Danmarks Tekniske Universitet

Diplomingeniørprojekt

Orchestration of Bluetooth mesh networks for industrial IOT applications

Pil Skov Eghoff (s164922)

December 13, 2019

Abstract

In smart manufacturing, mesh networks offer a way to connect hundreds or thousands of low power devices without the need for complex wiring. This allows the creating of sensor networks that enables constant monitoring of factories and industrial equipment. One technology for industrial mesh networks is Bluetooth mesh.

Bluetooth mesh use the traditional Bluetooth low energy hardware, and is already in wide use in home automation, but several factors have kept it from being used in Industrial solutions. Remote setup and monitoring of networks is essential in industrial networks, yet no commercially available solution exists for Bluetooth mesh. This projects examines the specific needs for an Industrial solution, and creates a gateway that can be remotely controlled and create and monitor Bluetooth mesh networks.

The security of the gateway is evaluated using the industrial security standard IEC 62443, and was found to live up to the requirements laid out.

In addition to the gateway, a rudimentary server and several nodes were developed to test and demonstrate the setup. The test and demonstration proved the functionality of the gateway.

This gateway can act as the foundation of future Bluetooth mesh projects, and can help build confidence in Bluetooth mesh in Industrial Internet of things.

Contents

1 Introduction

As industry evolves, technology evolves to meet its needs. Industry 4.0 presents a whole new world of possibilities, yet it also creates a whole new set of requirements and challenges. Industry 4.0 is the next step in the evolution of industry, and is distinguished by smart manufacturing and the use of Internet of things (IoT).

Move Innovation is a company that offers custom Industrial IoT (IIoT) solutions. They have been on the hunt for a technology that offers cheap, low power mesh communication. They want to create large industrial sensor networks for collecting data on manufacturing to support other IIoT solutions.

There exist plenty of mesh solutions, but none offer the combination of large ecosystem, low power, low cost that Bluetooth mesh does.

For that reason, Bluetooth mesh has been looked at for several projects within Move Innovation, but some unanswered questions about Bluetooth mesh has prevented Move Innovation from actually including it in a product yet.

The main issue with Bluetooth mesh is orchestration. How do you control Bluetooth mesh networks remotely?

This project will examine and answer this question, and generate a product that can serve as the base for future Bluetooth mesh projects.

1.1 Problem definition.

Bluetooth mesh is a technology that enables many-to-many communication, and has been targeted both at home automation and IIoT.

In the field of IIoT, Bluetooth mesh offers a solution for highly scalable networks that reach further than any individual transmitter/receiver pair.

A Bluetooth mesh network require three essential steps to ensure that the network is usable.

- The network needs to be provisioned i.e. all devices have to be added to the network
- The devices in the network needs to be configured.
- The network needs to be monitored to ensure that everything is in order and no devices are lost to the network.

In home automation, the first two steps are handled via an app, while the third is up to the user, but no solution has found wide adoption in IIoT, and this has limited adoption of Bluetooth mesh in an industrial setting.

To solve this, this project will develop a Bluetooth mesh IIoT gateway. This gateway will enable provisioning, configuring and monitoring of Bluetooth mesh networks via a cloud connection. A server for controlling the gateway as well as nodes to test the capabilities will also be developed to demonstrate the viability of Bluetooth mesh as an IIoT technology.

The requirements for the project are as follows.

- Enable remote provisioning of Bluetooth mesh networks
- Enable remote configuration of Bluetooth mesh networks
- Monitor a Bluetooth mesh network, and report missing devices to a central server
- Demonstrate the above points using at least 4 Bluetooth mesh devices
- The solution should live up to the industrial security standard IEC 62443, to a reasonable degree

