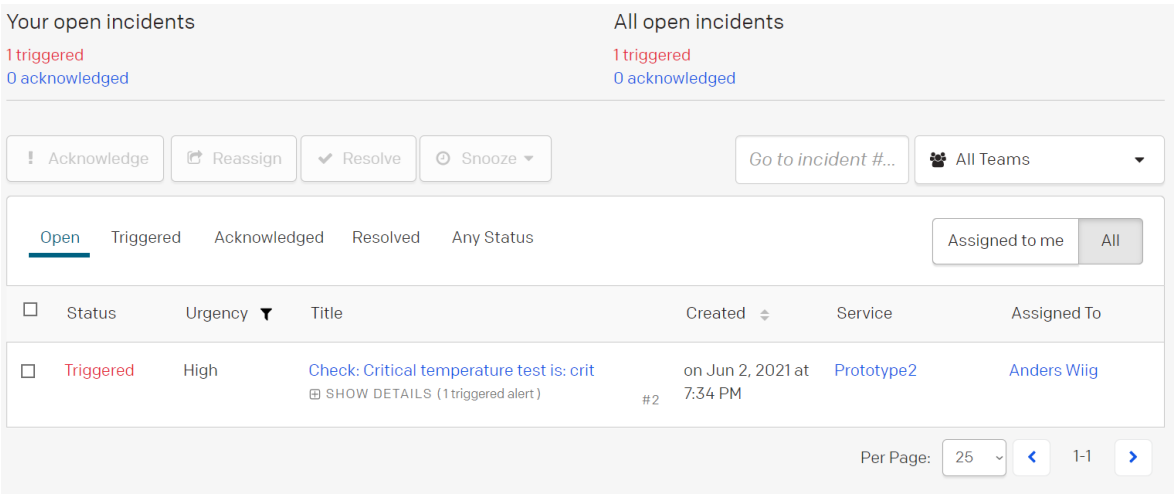

Figure 52 Overview of incidents on PagerDutys site

The notification appeared on the site, and an email and SMS were received when the check reach critical.

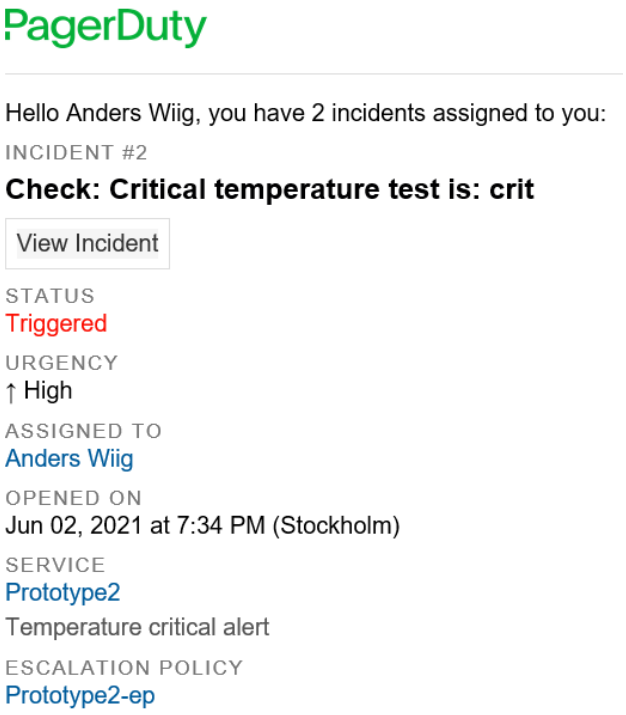

Figure 53 Email notification from PagerDuty

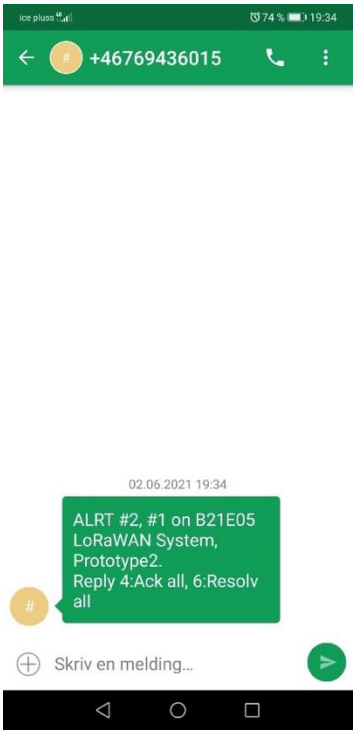

Figure 54 SMS notification from PagerDuty on critical value for the prototype

4.8 Testing the thermocouple at ultra-low temperatures

The thermocouples ability to measure ultra-low temperatures was tested by measuring a freezer that held -75°C at the bioengineer laboratory at HiØ. This test was done purely on the basis of looking how well the thermocouple did and was not performed with the final prototype, so the temperature is only measured and shown on Arduino IDE's serial monitor.

Figure 55 The thermocouple inside the ULT freezer

As seen below, the thermocouple needs some time to adjust to the new temperature. The voltage shown in the measurements is purely a variable used to calculate the temperature. When the thermocouple had cooled down, it stabilized and the temperature readings were accurate, with only a small deviation of 1-2°C.

