Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Otto Laine, Larri Myllymäki, Eetu Perälä

IoT Remote Monitoring System Utilizing LoRa

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's thesis

22.4.2019

Tekijä Otsikko Sivumäärä Aika	Otto Laine, Larri Myllymäki, Eetu Perälä IoT-etävalvontajärjestelmän Toteuttaminen LoRa-verkkoa Käyttäen 38 sivua 22.4.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Tietoverkot ja sovellukset
Ohjaajat	Lehtori Jukka Louhelainen

Insinöörityön tarkoituksena oli luoda prototyyppi IoT-järjestelmästä, minkä tehtävänä oli useiden eri laitteiden etävalvonta. Projekti toteutettiin yritykselle, jonka nimeä ei voida salassapitosopimuksen takia paljastaa. Samasta syystä useita teknisiä yksityiskohtia ei olla voitu selittää tarkasti ja laitteen valvomista asioista käytetään vain nimitystä "measurement point” (mittauspiste).

Projekti oli toteutettava noudattaen yrityksen antamia määrittelyjä, kuten LoRa:n käyttäminen laiteverkkona ja julkisen internetin käyttäminen runkoverkkona. Projektia varten tehtiin laite, joka kykenee mittaamaan useita mittauspisteitä sekä välittämään tietoja niiden tilasta. Laitteen lähettämä tieto tulisi käsitellä muotoon jossa se voitaisiin esittää verkkopohjaisessa applikaatiossa, joka tuli kehittää.

LoRa-laitteisto saatiin asiakasyritykseltä, mutta lähes kaikki muut prototyypin osat oli rakennettava alusta lähtien. Laite ohjelmoitiin C:llä. JavaScript, PHP sekä Python olivat käytössä verkkopalvelimella ja MQTT:tä käytettiin viestintäprotokollana.

Projekti saatettiin onnistuneesti loppuun ja kaikki yrityksen määrittelemät vaatimukset täytettiin. Laite ja kaikkien kokonaisuuden osien lähdekoodit toimitettiin yritykselle yksityiskohtaisen teknisen dokumentaation kera. Yritys oli valmiina ottamaan järjestelmän käyttöön välittömästi.

Avainsanat	IoT, LoRa, Internet of Things, MQTT, projekti

Author Title Number of Pages Date	Otto Laine, Larri Myllymäki, Eetu Perälä IoT Remote Monitoring System Utilizing LoRa 38 pages 22 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Communication Networks and Applications
Instructors	Jukka Louhelainen, Senior Lecturer

The objective of this thesis was to create a prototype of an end-to-end IoT system for remotely monitoring a large number of devices. The project was completed for a company whose name, along with many technical details about the project, cannot be disclosed due to a non-disclosure agreement. For the same reason, the items that the system is monitoring, or measuring, are simply called measurement points.

The project had to follow specifications defined by the company, which included LoRa being used as the device network and the public internet as the backbone network. A device was to be created in order to monitor several measurement points and communicate their states. This data was to be processed into a form that could be displayed on a web based front end application, which also had to be developed.

While the LoRa equipment was provided by the company, nearly everything else had to be purpose-built. The device was programmed in C. JavaScript, PHP and Python were used on the web server, and MQTT was used as the messaging protocol.

Ultimately the project was completed successfully, and all the specified requirements were met. The device and the source code of all the parts in the architecture were delivered to the company, along with detailed technical documentation. The company was prepared to immediately begin using the system.

Keywords	LoRa, IoT, Internet of Things, MQTT, project

Contents

Abbreviations

IoT	Internet of Things, the concept of a massive amount of interconnected physical “things”, such as the temperature of a room or the controls of a larger device.
MQTT	MQ Telemetry Transport or Message Queuing Telemetry Transport, a light-weight publish-subscribe based messaging protocol.
LDR	Light Dependent Resistor, an electrical component whose resistance changes with light intensity.
WAN	Wide Area Network, a network that extends over a large geographical area.
LoRa	Long Range, low power modulation technique developed by Semtech.
LoRaWAN	Long Range, low power network protocol owned by LoRa Alliance.

1 Introduction

This Bachelor's thesis project was completed for a customer company. The project goal was to create a prototype of an end-to-end Internet of Things (IoT) system to monitor a large number of devices. The system should use Semtech's low power, long range wireless network technology LoRa as the device network and public internet for the backbone network to reduce the need for new infrastructure. The system should also be designed in a way where it could be used in a wide variety of situations. The LoRa stack was provided by the company and was outside of the project's control.

The rise in demand for IoT solutions has led to the development of a massive number of new technologies. Choosing the correct technology for a specific use-case is not always clear with the number of new cases and new technologies, and one of the goals of this project was to provide an example of an IoT system built around a LoRa network.

IoT can be generally thought of as a network of interconnected objects that exist in the physical world, such as an industrial robot or the temperature of a room, and objects that exist in the virtual world, such as software applications and digital files. An IoT system generally consists of sensing or actuating devices, a network to transfer the data, an aggregator to gather the data, and applications that leverage the data.

A common problem that IoT solves is the need to connect a large number of nodes to the Internet for monitoring and remote control. While a single IoT system is nearly identical to a traditional remote-monitoring or remote-control system, the possibility to interconnect these systems allows for more complex applications.

2 Project Description

The project was completed confidentially through Metropolia University of Applied Sciences for the customer company. The work area and computers were provided by Metropolia while the hardware for the prototype was provided by the company. All project files and hardware were treated as confidential and the project was worked on in lockable areas with limited access at Metropolia's Leppävaara campus.

The goal of the project was to create an IoT remote monitoring system where a simple device would be used to monitor the state of the target and transmit the state for visualization at remote points. The customer requirements were:

- That optical sensors be used for the measurements.
- That LoRa be used for the device connection.
- That the measured states could be visualized with coordinates on a map.
- That the visualizing application be a web application that can be opened on multiple stations simultaneously.

The customer provided the project with:

- A LoRa gateway
- An STMicroelectronics STM32 Nucleo pack (includes an STM32L073RZT6 board and an SX1272MB2DAS LoRa expansion board)
- An STMicroelectronics Nucleo multi-sensor expansion board
- A testing environment for the measurement target

The LoRa stack was outside the project's control. Excluding it, the project was responsible for the overall system architecture. Major tasks included designing and implementing the sensor device, connecting the device to the LoRa network, designing and implementing the network after LoRa, and designing and implementing the monitoring application.

3 IoT Architecture

IoT can be generally thought of as a network of interconnected objects that exist in the physical world, such as the temperature of a room or an industrial robot, and objects that exist in the virtual world, such as software applications and digital files. These objects should be identifiable and connectable to the network [1]. For virtual objects this is simple, but with physical objects sensing or actuating devices must be used. By leveraging the interconnection of these objects, a wide variety of advanced services can be realized. Examples of common applications include predictive maintenance and smart transportation. It is important to note that systems of interconnected devices or classical automation systems are not IoT by themselves but can be connected to become a part of IoT [2].

3.1 Layers

Since there is currently no one commonly agreed architectural model for IoT, the model varies by the project goals. The design of this project and this document used the four-layered architecture model outlined by International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T). This model is shown in Figure 1. It was chosen as its layers match the goals of this project.

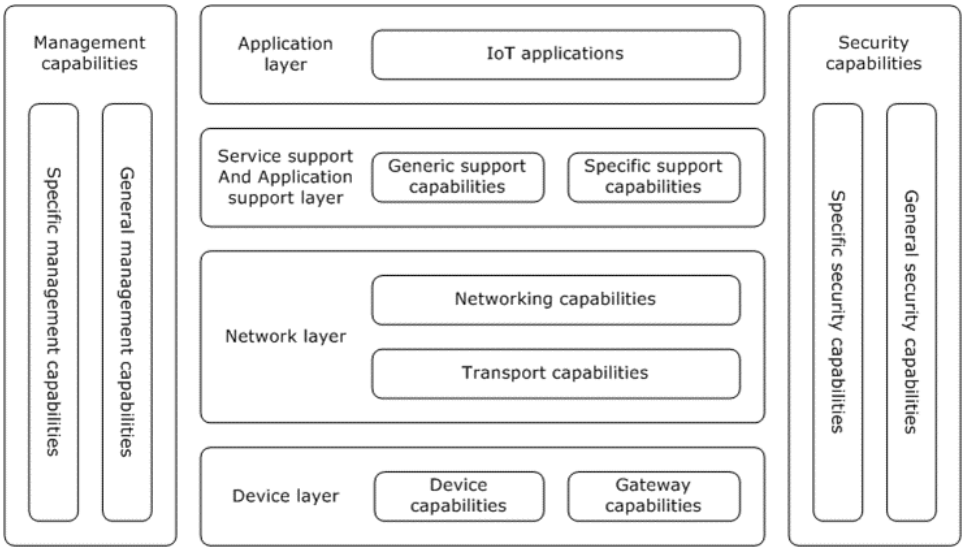

Figure 1. IoT reference model introduced by ITU-T [1].

This model divides IoT into four layers: application layer, network layer, service support and application support layer, or service layer, and device layer. Required management and security capabilities are also outlined by this model but were outside the scope of this project. The device layer contains the devices that interact and measure physical objects. The network layer contains the infrastructure and protocols that facilitate the transfer of information between devices and applications. The service layer provides middleware and services for applications and devices. The application layer contains IoT applications. [1], [3].