1.2 Use cases of Bluetooth mesh in IIoT

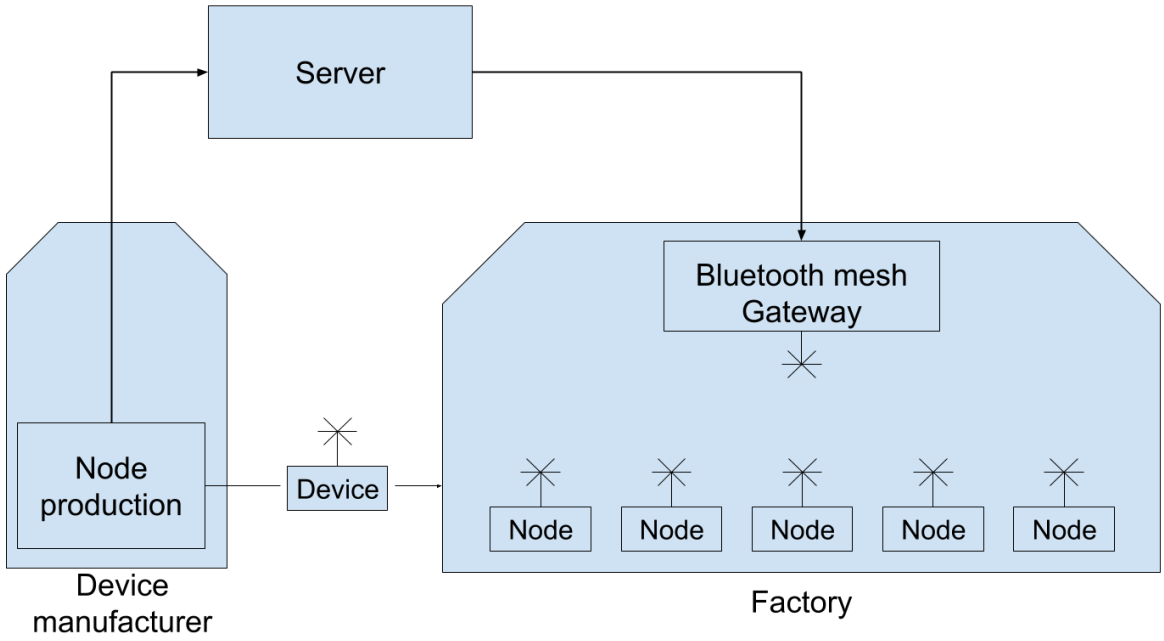

Figure 1: Illustration of the use case.

In this section a use-cases of Bluetooth mesh and a Bluetooth mesh gateway in an industrial setting will be presented.

A factory uses Bluetooth mesh devices to gather data on industrial equipment. On each piece of equipment a device is placed that measures temperature, vibrations etc. This data is made available to the network, and other nodes gather this data.

The device manufacturer produces the Bluetooth mesh nodes that are used in the factory. As soon as one device is produced, its serial number and a secret key is registered in a central database. These together will be used to authenticate the device at a later point. The device is then transported to the factory.

The factory has a Bluetooth Mesh gateway, and this recognizes that a new Bluetooth mesh device is present in the factory, and checks the central database to confirm the identity of the device.

Given that the device passes all checks, the device is provisioned, thereby being added to the mesh network and configured according to the configuration on the central database.

When the device has been provisioned, it becomes a node in the network.

This configuration can be changed at a later time by a network administrator, as the setup in the factory changes.

1.3 Existing solutions

Before developing the product, an investigation into existing solutions was made. A search was made using the phrase "bluetooth mesh gateway", and all relevant pages were evaluated. The following existing solutions were found.

DPTechnics Bluetooth mesh gateway[1]: There is very little information on this gateway. It is not available until Q2 2020. They advertise that it integrates with their existent IoT platform, BlueCherry.

Ble-mesh.com [2]: This website advertises a Bluetooth LE mesh technology, but it is unclear whether or not it is a proprietary solution or based on the SIG standard. The website it very sparse with information. They have a section dedicated to case histories but it contains only generic pictures.

Open Source Wi-Fi to Bluetooth 5 IoT Gateway[3]: This is an open source gateway from Fanstel. At the time of writing, the firmware is not fully developed, and they state that it is only ready for home networks. It does not seem to support provisioning and configuration of networks, as it is focused on transferring data from the mesh network to a remote server.

Telink semiconductor[4]: This solution seems to focus on adding Bluetooth Mesh capabilities to already existing hardware. It does mention a gateway that can be used for remote access. They write: "Gateway is implemented by adding a simple Telink BLE module to existing Wifi/IP routers". No ordering information or availability is present on the website.

Bluerange.io [5]: This supports both an older proprietary mesh technology as well as the SIG standard. The gateway offers "remote access", and they mention that it can be used to manage software updates, but no information about provisioning and configuration is present. They do mention that they offer an app for setting up networks, and that their proprietary solution offers "Auto-meshing", and it seems to indicate that no setup is needed.