Temperatures imediately after putting the thermocouple in the freezer	Measurements after the the temperature had stabilized
13:00:26.946 -> Temperature = 22.46 C 13:00:26.946 -> Voltage = 1.36 V 13:00:30.549 -> Temperature = -10.74 C 13:00:30.549 -> Voltage = 1.20 V 13:00:34.145 -> Temperature = -3.91 C 13:00:34.145 -> Voltage = 1.23 V 13:00:37.721 -> Temperature = -16.60 C 13:00:37.721 -> Voltage = 1.17 V 13:00:37.806 -> Temperature = -17.58 C 13:00:37.806 -> Voltage = 1.16 V 13:00:41.411 -> Temperature = -29.30 C 13:00:41.411 -> Voltage = 1.10 V 13:00:45.038 -> Temperature = -34.18 C 13:00:45.038 -> Voltage = 1.08 V 13:00:48.615 -> Temperature = -40.04 C 13:00:48.615 -> Voltage = 1.05 V 13:00:52.237 -> Temperature = -44.92 C 13:00:52.237 -> Voltage = 1.03 V 13:00:55.842 -> Temperature = -48.83 C 13:00:55.842 -> Voltage = 1.01 V	13:04:15.850 -> Temperature = -76.61 C 13:04:15.850 -> Voltage = 0.87 V 13:04:19.453 -> Temperature = -74.69 C 13:04:19.453 -> Voltage = 0.88 V 13:04:23.048 -> Temperature = -74.69 C 13:04:23.048 -> Voltage = 0.88 V 13:04:26.649 -> Temperature = -74.69 C 13:04:26.649 -> Voltage = 0.88 V 13:04:30.239 -> Temperature = -75.65 C 13:04:30.239 -> Voltage = 0.87 V 13:04:33.832 -> Temperature = -75.65 C 13:04:33.832 -> Voltage = 0.87 V 13:04:37.452 -> Temperature = -74.69 C 13:04:37.452 -> Voltage = 0.88 V 13:04:41.069 -> Temperature = -75.65 C 13:04:41.069 -> Voltage = 0.87 V 13:04:44.654 -> Temperature = -74.69 C 13:04:44.654 -> Voltage = 0.88 V 13:04:48.241 -> Temperature = -76.61 C 13:04:48.241 -> Voltage = 0.87 V

Table 16. Ultra low temperature measuring test

In addition to testing the measurement capability at ultra-low temperatures, three other tests were also performed. These tests were done on room temperature, a refrigerator, and a standard freezer. To see these result look at appendix 8.

5 Discussion

5.1 The development of a second prototype

During the project, the team developed two prototypes based on LoRa and LoRaWAN. There are two main reasons for why the project team chose to develop a second prototype. The first reason is that the first prototype, that was based on the Dragino IoT development kit, was too complete. Meaning that there was not a lot to do for the project team other than to follow the manual that came with the kit. There was also a

sketch for the Arduino which only needed some modification for the first prototype to be up and running. The project team wanted a deeper understanding and a prototype that was developed by the team itself.

The second reason is that there is a limitation in using the gateway that came with the kit. This gateway operates on only one frequency. This means that the sketch for the first prototype needed to use a specially adapted version of the LMIC library for Arduino to function towards this gateway. This library was not maintained by Dragino and they expected it to be full of bugs. As stated by Dragino support here:

> *"Our LMIC library end node used for Arduino is not maintain for a long time. So this should not be used for production enviornment and we expect there will be lots of bugs.*
>
> *For your case, if the Arduino keeps sending packets(even the packets doesn't arrive ChirpStack, for example power off the gateway). If you power on the gateway again, the ChirpStack should be able to get the packets unless there is some requirement on ChirpStack such as keep alive request. I am not familiar with ChirpStack so i can't help you further on this topic." - Edwin Chen, Dragino support*

A library that was not maintained and could not be used for a production environment was not suitable. Therefore, there was a need for hardware that could use standard LoRaWAN libraries. Furthermore, this also meant that no ordinary LoRaWAN node would work towards the LG01-N gateway. This was also confirmed by Dragino support:

> *"The single channel gateway has limitation to use with standard LoRaWAN sensor. In single channel gateway, the gateway can only set to work at one frequency and data rate. A standard LoRaWAN Sensor can works in different frequency (3~72 depends on the frequency bands) and different data rates. So Single channel gateway will also lost packets because the frequency and data rate mismatch." Edwin Chen – Dragino support.*

The full email correspondence is in appenix 9. The manual for the Dragino IoT kit v2 also states issues with the limitations of the LG01-N gateway as seen from the excerpt from the manual at appendix 10. This meant that the LG01-N gateway had to be replaced with a standard LoRaWAN gateway without these limitations.

It could be argued that the project team should have known about this issue beforehand and therefore purchased a standard LoRaWAN gateway and hardware for nodes that would be standard LoRaWAN nodes. But having no experience of the systems and technology from before, the project team had no prerequisite knowledge of this being an issue and could therefore not have come to this realization before the project began. The projects advisors were also not aware of this being an issue when the kit was suggested for the project at the beginning.

5.2 Temperature measuring device

The DHT11 temperature sensor was used with prototype 1 as it was based on the Dragino LoRa development kit. This was discovered to not fulfil the requirements for the use case as it could only measure down to 0°C and had to be replaced for prototype 2.

To choose the new temperature sensor for prototype 2, a few different options was considered (see “3.7”). By evaluating the different methods and looking at the availability, the thermocouple was concluded by the group as the best option. It is a low cost option, can measure down to the required temperatures, is very robust, and was possible to implement within the time limit.

An issue with the 24 AWG K-type thermocouple is that it cannot measure lower temperatures than -73°C (referring to Appendix 4). However in reality it did measure down to -76.61°C during testing (see 4.8 Testing the thermocouple at ultra-low temperatures). That is still not as low as the lowest point that Pfizer’s vaccine requires. The team was aware of this when ordering the thermocouple. The decision to order this thermocouple was due to time limitations, and that the retailers did not have a thermocouple that could reach the required temperatures in stock when the order was made.