3.1.1 Device layer

Responsibilities of this layer include measuring, gathering and processing information from sensors, interacting with physical objects through actuators or other means,

identifying and tracking the devices and connecting the devices to a communication network. This layer contains IoT devices and gateways [1]. [4], [5], [2].

Devices are directly connected to the sensors and actuators of the system [6], and can connect to the network directly or through a gateway [1]. Important considerations for these devices include their power usage [7], complexity and resources such as computing capabilities and storage [2]. These considerations affect the cost of these devices, and as it might be necessary to have a very large number of them, even small reduction in cost can be important. In the case of battery-powered devices low power consumption will directly affect the maintenance required by these devices [7].

Gateways are used to connect devices to the network. They include interfaces for device connections and a connection to the wider network, such as the Internet or the publicly switched telephone network [1]. Gateways are also responsible for any protocol translations necessary between the networks [1]. The important considerations with gateways are similar to devices, but as they are often used to compensate for device limitations, they are usually less limited than devices [6]. Their role also means that there are less gateways than devices, as one gateway should be able to support multiple devices. This makes it less important for them to be extremely cheap and means that it is more feasible to install them in such a way that they have easy access to power.

3.1.2 Network layer

Network layer responsibilities include the transport of IoT data and control information, network resource control, and mobility management [1]. This layer is composed of heterogenous networks and includes both the connections between devices and gateways, and the networks connecting them to the service layer [2]. The areas that are specific to IoT are the networks connecting devices and gateways, the access networks used to connect gateways or devices to the internet and the application protocols that define the communication process. An important note for these networks is that IoT traffic is generally uplink-dominant, most traffic generated by IoT is sent from the device layer rather than being received by it [8].

Device-to-gateway connections are used to connect low power or resource constrained devices to more capable gateways. Depending on the system’s needs the device-to-

gateway connection type can vary wildly, from short-range Personal Area Network (PAN) technologies to long-range Low Power Wide Area Network (LPWAN) technologies. Because network connections make up a major part of a device's power consumption [7], most technologies here focus on being less power and resource intensive. It is also important to consider that while a single device rarely needs a large amount of network capacity, there can be a massive amount of them. This is reflected in the scalability of the different technologies, where the number of devices each gateway can support and how complex the gateways need to be must be considered. Other considerations are interoperability with other technologies and already existing infrastructure.

Access networks connect gateways or devices to wider networks, such as the internet or company intranets. These include wireless and wired broadband technologies and some Local Area Network (LAN) technologies. Because many of these technologies are existing infrastructure, the needs of IoT were not considered when they were designed, and so gateways need to be designed to fit the available access networks instead.

Application protocols are used for communication and other services. The word refers to Open Systems Interconnection model (OSI model) layer 7 and not the IoT architecture layer. These are protocols that work over Transmission Control Protocol/Internet Protocol (TCP/IP) or User Datagram Protocol/Internet Protocol (UDP/IP) and carry data from device layer to the service layer or provide other functions such as name resolution and service discovery. The important considerations for these protocols are protocol overhead, power consumption and latency [9], [10].

3.1.3 Service support and application support layer

This layer is responsible for collecting, exchanging and storing data, managing application protocol connections from the device layer, providing abstraction of device functions, communication capabilities, technologies and applications, and providing primary and secondary services. [2]

Primary services, or generic support capabilities, are common services that applications or devices use, and can be directly related to the functionality of devices, applications or the service layer itself. Secondary services, or specific support capabilities, are more

specialized services that are used by specific applications and can consist of primary services or other secondary services [1], [2].

The abstraction provided by the service layer means that the underlying technologies are not exposed in the services provided by this layer. The abstract representations of device functions and applications are common regardless of the actual protocols and hardware used achieve them. This allows heterogenous networks, devices and applications to be used, and enables the development of applications without compatibility considerations [2]. Most importantly it means that changes in one part of the system, such as replacing devices with ones that use different protocols, will not influence the function of other parts of the system.

To interact with devices and applications the service layer must be able to manage connections [2]. This not only allows the collection and exchange of data, but without knowledge of the device or application connection status it would be impossible to determine which services are available.

3.1.4 Application layer

The application layer contains the IoT applications or virtual objects that are part of the IoT network. While this layer is mostly ignored by the ITU-T model [1], it contains much of what allows IoT solutions to function. The exact requirements and capabilities of this layer are specific to the solution and cannot easily be generalized.

4 LoRa

This section gives an overview of the LoRa protocol stack and its basic network architecture.

4.1 LoRa and LoRaWAN

LoRa is a modulation technology developed and owned by Semtech. Modulation in LoRa (Long Range) happens on OSI layer 1. LoRa is designed to be a simple low cost and

low power implementation targeting IoT applications. Using Chirp Spread Spectrum (CSS) for low interference can achieve a long-range, low-power, low-bitrate network. LoRa is often promoted as an industrial solution for the Internet of Things. Typical battery life for low cost LoRa applications is about 10 years. LoRaWAN is an OSI layer 2 protocol. It is developed by LoRa-alliance. LoRaWAN consist of a whole network stack including end devices, servers, security and communication. [11]

4.2 Three main LPWAN technologies

There are multiple Low Power Wide Area Network (LPWAN) wireless communication technologies available. However, only a few of them are suitable for large scale IoT deployment or industrial solutions. In terms of availability, cost efficiency and private management there are three technologies that dominate the market. These three are SigFox, LoRa and NB-IoT. These are compared in Figure 2. [12]

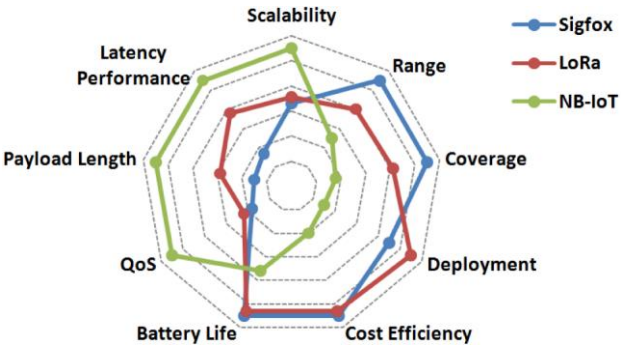

Figure 2. Technical key differences between Sigfox, LoRa and NB-IoT [12].

Which of these is best suited for IoT solutions depends on its feasibility for specific applications. Technical differences and restrictions limit cases where different technologies could be used. [12]

4.3 Network architecture

LoRa networks are typically a spine-leaf architecture that can also be called a “star-of-stars” architecture. End devices are connected to gateways and gateways are connected to network servers. End devices collect all the data and forward raw LoRaWAN frames

from devices to a network server. Servers and Gateways are connected to each other via an IP connection to utilize higher throughput. The IP connection is typically a cellular or Ethernet connection. Figure 3 shows how devices in LoRa are interconnected to each other. [11]

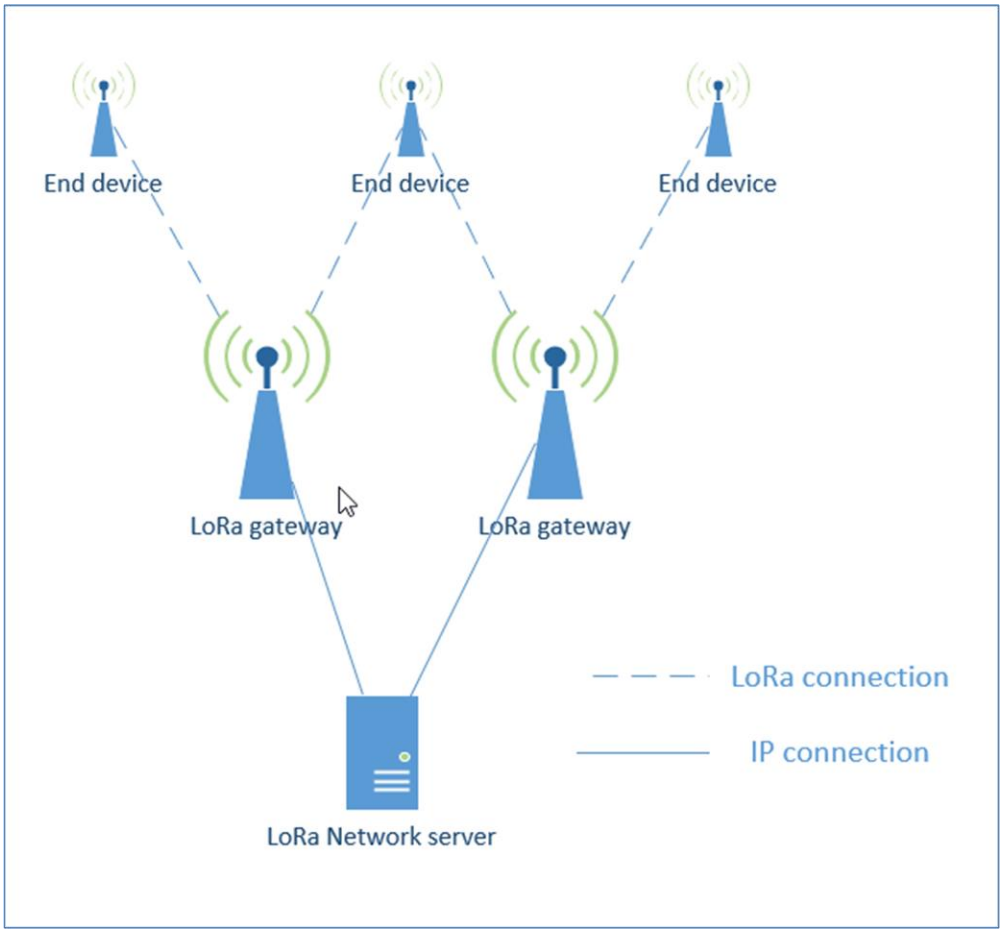

Figure 3. LoRa network architecture.