In **conclusion**, it is unclear if any solutions offer remote provisioning, configuration and monitoring of Bluetooth mesh networks according to the SIG standard. Most solutions are either unavailable or use a proprietary mesh technology. Few solutions seem to suitable for industrial use. As no solutions seem to cover the use case presented earlier, development of a new solution is necessary in order to solve the problem.

1.4 Report structure

This section will give brief overview of the structure of the report.

The report goes through each step of the development phase. In reality agile development was used, and therefore development was made in cycles, and cannot be seen as linear, but in this report, each sections presents the collective work done in that phase throughout the project.

The phases are:

- **Planning**, where the plan for the project is laid out
- **Analysis**, where the project and the problems it presents are analyzed to get an understanding that can inform further decisions
- **Design**, where the design is developed
- **Implementation**, where the solution is implemented based on the design
- **Testing**, where the implementation is tested
- **Evaluation**, where the results and solutions are evaluated
- **Further works**, where it is determined what further work there is to be done

2 Project plan

During the initial planning of the project, a gannt-chart was created, based on the initial estimates of the different parts of the project and how long they would take. This was to created to better structure the work and such that the progress could be monitored.

The topics in this chart were very surface level, as much of the research was yet to be conducted.

This initial estimation as well as how much time was actual spent can be seen in Appendix A. In the end, a lot less time was spent on the actual implementation of most features, expect for cloud communication, which took 13 hours longer than expected. This was caused by the fact that cloud integration also acted as a general system integration where all parts had to play together.

During the project several aspects of the agile development methodology was used. During the start of the project, the focus was to get a working minimal viable product. During each week, it was decided which new feature was to be added, based on the final goal.

This meant that there throughout the project was a working product, though with a limited feature set.

A circular development process was used where all the steps of planning, analysis, design, implementation, testing, evaluation were used to implement each new feature.

During the project there were weekly meetings with the company supervisor, to discuss progress and issues that arose.

These meetings were important to keep everything on the right track as the project was developed by a single person.

3 Analysis

This section will analyze the problem further and provide an overview of the technologies involved with the project.

To get an overview of what topics are needed to understand in order to complete the project, the image in figure 2 has been created. This illustrates the different components and which components communicate.

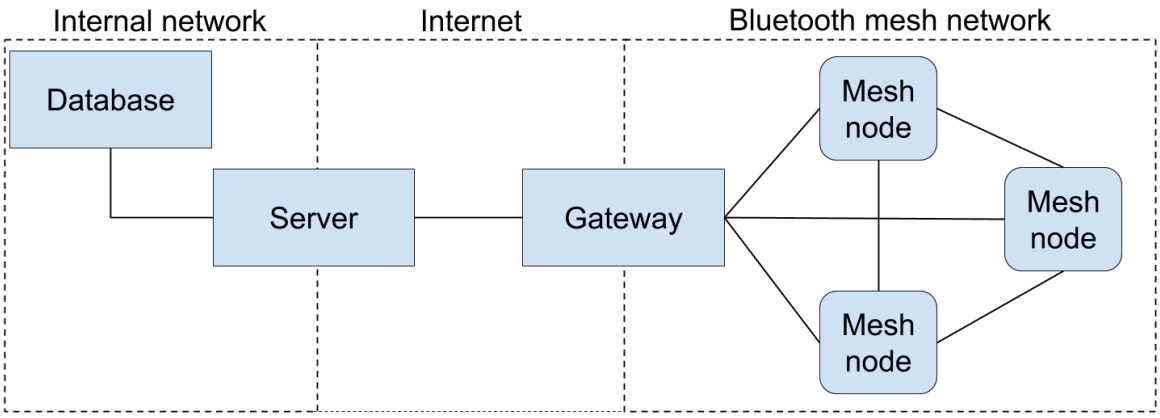

Figure 2: An overview of the components in the project.

It illustrates how the nodes communicate entirely within the mesh network, while the gateway acts as a bridge between the mesh network and the internet. The server connects both to the internet and a private internal network. This illustration provides the guide for choosing topics for the initial research.

The gateway will be the primary focus for the project, while the server and nodes are secondary and only developed in the extend that is needed to prove that the gateway fulfills the requirements.