5.3 The choice of database system for implementation

To store measurement data the system implementation needed a database system. The back-end system does already have the PostgreSQL database system installed, for which it relies on. Then why use InfluxDB instead of using the ChirpStack integration for PostreSQL [78] to store data from the application server? As the developer of ChirpStack puts it:

> *"It depends per use-case. E.g. it is one less component to worry about, as PostgreSQL is already required by LoRa App Server so you don't have to install InfluxDB. Also it could provide more flexibility as you can write more advanced SQL queries compared to InfluxDB. It could also be used as an intermediate storage which is used by applications to fetch (and then remove) data from. On the other hand, InfluxDB is written specific for time-series data and depending your use-case, you might find features in InfluxDB that are not present in PostgreSQL." – Orne Brocaar [79]*

The fact that InfluxDB is written as a timeseries database makes it more attractive [80]. And features that is interesting from InfluxDB is data visualization [56] many possible tools and integrations [81] and the setting of alerts with option for integration with a notification service [58].

If the inbuilt options for visualisations are insufficient, there are other services that can display data by using InfluxDB as data source. InfluxDB have integrations for Grafana and Google Data studio [82]. If the options for incident management and alerting are insufficient there are integrations for this [83] , as well as several other integrations via Telegraf [84]. These possibilities for integrations substantiate the capabilities of InfluxDB.

5.4 Existing solution and what our solution offers in comparison

This chapter describes the similar existing solution to the issues solved in this project. The existing solution is compared to the product designed. Pfizer and Orioin M2M has solutions similar to our system.

5.4.1 Pfizer's solution

Pfizer is supported by Softbox and Controlant in the global distribution of the Covid-19 vaccines. Softbox has developed a freezer for Pfizer, that uses a specialized reusable ultra-low temperature (ULT) shipper that also works as a point of use storage unit. Controlant has produced a real-time, reusable data logger with a temperature probe which is enabling the possibility to help manage temperature proactively and having the possibility to react. Controlant's solution also gives the opportunity to alert Pfizer when the Softbox container has reached its destination and when it is getting shipped back aswell.
[85] [86]

5.4.2 Orion M2M solution

Orion M2M has developed a solution that is using LoRa. The solution is making it possible to remote monitor temperature and humidity. Mainly this solution allows temperature monitoring and geolocation of refrigerators, monitoring of pharmacy refrigeration chambers and monitoring of warehouse and commercial refrigeration equipment. They are monitoring the refrigerators through their own radio modem called Orion Meter, which is then connected to LoRaWAN and OrionM2M network. The systems support 24/7 temperature and humidity control. It archives the temperature change readings hourly for 62 days. And there is SMS and email notifications [87].

5.4.3 The projects solution in comparison

The intention of this project was not to make an improved version of an already existing solution, but rather to develop and implement a functional prototype based on the set use case the project team found interesting. The team was not aware of already existing solutions, as some of these were recently released on the market, even during this projects time frame. Therefore the use case was not set based on existing solutions. Our product does not have a direct comparison to the existing solutions. The project team thinks that the solution developed is closing in on the same functionality and level of

service as some of the existing solutions. If the project were to continue towards a commercial product, a more thorough market research would be necessary.

5.5 Alternatives to our solution

This chapter will discuss different alternatives to the parts in this project. Alternative solutions for end device and alternatives for the back-end system are covered.

5.5.1 Using mobile networks exclusively

For an alternative end device, one could use devices that communicates by mobile networks exclusively. But then, each device would have to talk to the mobile network directly. This would mean that one would have to have subscriptions for every device. This could be costly, and if one is to ship large amounts of cargo, it would quickly add up in number of devices. In addition, there is the issue of price variations according to the which nation the cargo is in at the time of transmission. But since the amount of data to be transmitted is small, this cost might not be too great, dependent of the cost for operating a given number of devices. Alternatively, one could instead have one gateway to talk to the mobile network if the transport is outside the coverage of a LoRaWAN network. This way there is only need for one unit to subscribe to the mobile network, thus reducing cost. This makes LoRaWAN a good supplement to mobile networks for collecting the data. The gateway may use the mobile network if there is a need for such a connection. This also will reduce the number of devices on the mobile network. Dependent on how many messages that requires transmission one gateway can support a varying, but large number of devices:

> A single eight-channel gateway can support a few hundred thousand messages over the course of a 24-hour period. If each end device sends 10 messages a day, such a gateway can support about 10,000 devices. If the network includes 10 such gateways, the network can support roughly 100,000 devices and one million messages. If more capacity is required, all that is needed is to add additional gateways to the network. [9]

One LoRaWAN gateway can therefore reduce the number of devices required to connect to mobile networks by a significant amount.

5.5.2 Alternative back-end solution

The back-end system in this project uses ChirpStack. There are alternative existing solutions which can be used. The Things Network is a network one can connect to for using LoRaWAN. But, if one is to use LoRaWAN in a professional setting, The Things Network (TTN) cannot cover this. This is because TTN have limitations and a fair use policy [88]. For example, the fair use policy states that one is limited to:

- An average of 30 seconds uplink time on air, per 24 hours, per device.
- At most 10 downlink messages per 24 hours, including the ACKs for confirmed uplinks. [89]

This is fine for testing purposes and getting started with LoRaWAN, but would prove not ideal for professional use.