LoRa technology is optimized for energy efficiency. One gateway can simultaneously receive traffic from multiple end devices. The end devices can be connected to one or more default gateways via single hop wireless communication. End-point communication is bi-directional with a short receive window following transmissions and can support multicast with a software upgrade. If an end device is mobile or moving there is no hand-over needed from gateway to gateway. [13]

Communication between LoRa devices and gateways is spread to different frequency channels and different data rates. Data rate is dependent on two variables: distance

between end device and gateway, and message duration. LoRa utilizes chirp spread spectrum technology with adaptive data rate (ADR). Channels do not interfere with each other. LoRa data rates can vary between 0,3 kbps to 50 kbps. To maximize battery life and end device network capacity, the network server is responsible for managing RF (Radio Frequency) output individually by adaptive data rate. [13], [11].

4.4 LoRa frequencies

LoRa operates in 433, 868 or 915MHz Industrial, Scientific and Medical (ISM) bands, depending on the region in which it is deployed. Network connectivity in different regions of the world such as the US, EU and China require different frequency channels. Both the gateway and the end device can use the same frequency for transmission, but different timeslots are used. This concept is known as Time Division Duplex (TDD). [11]

	Europe	North America	China	Korea	Japan	India
Frequency band	867-869MHz	902-928MHz	470-510MHz	920-925MHz	920-925MHz	865-867MHz
Channels	10	64 + 8 +8	In definition by Technical Committee	In definition by Technical Committee	In definition by Technical Committee	In definition by Technical Committee
Channel BW Up	125/250kHz	125/500kHz				
Channel BW Dn	125kHz	500kHz				
TX Power Up	+14dBm	+20dBm typ (+30dBm allowed)				
TX Power Dn	+14dBm	+27dBm				
SF Up	7-12	7-10				
Data rate	250bps- 50kbps	980bps-21.9kpbs				
Link Budget Up	155dB	154dB				
Link Budget Dn	155dB	157dB				

Figure 4. LoRa bands, data-rates and transmit power. [13]

As seen in Figure 4, there are ten communication channels in the EU. Eight out of ten are multi data rate channels, which are capable of data rates from 250 bits/s to 5,5kbit/s. One channel operates at a higher data rate of 11kbit/s, and another Frequency Shift Key channel operates at 50 kbit/s. [13]

4.4.1 Data rates in LoRa bands

There are several parameters that affect the modulation and data rates in LoRa. These are Bandwidth (BW), Spreading Factor (SF) and Code Rate (CR). The gross bit rate (R_b) in LoRa is calculated as shown in equation (1).

$$R_b = SF \; x \; \frac{BW}{2^{SF}} \; x \; CR \tag{1}$$

R_b is the bit rate of the channel
SF is the Spreading Factor of the communication channel
BW is the bandwidth of the communication channel
CR is the code rate of the communication channel [14]

Higher BW and SF affect the minimum sensitivity of the receiving device. Higher values would decrease receiving sensitivity. As an example, sensitivity values for Semtech SX1276 as affected by bandwidth and spreading factor are shown in Figure 5. [15]

		Spreading Factor					
RF sensitivity (dBm)		7	8	9	10	11	12
Bandwith kHz	125	-123	-126	-129	-132	-133	-136
	250	-120	-123	-125	-128	-130	-133
	500	-116	-119	-122	-125	-128	-130

Figure 5. Teceiver sensivity of SX1276/77/78/79.

Higher bandwidth doubles the bit rate as seen in Figure 6 below, but it also decreases receiver sensitivity as seen above in Figure 5. LoRa functions better with narrower bandwidths and lower spreading factors.

Coding rate 4/5		Spreading Factor					
Bit rate kbit/s		7	8	9	10	11	12
Bandwith kHz	125	5,46875	3,125	1,75781	0,97656	0,53711	0,29297
	250	10,9375	6,25	3,51563	1,95313	1,07422	0,58594
	500	21,875	12,5	7,03125	3,90625	2,14844	1,17188

Figure 6. Data rates with different BW and SF calculated using formula 1.

A LoRa end device may choose any channel available at any time if the following requirements are met:

> The end-device changes channel in a pseudo-random fashion for every transmission. The resulting frequency diversity makes the system more robust to interferences.
>
> The end-device respects the maximum transmit duty cycle relative to the sub-band used and local regulations.
>
> The end-device respects the maximum transmit duration (or dwell time) relative to the sub-band used and local regulations. [16, p. 8]

4.5 ALOHA method in LoRa

ALOHA is a system for multiple device to share the same communication medium. ALOHA defines ways of handling collision errors and interference in the communication medium. ALOHA is a random-access protocol and operates at the datalink layer (OSI Layer 2). The ALOHA method and its variants are shown in Figure 7. [17]

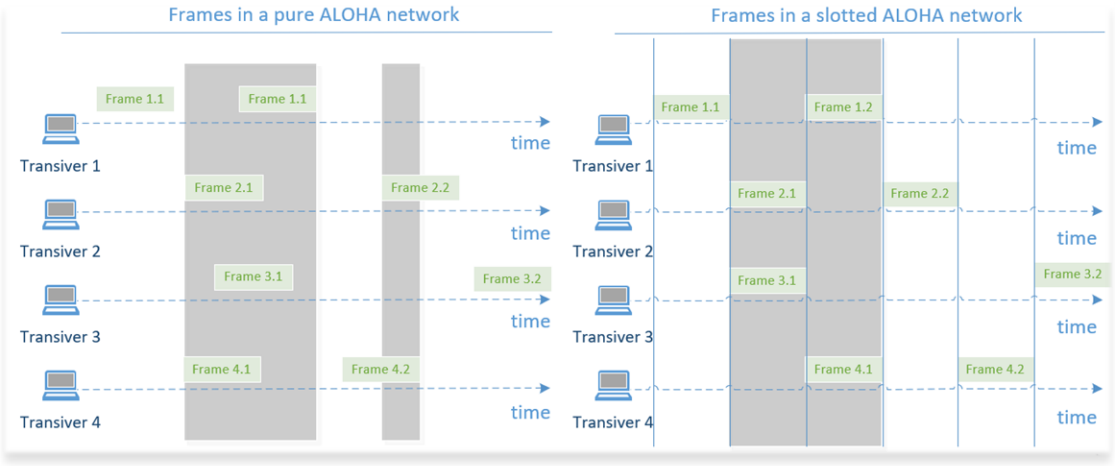

Figure 7. Collisions in different ALOHA methods.

In regular ALOHA, also called pure ALOHA, devices transmit frames as needed with no synchronization with each other. If a frame is received with no errors, it is acknowledged, and the transfer is done. If the sending device does not receive an acknowledgement the frame is assumed to be destroyed and a new one is sent after a random wait time. With no synchronization between the sending devices a common reason for frames to be lost is frame collision occurring due to multiple devices transmitting simultaneously, an example of which is shown in Figure 7, with only two frames surviving intact. [17]

An improved version called slotted ALOHA splits the channel into timeslots. Devices still choose to transmit on their own but must transmit their frame within one of these timeslots. As shown in Figure 7 this maximizes the channel utilization and doubles the efficiency of pure ALOHA with four surviving frames, but requires time synchronization between the devices, losing the advantages of an entirely unsynchronized network. [18]

4.6 Security

Security is an essential aspect of communication networks. LoRaWAN has two layers of security: one for the network layer and another in the application layer. Application layer security ensures that the network operator does not have access to the end user application data. Network layer security ensures that the traffic is secured. AES (Advanced Encryption Standard) encryption is used. Figure 8 represents these two layers of security as they are implemented in LoRaWAN architecture. [19]