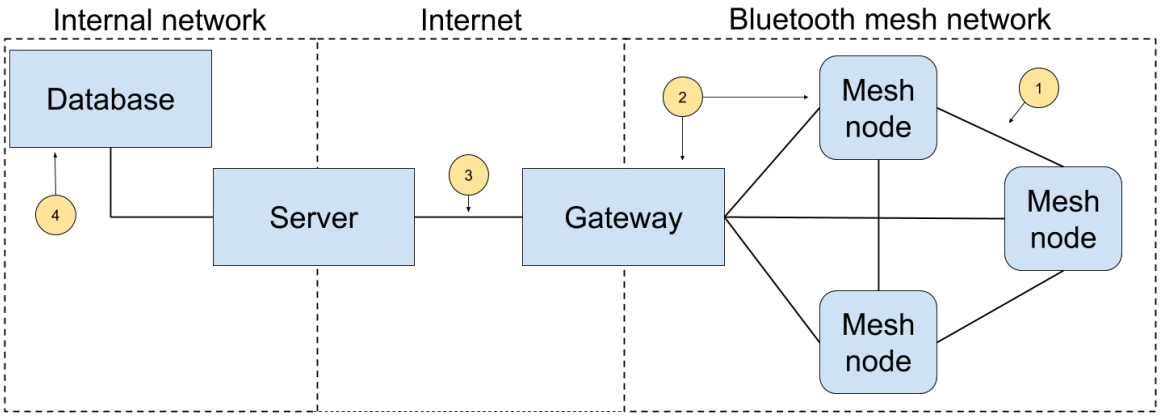

Figure 3: An overview of the components in the project and which parts will be researched initially.

Figure 3 shows the same overview, but also highlight the four topics that will be examined in the following sections, namely:

1. The Bluetooth mesh networking protocol e.g. how it functions, which parts are needed for the project, which requirements the use places on the hardware/software platform.
2. The hardware and software platform for the development of the gateway and nodes.
3. The protocol for communicating between the server and gateway.
4. The database to store the information on the network.

In addition to this security in industrial automation will also be examined.

3.1 Bluetooth mesh.

This section will be a brief introduction to Bluetooth mesh.

Bluetooth mesh is a mesh networking protocol introduced in 2017 [6] by the Bluetooth special interest group (SIG), who oversee the development of Bluetooth technology. Using previous Bluetooth technologies, both 1-to-1 and 1-to-many communication was possible but Bluetooth mesh opened the door for many-to-many communication. The difference between the three is illustrated in figure 4

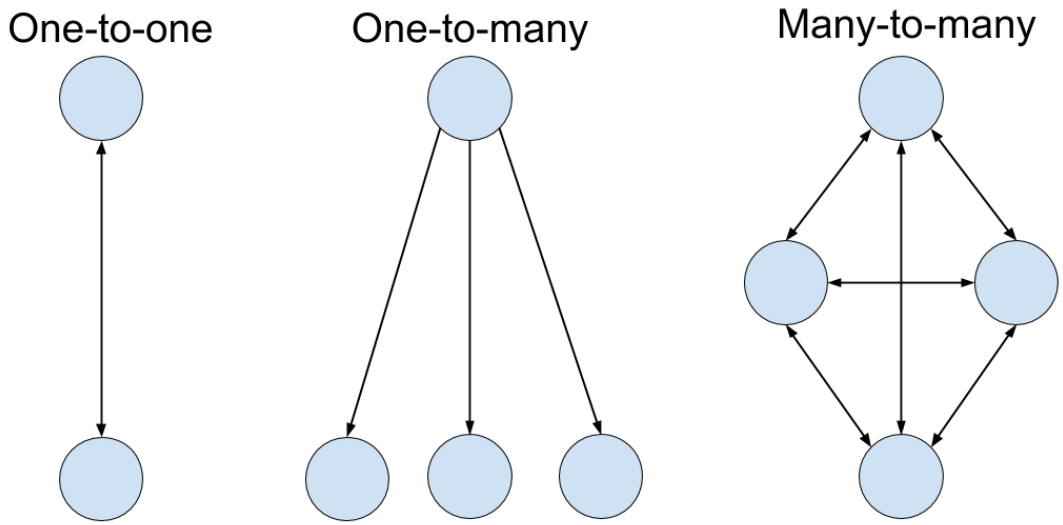

Figure 4: Examples of different networks types possible using bluetooth technology.