There is an alternative server system to ChirpStack that one could install on a server or host in the cloud. This is called The Things Stack (TTS), offered by The Things Industries [90]. But, unlike ChirpStack there is a prerequisite license required if one is to run more than 30 devices, have more than 3 gateways and more than 3 users. So, the free tier of The Things Stack might not cover the need for a professional environment. The Things Industries does offer cloud services for professional use, but at the base price at 190 Euros per month. They also offer an enterprise solution which starts at 500 Euro per month. If one is to choose between options for support, it becomes even more expensive. The cost also increases per 1000 devices. This makes the use of ChirpStack for the back-end solution more desirable, as it is free at open source. [91]

5.6 LoRa and LoRaWAN Network coverage

If there is no need for an existing network with a vast coverage of LoRaWAN gateways, one can simply install gateways where the coverage is needed and run a private network. This might be cheaper than purchasing a service that has LoRaWAN coverage in a large

geographical area, if this is coverage is unnecessary. For example, if one only needs LoRaWAN coverage in a warehouse and in the trucks that transport shipments, this is solved easily by having a gateway in the warehouse and one gateway in each truck. These gateways will provide LoRaWAN coverage where it is needed – and devices can be moved wirelessly from the warehouse to the truck and change gateway without any need for gateway handover as covered in chapter "3.3.1.2.2 Gateway". The gateway in the truck can then have access to internet via mobile network as discussed in chapter "5.5.1". Having full control of the network helps with keeping control over aspects such as security, and data integrity by not sending data to unknown servers.

5.7 Further development of the project in the future?

This chapter will cover some possible steps for further development. Developments seen as a natural progression are:

- Compliance with regional parameters
- Data logging in the case of communication and critical levels
- Securing MQTT communications
- End device memory usage issues
- End device current consumption issues
- Further development for system on chip
- Expanded temperature range measuring

5.7.1 Compliance with regional parameters

To ensure usage in Norway and EU, there is regulations that needs to be followed. Specifically in Norway the product would have to meet the regulations of "fribruksforskriften". This has not yet been implemented into the prototype but would be one of the main goals in making the system into a commercial product.
"Fribruksforskriften" in Norway is allowing data-transmissions in data-networks in different frequencies. [17] In EU, LoRa regional parameters must be in compliance with LoRaWAN reginal parameters [70].

5.7.2 Data logging in case of loss of communication and critical levels

If the end device loses communication with the back-end system and the temperature reaches warning or critical levels, a functionality for datalogging on the end device should be implemented. With further development of the product, a solution to this issue would be necessary. Several options were discussed during the design process. A possible solution could be implementing a check for loss of communication with gateway and flags that can be set at certain temperature levels in the end device firmware.

5.7.3 Securing communications - Encryption of MQTT communication

In the back-end system MQTT is a vital part, and therefore it should secure. As the system is now, the MQTT communication is not encrypted. In further development of the system such encryption should be implemented to secure the communication and data integrity. The integrity of the data may be compromised if the system is fed invalid data from unauthorized sources.

5.7.4 Reviewing use of library for prototype

The library used in the prototype implementation has certain limitations. If it was desirable to create a class B or C device, or a LoRaWAN 1.1 device - we would have to find another library, as the one used only has support for class A and LoRaWAN version 1.0.2 and 1.0.3 devices. Class B support is present in the library, in an untested capacity. The library does also uses a lot of memory. [68]

> This library can be quite heavy on small systems, especially if the fairly small ATmega 328p (such as in the Arduino Uno) is used. In the default configuration, the available 32K flash space is nearly filled up (this includes some debug output overhead, though). [68]

The library, with the added functionality for temperature measurements takes up 75% of the available memory, and a warning for instability is given in the compiler.

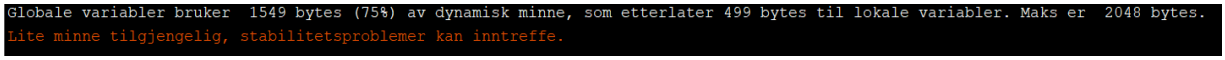

Figure 56 Memory warning - Arduino IDE

This may be an argument for using a different library, or also changing the hardware for end device.

5.7.5 Replacing or supplementing hardware for end device

For the first prototype, the LoRa MAC Handler library provided by Dragino and the Arduino MCU does not provide flexibility with configuring low power modes. It is therefore not an ideal MCU to create a product with acceptable battery life.

A better choice of end device for this project might be to base it on a STM32WL55 board. The two main arguments being that the STM32 microcontroller provides more storage and lower power consumption compared to the Arduino Uno CPU. The Arduino Uno provides 32k bytes of flash, while the STM32 microcontroller provides 256k. The increase of storage would be a necessity with the implementation of GPS for our system. The STM32 microcontroller offers more flexibility with low power configuration compared to the Arduino Uno. The STM32 MCU has the capability of running in stop mode most of the time with a timer and RTC active for interrupting purposes. Stop mode on the STM32 has a current consumption of 1,07µA. The Arduino Uno low power modes do not meet the requirements for our project design and needs to be run in active mode constantly. Active mode has a current consumption of 1,5mA which is 1400x more compared to the STM32 microcontroller.

Because of the massive benefits provided, an attempt in transferring the design to a STM32WL55 board was made, but with little success. More detail about this process is given in the appendix 11.

5.7.6 System on chip – A prototype / product on a printed circuit board

When the development of an end device prototype is considered finalized, it could be integrated on a printed circuit board (PCB). Firstly, this must be designed in a software for PCB, then a small amount could be ordered for testing purposes. This can be ordered

from for example JLCPCB [92] or any other such manufacturer. If these prototypes are approved, the next stage would then be the production of complete products. This production could be done by the same manufacturer of the integrated prototypes.

When the development of an end device prototype is considered finalized, it could be integrated on a printed circuit board (PCB). Firstly, this must be designed in a software for PCB, then a small amount could be ordered for testing purposes. This can be ordered from for example JLCPCB [92] or any other such manufacturer. If these prototypes are approved, the next stage would then be the production of complete products. This production could be done by the same manufacturer of the integrated prototypes.

5.7.7 Expanded temperature range measuring

The vaccines can be stored with other temperature requirements in limited time periods [2]. The prototype can easily be modified by changing the firmware to meet these requirements. The thermocouple covers a wide enough range of temperatures (see appendix 4).