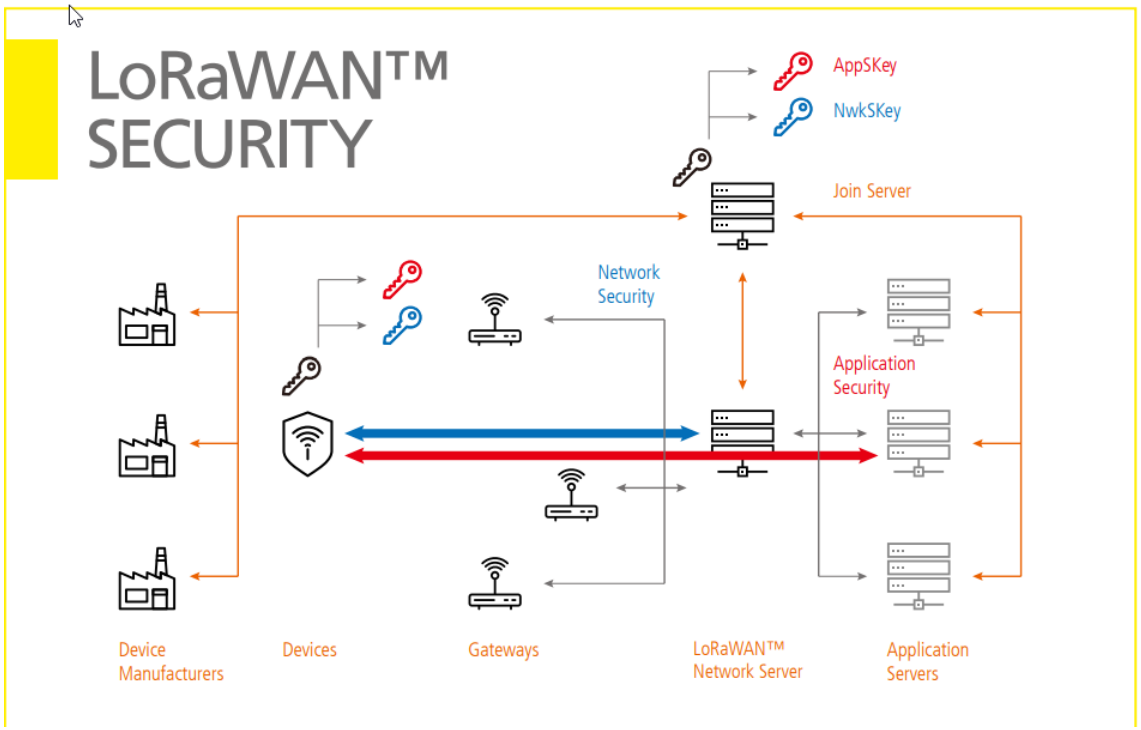

Figure 8. Two-layer security in LoRaWAN. [20]

These layers differ in which encryption keys are used and which devices use them:

- Network layer security is implemented with a Network Session Key (NwkSKey). 128-bit AES encryption is used. It takes place between end devices and the network server. [19]
- Application layer security is implemented with an Application Session Key (AppSKey). 128-bit AES encryption is used between end devices and the application server. [19]

4.7 Joining a LoRa network

There are two methods by which Nodes are allowed to join a LoRa network. These methods are called Over-The-Air-Activation (OTAA) and Activation by Personalisation (ABP). OTAA devices must have a DevEUI (Device Extended Unique Identifier) stored in the device, and it is recommended but not necessary for ABP devices to do the same. The DevEUI is a globally unique identifier for every end device. [16]

4.7.1 Over the Air Activation

Devices can be allowed to join a LoRa network through OTAA. Each device is deployed with a unique 128-bit app key (AppKey) which is used when the device sends a join-request message. This message is not encrypted, but it is signed using the AppKey. The device then sends its AppEUI, DevEUI and DevNonce signed with a 4-byte Message Integrity Check (MIC). The server re-calculates the MIC with the AppKey. If the result is valid, the server may respond with a join-accept message. The device calculates the NwkSKey and AppSKey based on the values sent to it in the join-request message. [19]

4.7.2 Activation by Personalisation

ABP differs from OTAA as the devices are shipped with a DevAddr and both session keys (NwkSKey and AppSKey), which should be unique to the node. As these devices already have the information and keys they need, they can begin communication with the Network Server without the need for join messages. [19]

4.8 End-device classes

End-devices serve different kinds of applications. Applications have different latency requirements. Communication methods of LoRa devices fall in three categories. Classes and their place in the LoRa protocol stack are shown in Figure 9.

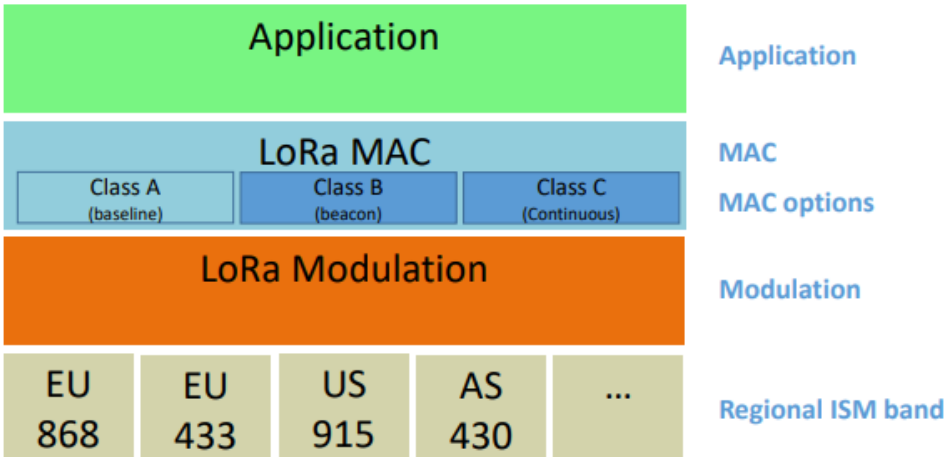

Figure 9. Classes in LoRaWAN stack. [16]

Categories are defined by different end device downlink receiving slots and availability to send data. At the lowest service level, battery life is maximized, and latency is at its highest.

4.8.1 Class A end devices

Bi-directional end devices of Class A allow for bi-directional communications where each end device's uplink transmission is followed by two short downlink receive slots. The server communicates with end devices (downlink) after the end device has transmitted. Every uplink transmission is followed by two short downlink receive windows. This communication sequence is shown in Figure 10.

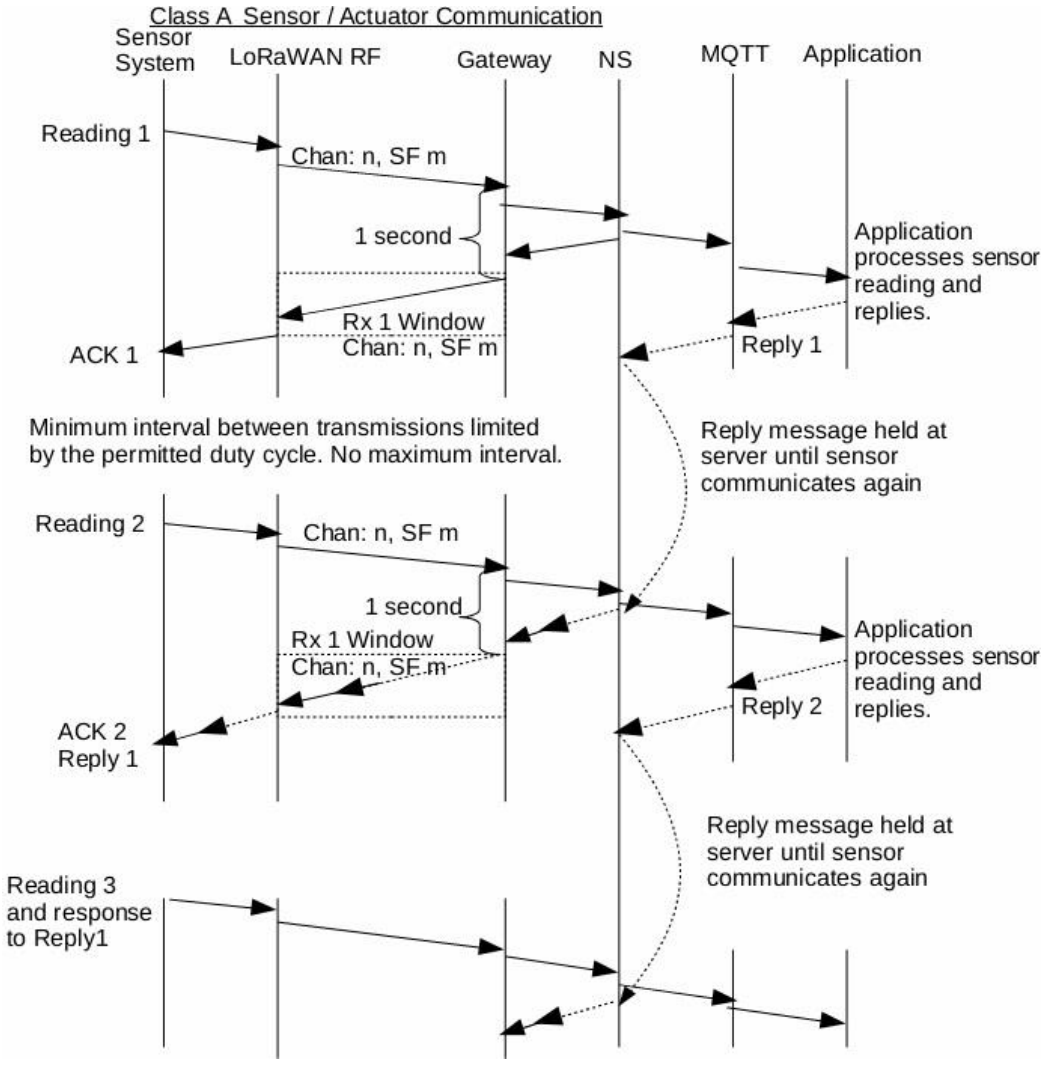

Figure 10. LoRa class A end device and gateway communication.