Bluetooth mesh was targeted at both home automation and industrial IoT[6]. Bluetooth mesh works using existing Bluetooth hardware, meaning that existing chips can support Bluetooth mesh without any hardware modifications. This means that even though Bluetooth mesh is a new technology, there is already an extensive hardware ecosystem available.

3.1.1 Routing

Bluetooth mesh uses managed flooding that works as follows.

Every message send is relayed by a subset of all nodes that receives it. Every message has a TTL (time to live) counter that makes sure that every message decays over time. The subset of nodes that relay messages are called relay nodes, and which nodes acts as relays is configurable and can affect the network topology and performance.

This simple mechanism means that no routing tables has to be configured for the network and because of this no node needs knowledge of other nodes. This means that nodes can, for the most part, go in and out of the network without consequence.

The relay mechanism allows the network to stretch much further than any single Bluetooth connection is able to reach.

3.1.2 Communication model

Bluetooth mesh works using a publisher/subscriber model. A client publishes to an address, and a server subscribes to one or more addresses. Addresses can represent a specific element, a group of elements or the entire network. [7, Sec. 2.3.8] A element is a group of models, and a node must contain at least one element, but can contain as many as it likes. Figure 5 illustrates how a network can be set up using this model. Addresses have been allocated for different rooms in a house, and light-bulbs and switches then subscribes and publish to these addresses respectively.

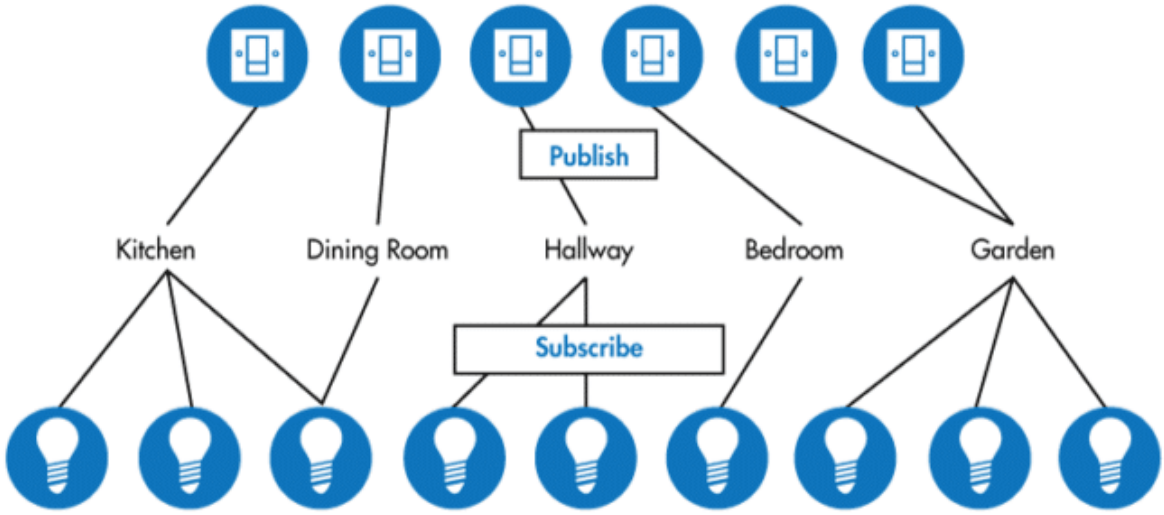

Figure 5: An illustration of mesh connected light bulb subscribing to addresses representing different rooms in a house, and light switches publishing messages to these addresses. Source: Bluetooth SIG

Models define how messages are structured. Models are split into two categories, servers and clients. Servers define a state, such as a light bulb, while clients modify or read states, such as light switches. [7, Sec. 2.3.6]

The foundation models define the necessary models in order for a network to operate. They are a configuration server and client and health server and client. Both the health and configuration servers are mandatory for all nodes in a network, while the corresponding clients are mainly used by network administrators. [7, Sec. 4]

The standard also defines a set of generic models, which includes different abstract server/client combinations, that can be used to represent a broad range of applications. This includes an on/off model for binary states, such as lights or fans, a level model for representing numbers between 0 and 255 which can used for light intensity etc. There are 14 generic models and these are meant to cover most use-cases. [8]

3.1.3 Provisioning and security

A mesh network is defined by a network key. This key is used to encrypt all messages in a network, and this ensures that all messages in the network cannot be read by outsiders. To each element in a node is bound an application key, this is used to encrypt messages at an application level, meaning that nodes that are run different applications and should

not communicate can exist on the same network and still relay the messages of each other.