5.8 Alternative use-cases

The components of this project can be used in other use cases or other forms for deployment. Both the prototype and the back-end system can be used as they are or be configured or altered to fit other cases.

5.8.1 Prototype

The prototype can be used in other use cases, as it is. It measures extreme temperatures and transmits its data via LoRa and LoRaWAN. Such device has loads of alternative use-cases where the monitoring of extreme temperatures is needed. With simple configurations the system could be used to monitor other products or systems like other vaccines with low temperature requirements. The 24 AWG K-type thermocouple can also easily be replaced with another K-type for an even larger range of temperatures (see chapter ”3.7”) as the AS8495 K-type thermocouple amplifier supports any K-type.

With the established LoRa and LoRaWAN functionality of the prototype, it can be altered with both hardware and firmware to new applications using this form of communication. This way the prototype can be used as a platform for developing other types of end devices.

5.8.2 The back-end system

There was no need for special adaptation of the back-end system for this project's use case. This means that the back-end system can be used for any other use case utilizing LoRa and LoRaWAN. The LoRaWAN part with ChirpStack does not need any modification, adding new types of end devices can easily be done. The data handling with InfluxDB can be configured to suit new types of data or measurements with simple steps.

Parts of the back-end system can also be extracted to be used with other forms of communication other than LoRa and LoRaWAN. This can be over Ethernet, Wi-Fi, mobile networks, Bluetooth, or any other form of communication. For example, the parts Mosquitto, Telegraf and InfluxDB can be used in any application where MQTT is used for data/ payload communication. Telegraf and InfluxDB can still be used for conveying data, data storage, visualization, and notification. Further, Mosquitto and MQTT can be removed and Telegraf and InfluxDB can be used for handling data with the use of any other type of protocol.

Also, the system can support LoRa, LoRaWAN, MQTT and any other forms of communication and protocols at the same time. By assigning different data-buckets in InfluxDB to each application, the system can gather data from a variety of sources simultaneously. Therefore, it has the capability to be a universal back-end systems for several different deployments of technologies. This shows that the back-end system and its parts offer a great deal of flexibility.

5.9 Covid's impact on the project

The development of a second prototype presented an issue with the access to hardware required for development. There was a need for a new gateway, one that could support a

standard LoRaWAN node. The gateway LG01-N that came with the Dragino IoT kit used for the first prototype (see chapter “4.2.1”) could not support standard LoRaWAN nodes as covered in chapter “5.1 The development of a second prototype”. Østfold university college provided a standard LoRaWAN gateway, that the project team needed to share. The situation due to Covid, meant that only one of the team members could use the gateway at a time, to develop a second prototype. This was far from ideal because it meant that there were three other members that were idle in the development aspect, because there is a need for a gateway to test the prototype against. Having only one gateway would have been too crippling. This was somewhat resolved by one team member purchasing a gateway for himself. If there had been no Covid lockdown or restrictions, the project team could have all been working on developing prototypes and testing it against the same gateway, at the same time. The lost access to the campus, laboratory and all its resources and the possibility for a closer collaboration, have heavily impacted the project and its team. This situation, which has made things difficult have resulted in slower progress than normal circumstances would have.

Further, the work at home situation led to the implementation of the back-end system being done in a way that was not ideal. This system implementation was installed on a disused desktop on a local network. This was due to the “work at home” situation caused by Covid-19. This means that there was no way for the entire team to connect to the server due to it running on a local IP-address.

6 Conclusion

The project team have through theoretical study, prototype development, and practical implementation of a back-end system successfully implemented a working LoRaWAN end device and a complete LoRaWAN back-end system. Through this study and implementation, the team have gained an understanding of how a LoRaWAN system works.

To solve the task of measuring ultra-low temperatures the team implemented relevant technology with the use of a K-type thermocouple. The thermocouple measurements were tested in ultra-low temperatures successfully. The prototypes then transmit these measurements by LoRa and LoRaWAN to a back-end system with success.

The implementation of a complete back-end system solves the task of temperature monitoring with the storage of measurement data, data visualisation, and notification of crucial temperature levels. To solve the implementation of LoRaWAN in the back-end, the team utilized ChirpStack. The data processing was solved by implementing Telegraf and InfluxDB. Notifications were implemented through InfluxDB and were tested. Both email notification and SMS arrived at user end-point successfully.

The project team considers the prototype to be a good starting foundation. However, further research and development is required for a more completed product.

By trying to solve a relevant problem in society with new technology and compiling this knowledge into an overall understanding of how such a problem can be addressed, this project has an added value beyond the pure academical aspect.

7 List of words, acronyms, and expressions

Acronyms

ABP	Activation By Personalization.
API	Application Programming Interface
CPU	Central Processing Unit
ERP	Effective Radiated Power
FSK	Frequency Shift Keying
JSON	JavaScript Object Notation
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
MCU	Micro Controller Unit
MQTT	Message Queuing Telemetry Transport
OTAA	Over The Air Activation
SSL	Secure Sockets Layer
TLS	Transport Layer Security

Other Words or expressions

Protobuf	Protocol buffers

8 List of images and figures

Figures in 1.4.1 are all based on figures from Openclipart, which are free to use as one wants [4].

9 List of tables

10 List of C / C++ code snippets

11 References

[1] Hiof, “Studieplan for Bachelorstudium i ingeniørfag - elektro (2018–2021),” 05 June 2021. [Online]. Available: https://www.hiof.no/studier/programmer/ele-bachelorstudium-i-ingeniorfag-elektro/studieplaner/H2018.html. [Accessed 06 June 2021].