Transmission slots scheduled by the end device are based on its communication needs. Class A end devices are pure ALOHA type devices, and they have the lowest power consumption. Class A can be used in end devices where downlink communication is a low priority. In LoRaWAN every device must support the class A communication method. [16]

4.8.2 Class B end devices

Class B consists of bi-directional end devices with scheduled receive slots. In addition to the Class A random receive windows. there is one scheduled receive slot called a Beacon period. The Beacon period is scheduled by the gateway. Class B end devices can send unicast and multicast messages, and they are considered to be very low latency. [16]

4.8.3 Class C end devices

Class C is for end devices with bi-directional communication and maximal receive slots. These end devices will receive almost continuously, except when transmitting. Class C end devices can send unicast and multicast messages. Class C devices are considered as no latency devices, and they usually have a fixed power supply because of high energy consumption. [16]

5 MQTT

MQTT (MQ Telemetry Transport or Message Queuing Telemetry Transport) is an ISO (International Organization for Standardization) standard (ISO/IEC PRF 20922:2016) [21]. MQTT is a client/server publish/subscribe messaging transport protocol. MQTT is designed to be light, open and easy to implement. [22]

It is ideal for use in machine-to-machine (M2M) situations and Internet of Things (IoT) solutions. MQTT messages are lightweight and require little bandwidth. The protocol is used over TCP/IP or others that provide ordered, lossless, bi-directional connections. [22]

5.1 Versions

The original MQTT was designed in 1999 by Dr. Andy Stanford-Clark of IBM and Arlen Nipper of Acrom. Initially, MQTT was designed for use in TCP/IP networks [23]. The previous standard of MQTT was 3.1.1. A new version of MQTT 5.0 was launched in 2017, and it was published as a standard in March of 2019. [24]

5.2 Model

MQTT uses a publish / subscribe model which requires central server. Figure 11 represents a simplified MQTT model [22].

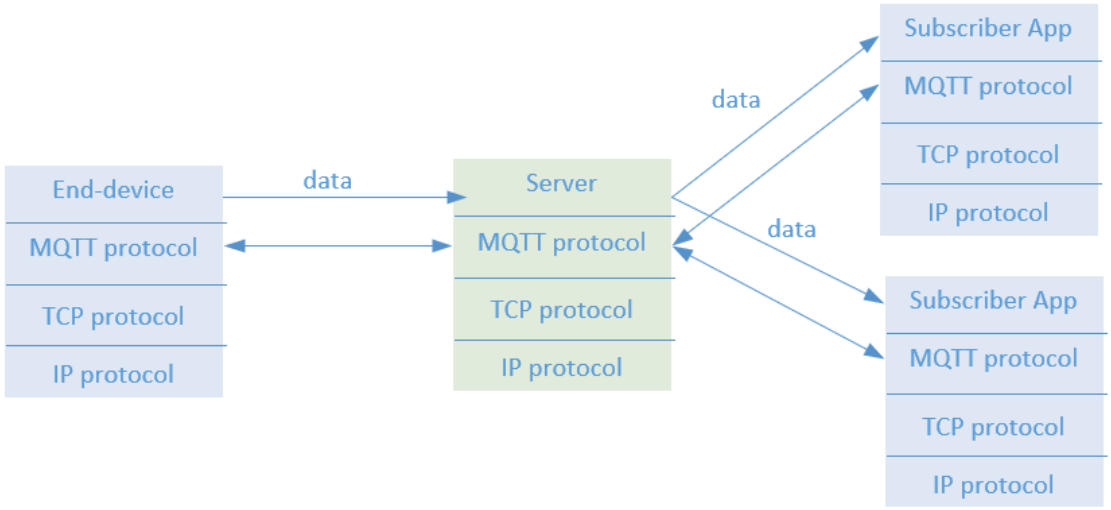

Figure 11. MQTT end-to-end connectivity model.

In MQTT, a data collecting end device is connected to a server. The server distributes messages onwards to subscribed clients. The MQTT protocol is bidirectional. One service or device can act as a publisher and a subscriber. [22]

5.3 MQTT Server

The MQTT server is the counterpart to the MQTT client. The server is responsible for accepting network connections from clients, receiving all messages, filtering them and

sending them to all subscribed clients. It holds data of all persisted clients including subscriptions and missed messages. Another responsibility of the server is the authentication and authorization of clients. [22]

5.4 MQTT clients

An MQTT client is a device or application which is connected to an MQTT server with subscribe/publish messages. An MQTT client can act as a publisher, subscriber or both. The client can be a small device or a full-fledged server as long requirements to connect to the MQTT server are fulfilled. When a client publishes a message belonging to a topic, the server will then distribute that message to any client that is subscribed to the topic. [22]

5.5 Topic

An MQTT server is usually centralized for serving multiple different areas and running multiple different cases. Topics are used to separate and organize by different areas. Topics are simple, hierarchical strings, in UTF-8. A forward slash is used to separate different levels of topics. [22]

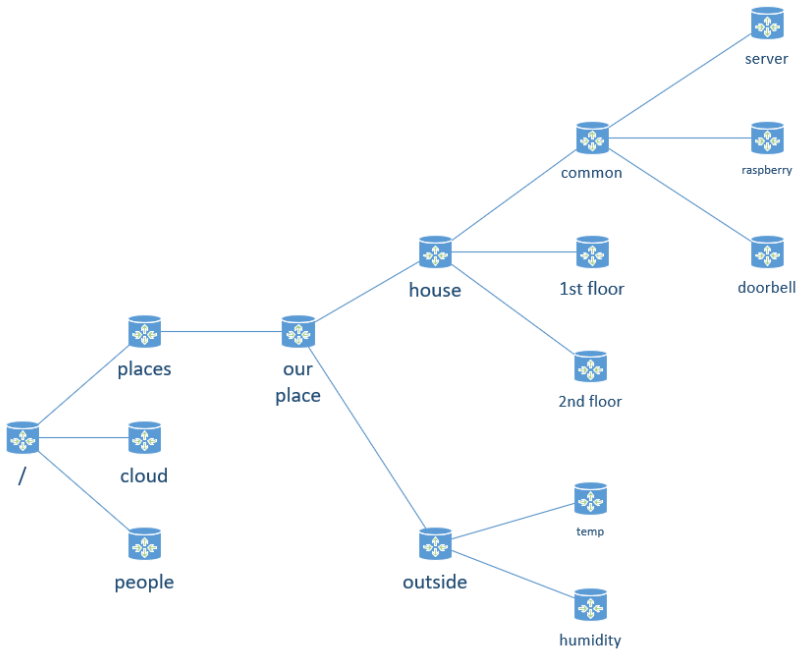

Figure 12. MQTT topic structure.

As seen in Figure 12, there is a root directory with different special areas. The root directory can be named arbitrarily. It is worth noting that '/root' is different from 'root'. Under the main topic there can be multiple subtopics which categorize and specify where a message belongs. When a client subscribes to a topic, different special characters can be used to modify subscribing paths. The most used characters are "+" and "#". Any topic level can be replaced with the "+" character, while the "#" character is used subscribe to every subtopic under a topic. Topic names are not permanent, as they only exist on a server when someone has subscribed to that topic. [22]

5.6 Security

Because MQTT solutions are often deployed in hostile environments where devices or network traffic could be compromised by different attacks, additional security should be implemented.

There are two ways of protecting MQTT messages: TLS/SSL (Transport Layer Security/Secure Sockets Layer) security or payload encryption. TLS/SSL security is done between the client and the server and is called secure-mqtt. With payload encryption the

subscribing device will encrypt the payload, which is then decrypted on the application level. This means that no changes to the MQTT server are necessary for payload encryption, and even if the server is compromised, the payload will remain secure. [22]

MQTT has two ports for traffic. Port 1883 is for unencrypted traffic where messages and passwords are transmitted as plaintext. Port 8883 is used for encrypted traffic where TLS and SSL are used for encryption. This protects the whole message, not only the payload section. SSL/TLS provides data encryption, data integrity and authentication. [22]

5.7 Quality of Service

MQTT defines three levels of Quality of Service (QoS). QoS defines how the server and client ensure that the message is received. Higher levels of QoS are more reliable but involve higher latency and bandwidth requirements due to increased overhead.

- QoS value 0: The Server/Client will deliver the message once, with no confirmation.
- QoS value 1: The Server/Client will deliver the message at least once, with confirmation required
- QoS value 2: The server/Client will deliver the message exactly once by using a four-step handshake.

Both the server and the client have their own QoS level. If the server uses the highest QoS level, then the client can choose which level to use. If the server's QoS is lower than maximum, the client can only use that level of QoS or lower. [25]

6 Project Solutions

The majority of the project design was done based on the IoT architecture research. Apart from LoRa the technological choices were made by the project and discussed with the customer to make sure they fit in their existing infrastructure.

6.1 Project architecture

The IoT system comprises of an optical sensor read by the microcontroller which converts the sensor data to states. The states are sent over LoRa through the LoRa stack to an MQTT server where all the device data is aggregated. The web server receives updates on the device data from the MQTT server and writes them to a database and serves the data on the web server to web clients. The overall architecture is shown in Figure 13.