Lastly, there are device keys. These are exclusive to a device and a configuration client, and exists to ensure that other devices can't change the configuration of that device. It is transferred from the device to the configuration client during provisioning. [7, Sec. 2.3.9]

Provisioning is the procedure, by which new nodes are added to the network or the network itself is created[7, Sec. 5] and works as follows

- An unprovisioned device starts by advertising it's existence.
- An provisioner then invites it and a shared secret key is generated used Elliptic-curve Diffie–Hellman. This algorithm is resistant to Man in the middle attacks.
- Then an authentication method is decided by the provisioner, based on the capabilities of the device.
- If the device is successfully authenticated, it is provided the network keys and configuration can begin.

The authentication methods can be split into two categories dynamic and static. Dynamic is based on a random number, and one form is that the device displays a random number, which is typed into the provisioner. Using the a hash of the random number, the two devices can confirm that they are who they claim to be.

The static method uses a static (or non-random) key, that can be programmed into both devices during production, or calculated at a later point. Because of the lack of user interaction needed with the static method, it is both less secure and more convenient. [7, Sec. 5]

In Bluetooth mesh, a device is anything that supports Bluetooth mesh, while a node is a device that is part of the network.

3.1.3.1 Trash-can attack

A common type of attack is called the trash-can attack. In this scenario a device is provisioned and stores the keys in flash. The device is then later decommissioned and thrown in the trash. An attackers picks the device out of the trash and extracts the keys, giving the attacker the keys needed to decrypt network packages.

One way to mitigate this attack is to change all keys whenever a device is decommissioned, or on a regular basis. This is called a key refresh, and the new keys are send encrypted using the device keys such that the message is secure transported from the configuration client and each node.

3.2 Choosing a platform

This section will look at different options for hardware/software platforms for the project. The chosen platform will be used both for the gateway and the network nodes to simplify development.

3.2.1 Requirements for the platform

1. Bluetooth mesh node support

It needs to support the basic Bluetooth mesh functionality, including all the required foundational models.

2. Bluetooth mesh provisioner support

To be usable for the gateway it needs to support provisioning mode.

3. Bluetooth mesh configuration client support

To be usable for the gateway it needs to support configuration client, as it enables configuration of other nodes.

4. OS support

The platform needs to support an operating system, e.g. an RTOS, as it enables faster development when multiple complex communication protocols are in play at the same time.

5. Support for cloud connection (WiFi or Ethernet)

To enable cloud connectivity the platform needs to support WiFi, Ethernet or a third option for connecting to the internet. The platform will also need support for technologies such as TCP/IP.

3.2.2 The candidates

In this section the candidates that were up for consideration are described. These were chosen because of previous personal experience or based on recommendations from engineers at the company.

	NRF52 & NRF5 SDK	NRF52 & Zephyr	ESP32 & ESP-IDF	Raspberry Pi
Node	✔	✔	✔	×
Provisioner	✔	×	✔	✔
Configuration client	✔	✔	✔	✔
OS	✔	✔	✔	✔
Cloud connection	×	✔	✔	✔

Table 1: A table showing the candidates against the parameters

Nordic NRF52 and NRF5 SDK for Mesh: NRF52 and the accompanying SDK from Nordic Semiconductor was one the first platforms to support Bluetooth mesh and has support for the complete Bluetooth mesh spec, meaning requirement 1, 2 and 3 are fulfilled. It has some rudimentary scheduling support through the soft-device library [9], but still lacks many features like task priorities etc. None of the supported development boards include support for WiFi or Ethernet.

Nordic NRF52 and Zephyr: Zephyr is a IoT focused RTOS supported by the Linux foundation, Intel, NXP and Nordic semiconductor [10]. It has wide support for Bluetooth mesh, but an early decision in the project was to not support provisioning [11], though work towards support for this feature has begun[12]. It support requirement 1,3 and 4, but lacks support for both provisioning and internet connectivity.

ESP32 and ESP-IDF by Espressif Systems: ESP-IDFs support for Bluetooth mesh is still relatively new, and is part of their version 4.0 beta release. It is nonetheless still certified by the Bluetooth Special interest group [13]. The Bluetooth mesh stack in ESP-IDF is based on the Zephyr stack, but they have added support for provisioning. In addition to this both FreeRTOS and WiFi/Ethernet is supported, meaning they support all 5 requirements.