[2] Center for Disease Control and Prevention, “Pfizer-BioNTech COVID-19 Vaccine Storage and Handling Summary,” Pfizer-BioNTech, 21 May 2021. [Online]. Available: https://www.cdc.gov/vaccines/covid-19/info-by-product/pfizer/downloads/storage-summary.pdf. [Accessed 26 March 2021].

[3] LoRa Alliance, “What is LoRaWAN® Specification,” February 2021. [Online]. Available: https://lora-alliance.org/about-lorawan/. [Accessed 8 February 2021].

[4] Openclipart, “About Open Clipart,” 2021. [Online]. Available: https://openclipart.org/share. [Accessed 6 June 2021].

[5] A. Wiig, *Figure and figure versions describing the entire system implementation,* Trøgstad, 2021.

[6] S. Seletskyi, “LoRaWAN will temporarily replace 5G networks for IoT,” IoT Business News, 01 03 2020. [Online]. Available: https://iotbusinessnews.com/2020/11/03/90590-lorawan-will-temporarily-replace-5g-networks-for-iot/. [Accessed 21 May 2021].

[7] Engineering Product Design, “Rapid prototyping,” [Online]. Available: https://engineeringproductdesign.com/knowledge-base/rapid-prototyping-techniques/. [Accessed 31 May 2021].

[8] M. Raza, “OSI Model: The 7 Layers of Network Architecture,” 29 Juni 2018. [Online]. Available: https://www.bmc.com/blogs/osi-model-7-layers/. [Accessed 11 April 2021].

[9] Semtech, “What are LoRa® and LoRaWAN®?,” 2021. [Online]. Available: https://lora-developers.semtech.com/library/tech-papers-and-guides/lora-and-lorawan/. [Accessed 20 February 2021].

[10] M. Kaneko , A. Guitton and N. E. Rachkidy, “Decoding Superposed LoRa Signals,” *The 43rd IEEE Conference on Local Computer Networks,* pp. 184-190, 1 October 2018.

[11] Moko Smart, “What is the Technology Behind LoRa Frequency,” [Online]. Available: https://www.mokosmart.com/lora-frequency/?fbclid=IwAR1v07Yv_qgOceXFk0fz2bJkDM82KQfkJsIiTjTHhJ3R4GgUFjtq9flWwSU. [Accessed 5 Juny 2021].

[12] H. Mroue, A. Nasser, B. Parrein, S. Hamrioui, E. Mona-Cruz and G. Rouyer, “Analytical and Simulation study for LoRa Modulation,” *2018 25th International Conference on Telecommunications (ICT),* pp. 655-659, 26 September 2018.

[13] The things network, “LoRaWAN,” [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/index.html. [Accessed 13 April 2021].

[14] The LoRa Alliance, “What is LoRaWAN,” 2015. [Online]. Available: https://lora-alliance.org/resource_hub/what-is-lorawan/. [Accessed 17 April 2021].

[15] R. Prashant, “LPWAN, LoRa, LoRaWAN and the Internet of Things,” Medium, 2018.

[16] Techplayon, “LoRa- (Long Range) Network and Protocol Architecture & Frame Structure,” Techplayon, 2018.

[17] Lovdata, “Forskrift om generelle tillatelser til bruk av frekvenser (fribruksforskriften),” Lovdata, 31 January 2012. [Online]. Available: https://lovdata.no/dokument/SF/forskrift/2012-01-19-77. [Accessed 26 May 2021].

[18] MQTT.org, “MQTT.org,” MQTT.org, 2020. [Online]. Available: https://mqtt.org/faq/. [Accessed 07 April 2021].

[19] A. Gerber and R. J, “Connecting all things in the Internet of Things,” IBM Developer, 23 May 2017. [Online]. Available: https://developer.ibm.com/articles/iot-lp101-connectivity-network-protocols/. [Accessed 10 April 2021].

[20] ISO.org, “ISO/IEC 20922:2016 Information technology - Message Queuing Telemetry Transport (MQTT),” ISO.org, 08 June 2016. [Online]. Available: https://www.iso.org/standard/69466.html. [Accessed 08 April 2021].

[21] A. Banks, E. Briggs, K. Borgendale and R. Gupta, "MQTT Version 5.0, OASIS Standard," OASIS, 07 March 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf. [Accessed 01 April 2021].

[22] M. Yuan, “What is MQTT? Why use MQTT?,” IBM Developer, 12 May 2017. [Online]. Available: https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/. [Accessed 08 April 2021].

[23] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” IETF Tools, 20 March 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-28#section-4.1.2. [Accessed 01 April 2021].

[24] A. Minteer, “Chapter 2; IoT Devices and Networking Protocols,” in *Analytics for the internet of things (iot)*, Birmingham, Mumbai, PACKT Publishing Limited, 2017, pp. 24-60.

[25] Eclipse Mosquitto, “Eclipse Mosquitto,” Eclipse Foundation, [Online]. Available: https://mosquitto.org/. [Accessed 26 April 2021].

[26] Light, “Mosquitto: server and client implementation of the MQTT protocol,” Journal of Open Source Software, 2017. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.00265. [Accessed 27 April 2021].

[27] OMA SpecWorks, “OMA LightweightM2M (LwM2M) Object and Resource Registry,” 2021. [Online]. Available: http://openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html. [Accessed 20 May 2021].

[28] myDevices, “Cayenne docs - LoRa,” 2021. [Online]. Available: https://developers.mydevices.com/cayenne/docs/lora/#lora. [Accessed 20 May 2021].

[29] The Things Network, 2021. [Online]. Available: https://www.thethingsindustries.com/docs/integrations/payload-formatters/cayenne/. [Accessed 20 May 2021].

[30] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server - Application server - Device profiles,” 2021. [Online]. Available: https://www.chirpstack.io/application-server/use/device-profiles/#cayenne-lpp. [Accessed 20 May 2021].