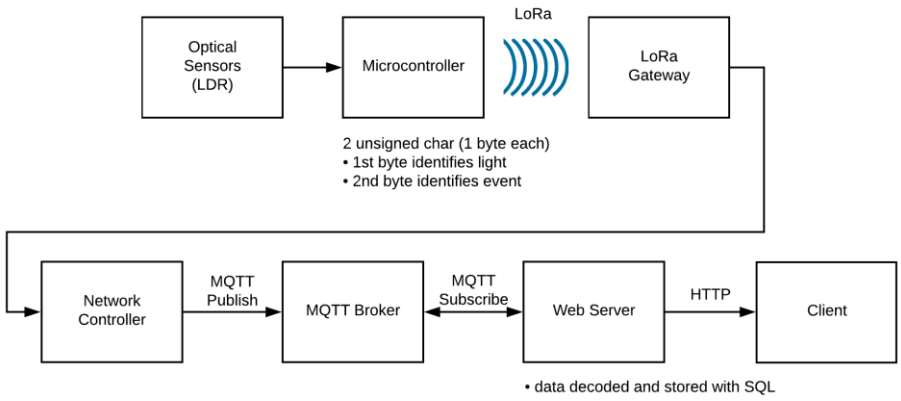

Figure 13. Project technological architecture.

The architecture was designed based on research discussed in chapter 3. The LoRa Gateway and network controller were provided by the customer and were outside the project's control.

6.2 Project technologies

The technological choices were made based on availability and previous familiarity with the technologies. This allowed for the prototype to be created faster without long acquisition periods. Making choices that would be usable after the prototyping phase was another large consideration. All the choices were discussed with the customer and only ones that they had previous experience with or were willing to adopt were chosen.

6.2.1 Device

Light dependent resistor (LDR) circuits were chosen as the optical sensors due to their simplicity, comprising only of a simple LDR and a generic resistor, both of which were readily available to the project. An LDR is a resistor which experiences a decrease in resistance as it is exposed to more light, allowing it to transmit light intensity changes as voltage changes in the circuit. [26]

The STM32 Nucleo pack was recommended by the customer and already at hand. As the pack came with a LoRa board and middleware it was an ideal choice for the prototype development board.

6.2.2 MQTT

MQTT was chosen as the data aggregator as both the project and the customer had previous experience with it. There was also ready support for it in the LoRa stack which made it a good choice even past the prototyping phase.

6.2.3 Python backend

Python 2was chosen as the programming language for the backend program because the project was already familiar with it and because it is designed for quick integrations [27]. A more complex MQTT implementation could have been used instead, but a Python based solution was deemed as the best option to cut down on the amount of technologies that had to be learned.

6.2.4 Web server

CentOS as the operating system, Apache as the Hypertext Transfer Protocol (HTTP) server and MariaDB as the database software were chosen because they are free and the team already had prior experience with all of them [28], [29], [30]. PHP and JavaScript were more difficult to choose because the project had no prior experience with web development, but they were chosen due to their popularity and the amount of readily available material online [31].

7 Sensor Device

The sensor device uses sensors to take measurements and uses those measurements to make decisions on the state of the measuring point. The state information is periodically communicated upstream.

7.1 Design requirements

The sensor should be a generic solution and be easy to install. Installing the sensor device shouldn't obstruct any previous operations. A single sensor device should be able to measure three to five different points simultaneously and distinguish between three different states. The device should connect to the rest of the system using LoRa.

7.2 Hardware

The device was built using an STMicroelectronics 32-bit NUCLEO-L073RZ development board and a fitting SX1272MB2DAS LoRa expansion board that functions as an RF transceiver and LoRa modem. Both are shown in Figure 14. These were included together in the STM32 Nucleo pack [32].

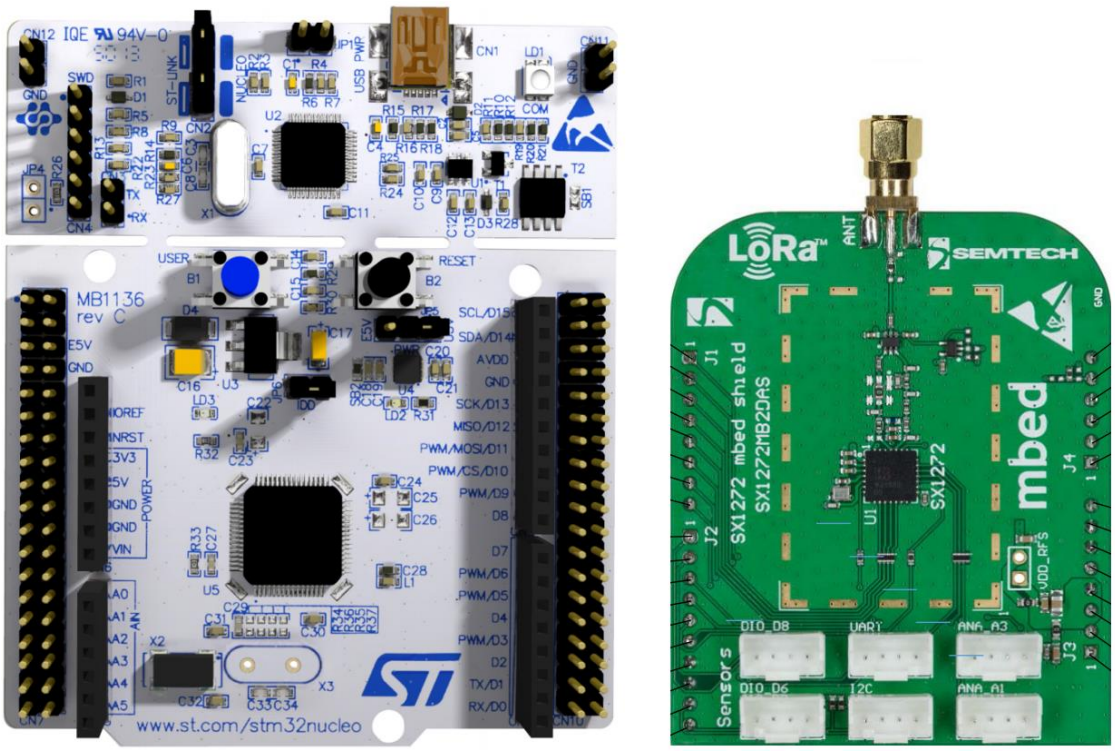

Figure 14. The STMicroelectronics NUCLEO-L073RZ development board on the left and MBED SX1272MB2DAS LoRa expansion board on the right. [33], [32].

The sensors were light dependent resistor circuits. The circuit is show in Figure 15. The LDR used was a 1M ohm Advanced Photonix NSL 4962. The resistor was a generic 1k ohm resistor.

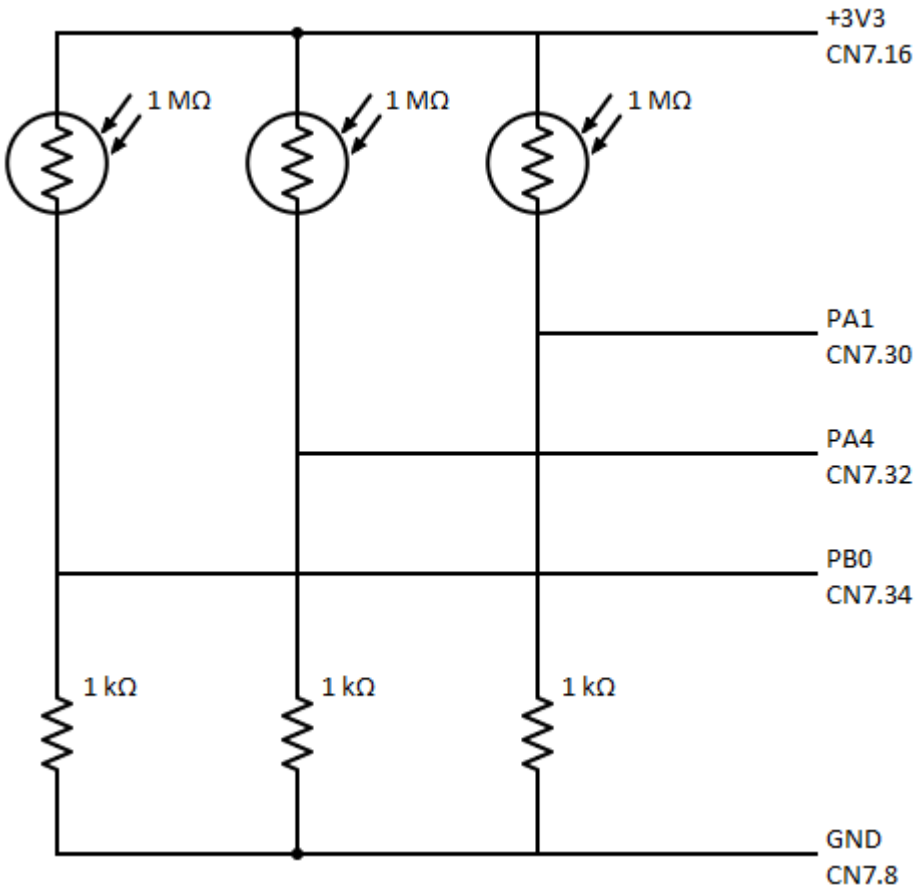

Figure 15. Sensor connection diagram. Two additional sensors can be connected using the CN7.36 and CN7.38 pins.

The sensor circuit was connected to the board's 3,3V output, Analog-to-Digital Converter (ADC) inputs and ground. Unused ADC inputs were connected to the board's Input Output Voltage Reference (IOREF) pin CN7.12 to anchor the reading and prevent false state changes when all five sensors were not used.