Raspberry pi and Linux: Raspberry Pi 3 b+ has both a Bluetooth model, Wi-Fi and Ethernet. BlueZ is the Bluetooth stack for Linux and it supports provisioning and configuration of Bluetooth mesh networks, but does not support acting as a node[14].

Table 1 shows and overview of the above information, and from that can be concluded that only the ESP32 using ESP-IDF fulfills all the requirements.

3.3 ESP32

This section will provide an overview of the ESP32 family of chips.

The module used in this project is the ESP32-WROOM-32D, using the ESP32 chipset, developed by Espressif. Two different development boards will be used during this project. The ESP32-Gateway from Olimex [15] will be used for the gateway, as it contains an Ethernet connection, while the smaller and more low cost ESP32 Devkit C V4 from Espressif will be used for the nodes.

The ESP32 is designed as a complete integrated solution for IoT products providing both Wifi, Bluetooth Classic and Bluetooth Low Energy support, along with support for a wide variety of peripherals and low-power modes. [16, Sec. 1.1.2]

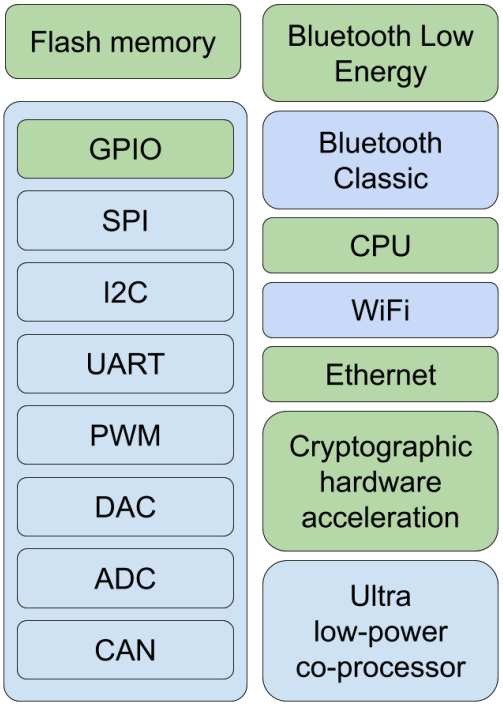

Figure 6: A block diagram of some of the functional block in the ESP32 modules. Based on figure 1 from the ESP32 datasheet.

It contains an Xtensa single or dual-core 32-bit processor, depending on the model. This project only requires a single core. It supports several important security features such as flash encryption and secure boot and has hardware acceleration for many cryptography functions [16, Sec. 1.4.4] In figure 6 there is a block diagram with a subset of the functional blocks in the ESP32 family of modules. The functional blocks that will be used in this project have been highlighted using green, while the blue blocks will be unused. Espressif supplies a development environment called Espressif IoT Development Framework (ESP-IDF). This SDK provides support for a wide variety of protocols and features. To take advantage of both processors as well as all the many options for

connectivity, the ESP-IDF is based on a custom version FreeRTOS.

3.3.1 Secure boot and flash encryption

Secure boot and flash encryption are two separate security features that are often used together [17]. Secure boot is a boot-loader that guarantees that only code signed by a secret key will be run on the device, while flash encryption ensures that an attacker cannot read the flash on the device. The latter is useful when keys are stored in the flash [17]. Both technologies use internal eFuses to store the keys that are used to verify/decrypt data [17], [18]. The eFuses are one-time settable bits, that cannot only be read from inside the device, and are not accessible to an outside attacker, without first running their code on the device [19, p. 503]. Both features slow down development, as it requires signing and encryption of all firmware, and will first be useful in production. It will therefore not be implemented in this project.

3.3.1.1 Recent attacks against flash encryption Recently an exploit against flash encryption on all ESP32 modules was revealed [20]. This attack uses fault injection to read out the secret key that is stored in the eFuses of the ESP32. The attacker has to inject specific signals into the power and clock signal of the module, this sometimes a glitch where the device skips instructions during a reboot and causes the device to spit out the secret keys.

Espressif has released a new hardware revision that fixes the issue, and this new module is recommended for further development.