[31] ChirpStack, “ChirpStack logo,” 2021. [Online]. Available: https://www.chirpstack.io. [Accessed 19 March 2021].

[32] ChirpStack, “The ChirpStack project,” 2021. [Online]. Available: https://www.chirpstack.io/project/. [Accessed 15 March 2021].

[33] ChirpStack, “ChirpStack community forum,” 2021. [Online]. Available: https://forum.chirpstack.io. [Accessed 23 March 2021].

[34] ChirpStack, “ChirpStack architecture,” 2021. [Online]. Available: https://www.chirpstack.io/project/architecture/. [Accessed 15 March 2021].

[35] gRPC, “About gRPC,” 2021. [Online]. Available: https://grpc.io/about/. [Accessed 4 June 2021].

[36] ECMA, “Introducing JSON,” [Online]. Available: https://www.json.org/json-en.html. [Accessed 1 June 2021].

[37] Google, “Protocol Buffers,” [Online]. Available: https://developers.google.com/protocol-buffers. [Accessed 1 June 2021].

[38] ChirpStack, “Introduction,” 2021. [Online]. Available: https://www.chirpstack.io/gateway-bridge/. [Accessed 15 March 2021].

[39] W. Stallings, Data and computer communications, Kolkata: Pearson, 2014.

[40] ChirpStack, “ChirpStack Gateway bridge - Requirements,” 2021. [Online]. Available: https://www.chirpstack.io/gateway-bridge/install/requirements/. [Accessed 19 March 2021].

[41] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server- Introduction,” 2021. [Online]. Available: https://www.chirpstack.io/network-server/. [Accessed 15 March 2021].

[42] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server - Requirements,” 2021. [Online]. Available: https://www.chirpstack.io/network-server/install/requirements/. [Accessed 15 March 2021].

[43] restfulapi.net, “What is REST,” 2021. [Online]. Available: https://restfulapi.net. [Accessed 4 June 2021].

[44] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server - Application server - Introduction,” 2021. [Online]. Available: https://www.chirpstack.io/application-server/.

[45] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server - Application server - use - Firmware update over the air,” 2021. [Online]. Available: https://www.chirpstack.io/application-server/use/fuota/. [Accessed 2 June 2021].

[46] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server - application server - Features,” 2021. [Online]. Available: https://www.chirpstack.io/application-server/features/. [Accessed 25 March 2021].

[47] Influx data, “Telegraf,” 2021. [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/. [Accessed 22 May 2021].

[48] Influx data, “Telegraf 1.18 documentation,” 2021. [Online]. Available: https://docs.influxdata.com/telegraf/v1.18/. [Accessed 22 May 2021].

[49] Influx data, “Documentation - Glossary - Time series data,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/reference/glossary/#time-series-data. [Accessed 27 May 2021].

[50] Influx data, “About InfluxData,” 2021. [Online]. Available: https://www.influxdata.com/about/. [Accessed 27 May 2021].

[51] Influx data, “InfluxDB,” 2021. [Online]. Available: https://www.influxdata.com/products/influxdb/. [Accessed 27 May 2021].

[52] S. /. I. data, “InfluxCommunity - Bucket - multiple data sources,” 26 May 2021. [Online]. Available: https://community.influxdata.com/t/bucket-multiple-data-sources/20072. [Accessed 27 May 2021].

[53] Influx data, “Documentation - Glossary - Bucket,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/reference/glossary/#bucket. [Accessed 27 May 2021].

[54] Influx data, “Documentation - Glossary . retention period,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/reference/glossary/#retention-period. [Accessed 27 May 2021].

[55] Influx data, “Process Data with InfluxDB tasks,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/process-data/. [Accessed 27 May 2021].

[56] Influx data, “Documentation - Visualize data with the InfluxDB UI,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/visualize-data/. [Accessed 27 May 2021].

[57] Influx data, “Documentation - Visualization types,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/visualize-data/visualization-types/. [Accessed 27 May 2021].

[58] Influx data, “Documentation - Monitor data and send alerts,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/monitor-alert/. [Accessed 27 May 2021].

[59] Influx data, “Documentation - Glossary - notification rule,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/reference/glossary/#notification-rule. [Accessed 1 June 2021].

[60] Influx data, “Create notification endpoints,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/monitor-alert/notification-endpoints/create/. [Accessed 1 June 2021].

[61] Influx data, “Send alert email,” 2021. [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/monitor-alert/send-email/. [Accessed 1 June 2021].

[62] Fluke, “Thermocouple Theory,” [Online]. Available: https://www.flukeprocessinstruments.com/en-us/service-and-support/knowledge-center/thermal-profiling-technology/thermocouple-theory. [Accessed 15 May 2021].

[63] R. Radebaugh, “Cryogenic Measurements,” in *Handbook of Measurement in Science and Engineering, Volume 3*, Hoboken, John Wiley and Sons, 2015, pp. 2183-2199.

[64] Atmel, “8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash (Datasheet),” January 2015. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. [Accessed 15 Mai 2021].

[65] Wiki Dragino, “Lora Shield (Datasheet),” 20 October 2020. [Online]. Available: https://wiki.dragino.com/index.php?title=Lora_Shield#Power_Consumption. [Accessed 15 May 2021].

[66] Dragino, “IoT Development Kit featuring LoRa® technology,” 2012. [Online]. Available: https://www.dragino.com/products/lora/item/120-lora-iot-kit.html. [Accessed 4 June 2021].

[67] Dragino, “LG01-N Single Channel LoRa IoT Gateway,” 23 December 2020. [Online]. Available: https://www.dragino.com/products/lora/item/143-lg01n.html. [Accessed 31 May 2021].

[68] MCCI, “Arduino-LMIC library,” 2021. [Online]. Available: https://github.com/mcci-catena/arduino-lmic. [Accessed 02 April 2021].