7.3 Environmental light

The design requirements and the use scenario gave rise to an additional major challenge. The sensor should accurately detect state changes regardless of environmental lighting. This is further complicated by the nonlinear nature of LDRs. Making decisions by comparing recent readings negates the effect of slow changes in environmental light and allowed the sensor to work in a wide variety of environments. Fast changes in environmental light were filtered by comparing multiple sensor readings to each other before deciding on a state change.

7.4 Device program

The device program was based on the STMicroelectronics STM32Cube LRWAN LoRa end-node code version 1.1.2. Hardware Abstraction Layer (HAL) drivers were used for the board. Semtech middleware was used for the LoRa stack.

7.5 Code Flow

The code runs in a single loop that controls two different parts, the read cycle and the LoRa code. The read cycle takes readings from the sensors and uses them to make decisions on the measurement point state. A single reading is an average of five conversions made by the ADC the sensor is connected to, each 2 ms apart. The LoRa communication is handled by a state machine that is continuously ran from the main loop. The state machine initializes LoRa and joins the network with the configured method. Once the join is completed, or immediately in case of authentication by personalization, the state machine prepares and sends a packet. After transmission the state machine enters a sleep state until it wakes up according to the LoRa duty cycle.

8 Network Infrastructure

The network over which the system communicates comprised mostly of customer-controlled infrastructure, with the public internet as the backbone. As such it was a smaller part of the project.

8.1 LoRa

The LoRa network infrastructure was controlled by the customer. The project provided the customer with parameter and settings information necessary for the system to connect through the LoRa network and did minimal setup to join the project device to the customer network.

8.2 MQTT server

The MQTT server that the system was designed to use was in the customer's infrastructure, but as MQTT is an open standard the server could easily be replaced by any open server. The device packages sent through the LoRa network are published to the MQTT server with a topic structure that can be used by the customer to split the devices in separate sets intended for different monitoring applications. The monitoring application can then be subscribed to the topics that contain the devices intended for its display and only receives the messages of those devices.

9 Application

A CentOS based server was set up for the purpose of handling and displaying data collected by the end devices. This includes handing MQTT messages coming from the MQTT server, storing them in a database, and displaying the relevant data to users through a browser-based frontend. CentOS was installed on a virtual machine provided by Metropolia. For the sake of deployment to production, instructions on how to move the contents of the virtual machine to the customer's infrastructure were given to the customer.

9.1 Backend

A Python program was written to act as an MQTT subscriber and to parse the messages from the MQTT server for the database. The program takes the raw MQTT messages and parses the relevant information, such as the device's LoRa EUI used as the unique identifier and the state information of the measurement points. This information is stored in a MariaDB database, which is discussed in chapter 9.2. If a message is received from a new device that is not in the database yet, it is automatically added. For every change in the database, the program also adds a new line to a log file. The program uses two external libraries, Paho and MySQLdb. Paho is used for handling the MQTT messages, and MySQLdb is necessary for the database connection.

9.2 Database structure

MariaDB was used for hosting the database. The database consists of two tables; “device” and “measurement point”. The structure of the database is shown in Figure 16. A script to create an identical database was provided to the customer.

In the “device” table, the “eui” field is used as a primary key, as the DevEUI of any LoRa end device should always be a unique value [16]. The “x” and “y” fields are used to store the coordinates of the device so that it can be displayed in the correct position on the map. The “heartbeat” value was initially planned to be used for storing the timestamp of the latest heartbeat to ensure that devices remain functional, but this feature was never developed. The “alias” value is used to store a fully customizable, human-readable alias for any device.

```
MariaDB [testlights]> describe device;
+-----------+--------------+------+-----+-------------------+-----------------------------+
| Field     | Type         | Null | Key | Default           | Extra                       |
+-----------+--------------+------+-----+-------------------+-----------------------------+
| eui       | varchar(100) | NO   | PRI | NULL              |                             |
| x         | int(11)      | YES  |     | NULL              |                             |
| y         | int(11)      | YES  |     | NULL              |                             |
| heartbeat | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| alias     | varchar(200) | YES  |     | NULL              |                             |
+-----------+--------------+------+-----+-------------------+-----------------------------+
5 rows in set (0.00 sec)

MariaDB [testlights]> describe measurement point
+------------+--------------+------+-----+-------------------+-----------------------------+
| Field      | Type         | Null | Key | Default           | Extra                       |
+------------+--------------+------+-----+-------------------+-----------------------------+
| id         | int(11)      | NO   | PRI | NULL              | auto_increment              |
| device_eui | varchar(100) | NO   | MUL | NULL              |                             |
| state1     | int(11)      | YES  |     | NULL              |                             |
| state2     | int(11)      | YES  |     | NULL              |                             |
| modified   | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+------------+--------------+------+-----+-------------------+-----------------------------+
```

Figure 16. Database structure. Picture has been modified to hide some technical details.

The “measurement point” table is used for storing data about the measurement points of the device. The “id” value is an automatically generated identifier for each measurement point, and the “device_eui” is used as a foreign key to link each measurement point to the device that they belong to. “state1” and “state2” are used to store state information about every measurement point. The “modified” field is a simple timestamp that is used to see when the status of a measurement point has last changed.

10 Front end

A web-based User Interface (UI) is used to visualize the incoming data. There are two slightly different versions of the UI, one of which is meant for regular users and the other which is meant for administrators. The administration version is functionally the same as the regular one, with a few buttons added for renaming, moving and deleting devices. Apache was used to host the UI, with most of the code being done in JavaScript and PHP. Bootstrap was used for CSS (Cascading Style Sheets) styles.

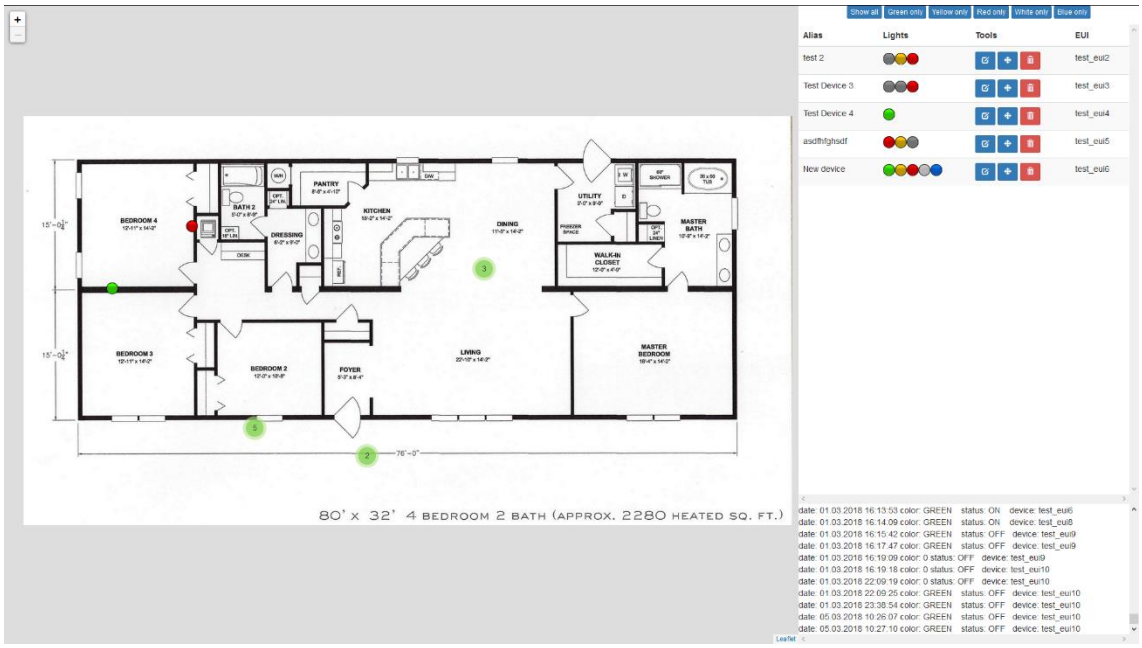

Figure 17. User interface (administration version).

As shown in Figure 17, the user interface consists of 3 major parts:

- the map on the left side,
- the table on the top right,
- the log in the bottom right.

The map shows the location and state of the devices overlaid on any picture. For Figure 17, a simple placeholder floor plan was used for demonstration purposes. Individual colored icons can be clicked to reveal the alias and DevEUI of that device. Additionally, clicking anywhere on the background displays the coordinates of that point. The map

can be zoomed and moved. Several measurement points on a single device will be clumped into a clickable cluster, shown in Figure 18.

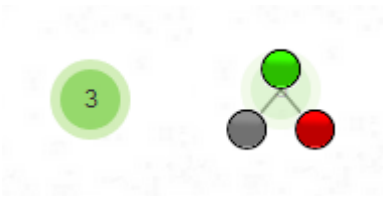

Figure 18. A closed and an open cluster.

The table shows the alias, status and DevEUI of each device in their own row. Additionally, a tools column is visible on the administration version, with buttons to rename, move and delete devices. Furthermore, above the table there are several buttons that can be used to filter the devices that are being displayed.

The log simply displays the end of the log file produced by the back-end Python program.

The user interface automatically scales to always utilize the entirety of the browser's viewport. The map is designed so that, when in default zoom and position, the entire picture will be visible in a maximized browser on a 1920 by 1080 resolution display.

10.1 Map

The map uses the Leaflet JavaScript library for most of the features, with the leaflet-realtime and Leaflet.markercluster add-on libraries. The leaflet-realtime library is used as Leaflet does not support updating the map in real time by default, but with the library it can be updated at any given frequency. Leaflet.markercluster is used for clustering the data points as shown in Figure 18.