[69] MCCI, “Arduino-LMIC - pin-mapping,” 2021. [Online]. Available: https://github.com/mcci-catena/arduino-lmic#pin-mapping. [Accessed 02 April 2021].

[70] LoRa Alliance, “RP2-1.0.2 LoRaWAN® Regional Parameters,” 8 October 2020. [Online]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/RP_2-1.0.2.pdf. [Accessed 19 March 2021].

[71] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server - Application server - Integrations - InfluxDB,” [Online]. Available: https://www.chirpstack.io/application-server/integrations/influxdb/. [Accessed 22 May 2021].

[72] ChirpStack, “InfluxDB 2.0 integration support #409,” 8 Jan 2020. [Online]. Available: https://github.com/brocaar/chirpstack-application-server/issues/409. [Accessed 20 May 2021].

[73] D. McKay, Influx data, 15 August 2019. [Online]. Available: https://www.influxdata.com/blog/influxdb-2-migration-path-instrumentation/. [Accessed 22 May 2021].

[74] Influx data, “MQTT Consumer Input Plugin,” 4 June 2020. [Online]. Available: https://github.com/influxdata/telegraf/blob/release-1.18/plugins/inputs/mqtt_consumer/README.md. [Accessed 22 May 2021].

[75] Influx data, “InfluxDB v2.x Output Plugin,” 7 October 2020. [Online]. Available: https://github.com/influxdata/telegraf/blob/release-1.18/plugins/outputs/influxdb_v2/README.md. [Accessed 29 May 2021].

[76] Pagerduty, “Events API v2 Overview,” 2021. [Online]. Available: https://developer.pagerduty.com/docs/events-api-v2/overview/. [Accessed 3 June 2021].

[77] Pagerduty, “InfluxData Integration Guide,” 2021. [Online]. Available: https://www.pagerduty.com/docs/guides/influxdata-integration-guide/. [Accessed 2 June 2021].

[78] ChirpStack, “ChirpStack open-source LoRaWAN® Network Server - Application Server - Integrations - PostgreSQL,” 2021. [Online]. Available: https://www.chirpstack.io/application-server/integrations/postgresql/. [Accessed 26 May 2021].

[79] O. Brocaar, “Storing device events into PostgreSQL database,” 19 May 2019. [Online]. Available: https://forum.chirpstack.io/t/storing-device-events-into-postgresql-database/4724/4. [Accessed 22 May 2021].

[80] Influx data, “Time series database (TSDB) explained,” [Online]. Available: https://www.influxdata.com/time-series-database/. [Accessed 1 June 2021].

[81] Influx data, “InfluxDB tools and integrations,” [Online]. Available: https://docs.influxdata.com/influxdb/v2.0/tools/. [Accessed 1 June 2021].

[82] Influx data, “Integrations,” 2021. [Online]. Available: https://www.influxdata.com/products/integrations/?_integrations_dropdown=dashboard. [Accessed 2 June 2021].

[83] Influx data, “Integrations - Incident management,” 2021. [Online]. Available: https://www.influxdata.com/products/integrations/?_integrations_dropdown=incident-management. [Accessed 2 June 2021].

[84] Influx data, “Integrations - Telegraf plugins,” 2021. [Online]. Available: https://www.influxdata.com/products/integrations/?_integrations_dropdown=telegraf-plugins. [Accessed 2 June 2021].

[85] S. v. P. World, “Softbox Supports Pfizer in the Global Cold Chain Distribution of COVID-19 Vaccines,” Softbox/Packworld, 01 04 2021. [Online]. Available: https://www.packworld.com/home/news/21354870/softbox-supports-pfizer-in-the-global-cold-chain-distribution-of-covid19-vaccines. [Accessed 26 May 2021].

[86] Controlant, “Controlant now providing monitoring and Supply Chain Visibility for Pfizer-BioNTech COVID-19 Vaccine distribution and storage,” Controlant, [Online]. Available: https://controlant.com/blog/2020/controlant-now-providing-monitoring-and-supply-chain-visibility-for-pfizer-biontech-covid-19-vaccine-distribution-and-storage/. [Accessed 26 May 2021].

[87] OrionM2M, “MONITORING OF COVID-19 VACCINES STORAGE AND TRANSPORTATION,” [Online]. Available: https://lora-alliance.org/wp-content/uploads/2021/02/Orion-Solution-Covid-ENG.pdf. [Accessed 1 Juny 2021].

[88] The Things Network, “Fair Use Policy,” 2021. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/. [Accessed 20 May 2021].

[89] The Things Network, “Fair Use Policy explained,” February 2016. [Online]. Available: https://www.thethingsnetwork.org/forum/t/fair-use-policy-explained/1300. [Accessed 20 May 2021].

[90] The Things Industires, “The Things Stack,” 2021. [Online]. Available: https://www.thethingsindustries.com/stack/. [Accessed 20 May 2021].

[91] The Things Industries, “Fee calulator,” 2021. [Online]. Available: https://accounts.thethingsindustries.com/fee-calculator/. [Accessed 20 May 2021].

[92] JLCPCB, “ JLCPCB,” 2021. [Online]. Available: https://jlcpcb.com. [Accessed 1 June 2021].

12 Appendices

Appendix 1. LoRa Modulation

Appendix 2. ChirpStack gateway bridge deployment

Appendix 3. ChirpStack Network server features

Appendix 4. Hardware

Appendix 5. Prototype 1

Appendix 6. Prototype 2 source code

Appendix 7. Installation manual of the back-end system

Appendix 8. Test of thermocouple

Appendix 9. Email communication with Dragino

Appendix 10. Limitations for LG01 excerpt dragino kit manual

Appendix 11. STM32 Programming