A PHP script queries the database for the status of the devices and then generates a JavaScript Object Notation (JSON) format file that the map reads on every leaflet-realtime update interval (3000 ms). Then an appropriately colored marker is placed on the given coordinates. If multiple markers would exist in the same coordinates, they are clustered via Leaflet.markercluster. The distance that markers must have from one another in order to be clustered can be freely configured.

The map was designed to be easily customizable. The floor plan image can be easily changed by copying a correctly named image into the right folder. The bounds of the map will be automatically scaled to fit the new image. The icon size and update interval can also be changed by modifying their respective variables. More detailed instructions on customizing the map were given to the customer.

10.2 Table

For the table, the database is queried for the required data via PHP with the MySQLi extension and a HTML table is generated. The administration buttons in the tools column can be clicked to trigger a SweetAlert function, which will update the database using Ajax.

There are multiple table files to accommodate for the filters and administrator status. The tables are loaded into a Hypertext Markup Language (HTML) div element via jQuery. Clicking on the filter buttons loads the appropriate table in its place. By default, the update interval is 3000 ms. The update interval and icon size can also be changed by changing their respective variables.

10.3 Log

The log is simply the log.txt file generated by the Python program, loaded onto a HTML div element via jQuery on 3000 ms intervals. This interval can be changed. The log is also automatically scrolled to the bottom on a 3000 ms interval by another function.

10.4 Issues

This chapter explains some known issues with the frontend. The issues were unimpactful and could not be solved due to time limitations, but they were described in the documentation provided to the company.

10.4.1 Map

The leaflet-realtime and Leaflet.markercluster libraries are not designed to work together, so a workaround had to be used. The cluster layer is recreated on each leaflet-realtime update, which causes opened clusters to close within the leaflet-realtime update interval time (set to 3000 ms by default).

If using very high-resolution images as the background map, performance will be poor on low-end machines. The demonstration background image file (2949 by 1568 pixels) works well on all tested machines, but another larger map image (11149 by 8045 pixels) would cause the map to occasionally freeze while moving it or zooming on less powerful machines. A possible solution would be to use a tiled map layer with zoom layers [34], but this requires external software to create the tiles.

For unknown reasons, the map will occasionally not load on the first page load, but refreshing the page fixes the issue.

10.4.2 Table

The default, unfiltered table generates a large number of Structured Query Language (SQL) queries as the status of the measurement points is queried separately for each device.

If a device has a very long alias, a single table row might continue onto a second text row to make everything fit.

11 Conclusions

The goal of the project was to create an IoT solution within the customer company's specifications to streamline the company's production processes. To meet this goal, a sensor device and an application backend were developed along with the means of

communication between them. The company was prepared to immediately begin using the system in production environments.

The project was very challenging due to its large scope, especially because many different technologies had to be learned from scratch. However, this led to a lot of knowledge being gained in technologies like LoRa, C programming, JavaScript, PHP and Python among others.

11.1 Future Improvements

Despite the project being a success overall, there was still some room left for improvement and some potentially useful functions were left unexplored. The most interesting of these was converting the device to be completely battery-powered. The code on the device is theoretically already quite power efficient; thus, this could have been fairly simple to implement. However, the company did not request this function as Alternating Current (AC) power was readily available to them, so no work was put into it.

A heartbeat function could have been useful in some circumstances. Essentially, the device would periodically send a message to broadcast that it is still online. If no such message is received within some time, an alert could be triggered to indicate that the device is not fully functional. In the current implementation this is not necessary, as a message containing the state of every measurement point is sent periodically, regardless of whether anything has changed or not.

Some improvements to the web server backend and frontend could have been made. For example, PHP and MariaDB were chosen mostly due to convenience, while some alternatives may have been better suited for this purpose. The map's performance could also have been improved by splitting the background map image into tile layers. Unfortunately, no free software to perform that function could be found.

References

[1] ITU Telecommunication Standardization Sector, "Series Y: Global Information Infrastructure, Internet Protocol Aspects and next Generation Networks - Frameworks and functional architecture models - Overview of the Internet of things," ITU-T, 2012.

[2] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, Wei Zhao > J. Lin et al., "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal,* vol. PP, no. 9, pp. 1-1, 2017.

[3] X. Jia, Q. Feng and T. Fan, "RFID technology and its applications in Internet of Things (IoT)," in *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, Yichang, China, 2012.

[4] R. Khan, S. U. Khan and R. Zaheer, "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges," in *2012 10th International Conference on Frontiers of Information Technology*, Islamabad, India, 2012.

[5] L. Da Xy, W. He and Sl Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics,* vol. 10, no. 4, pp. 2233-2243, 2014.

[6] J. Guth, U. Breitenbücher and M. Falkenthal, "Comparison of IoT platform architectures: A field study based on a reference architecture," in *Cloudification of the Internet of Things (CIoT)*, Paris, France, 2016.

[7] R. Kanan, "IoT devices: The quest for energy security," in *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Abu Dhabi, United Arab Emirates, 2016.

[8] C. Hray, R. Ayre and K. Hinton, "Power consumption of IoT access network technologies," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, London, UK, 2015.

[9] T. Yokotani and Y. Sasaki, "Comparison with HTTP and MQTT on required network resources for IoT," in *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, Bandung, Indonesia, 2016.

[10] P. Bellavista and A. Zanni, "Towards better scalability for IoT-cloud interactions via combined exploitation of MQTT and CoAP," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Bologna, Italy, 2016.

[11] T. Clausen, M. W. Townsley, J. Yi and A. Augustin, "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things," *Sensors,* no. Enabling the Move from Wireless Sensor Networks to Internet of Things and Cyber-Physical Systems, 2016.

[12] K. Mekki, E. Bajic, F. Chaxel and M. Fernand, "A comparative study of LPWAN technologies for large-scale IoT deployment," *ICT Express,* vol. 5, no. 1, pp. 1-7, 2019.

[13] LoRa Alliance, "A technical overview of LoRa® and LoRaWAN™," November 2015. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf. [Accessed 15 April 2019].

[14] Semtech Corporation, "AN1200.22 LoRa™ Modulation Basics, Revision 2," 2015.

[15] S. Corporation, "SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver, Rev. 5," 2016.

[16] LoRa Alliance Technical Committee, "LoRaWAN 1.1 Specification," LoRa Alliance, Inc, 2017.

[17] N. Abramson, "THE ALOHA SYSTEM—Another alternative for computer communications," University of Hawaii, Honolulu, Hawaii, 1970.

[18] I. Gitman, "On the Capacity of Slotted ALOHA Networks and Some Design Problems," *IEEE Transactions on Communications,* vol. 23, no. 3, pp. 305-317, 1975.

[19] R. Miller, "LoRa Security, Building a secure LoRa solution," MWR Labs whitepaper, 2016.

[20] Gemalto, Actility and Semtech, "LoRaWAN SECURITY, FULL END-TO-END ENCRYPTION FOR IoT APPLICATION PROVIDERS," February 2017. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/lora_alliance_security_whitepaper.pdf. [Accessed 21 April 2019].

[21] OASIS, "ISO/IEC 20922:2016, Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1," 2016.

[22] OASIS Message Queuing Telemetry Transport (MQTT) TC, "MQTT Version 3.1.1 Plus Errata 01," 2015.

[23] mqtt.org, [Online]. Available: http://mqtt.org/faq. [Accessed 22 4 2019].

[24] mqtt.org, [Online]. Available: http://mqtt.org/documentation. [Accessed 22 4 2019].

[25] International Business Machines Corporation (IBM), Eurotech, "MQTT V3.1 Protocol Specification," 2010. [Online]. Available: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html. [Accessed 16 4 2016].

[26] J. Fraden, "Light Detectors," in *Handbook of Modern Sensors*, Springer, 2016, pp. 525-567.

[27] Python Software Foundation, "python," 2019. [Online]. Available: http://www.python.org/. [Accessed 22 4 2019].

[28] The CentOS Project, "CentOS Project," 2019. [Online]. Available: http://www.centos.org. [Accessed 2019 4 22].

[29] The Apache Software Foundation, "Apache HTTP server project," 2019. [Online]. Available: http://httpd.apache.org/. [Accessed 22 4 2019].

[30] MariaDB Foundation, "MariaDB.org - Supporting continuity and open collaboration," 2019. [Online]. Available: http://mariadb.org/. [Accessed 22 4 2019].

[31] TIOBE Software BV, "TIOBE Index," 2019. [Online]. Available: http://www.tiobe.com/tiobe-index/. [Accessed 22 4 2019].

[32] STMicroelectronics, "UM2085 User manual, Ultra-low-power STM32 and LoRa Nucleo pack with Nucleo-L073RZ board and I-NUCLEO-SX1272D RF expansion board," [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/user_manual/group0/d3/69/4a/7c/aa/4d/42/16/DM00307583/files/DM00307583.pdf/jcr:content/translations/en.DM00307583.pdf. [Accessed 14 April 2019].

[33] STMicroelectronics, "UM1724 User manual, STM32 Nucleo-64 boards (MB1136)," [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf. [Accessed 14 April 2019].

[34] V. Agafonkin, "Leaflet," 2017. [Online]. Available: http://leafletjs.com/examples/zoom-levels/. [Accessed 22 4 2019].