3.4 FreeRTOS

FreeRTOS is one of the most used embedded operating systems on the market [21]. It provides deterministic scheduling in addition to a wide variety of built in features that makes the development of complex embedded software possible, such as semaphores, queues etc, while still staying small enough to be used on micro-controllers such as the ESP32. All of this could be considered necessary for developing embedded projects that need to communicate using multiple protocols such as Bluetooth and TCP/IP. To do this, FreeRTOS uses tasks. These are independent units of execution, meaning they execute different code and have different stacks. A scheduler is used to switch between the different task based on a tasks priority, which is typically assigned when the task is created.

One use of tasks is handling a Bluetooth communication:

When initializing the Bluetooth subsystem, a task is created to handle communication via the Bluetooth hardware. A priority is selected, such that no Bluetooth packets are missed, without taking up all the execution time of the chip. Different mechanism can then be used to communicate between the Bluetooth task and the application level tasks. Some considerations must be taken when developing using an RTOS like FreeRTOS to avoid race-conditions, dead-locks and stack overflow.

3.4.1 FreeRTOS Queues

The main mechanism that is used in this project for inter-task communication is Queues. These present a high-level interface for passing messages between tasks. When created, they are given a size, based on the size of the messages they should contain and how many instances they need to be able to hold, and a buffer is then allocated. A task can then copy an object into the queue, and another can receive the same object from the queue. If no object has been put into the queue, the receiver can choose to wait until a object becomes available. In many cases several tasks put objects into the queue, and the receiver(s) will receive the objects in order. If the queue is full, a tasks that is trying to place a object into the queue can either choose to wait until a spot is free, or continue execution.

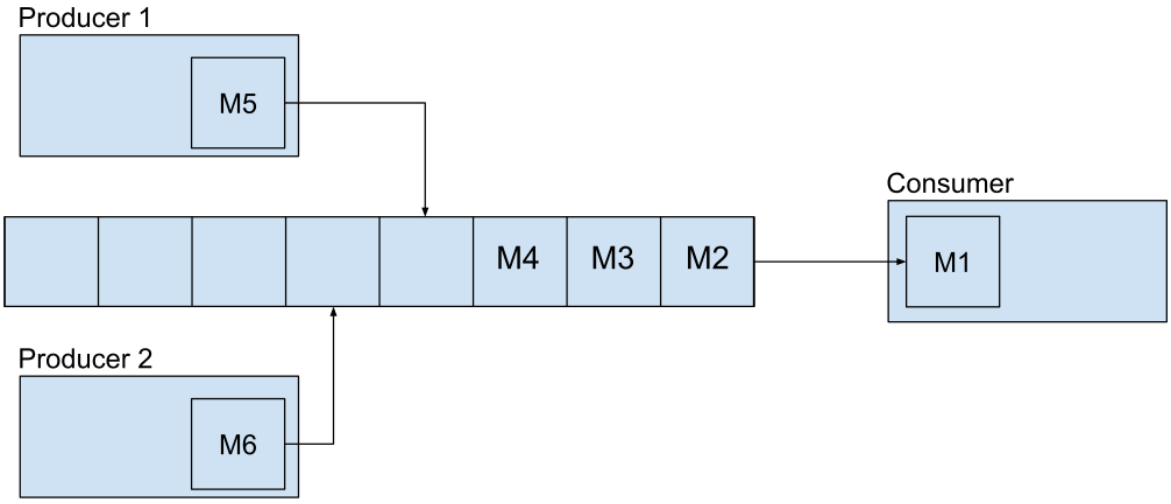

Figure 7: Queues takes messages that are produced asynchronously and serializes them such that they can be consumed synchronously.

Queues are used to serialize a operations, that are generated asynchronously, which is illustrated in figure 7. They decouple the producer and consumer of these operations with the drawback that producer have no way of knowing how long an operation will take to complete until it is done. Queues are simple abstractions on top of circular

buffers and provide the same speed and in many cases a simpler mental model. Queues provide a guarantee that the objects in the buffer is of a certain size and that nothing is overwritten before is has been consumed, which circular buffers typically do not.

3.5 MQTT

MQTT is an IoT communication protocol that will be used in this project and this section will give a brief overview of the technology.

MQTT used a publish/subscribe model, using different topics. A device can publish messages to a topic, and any device that subscribes to this topic will receive the message, which is shown in figure 8.

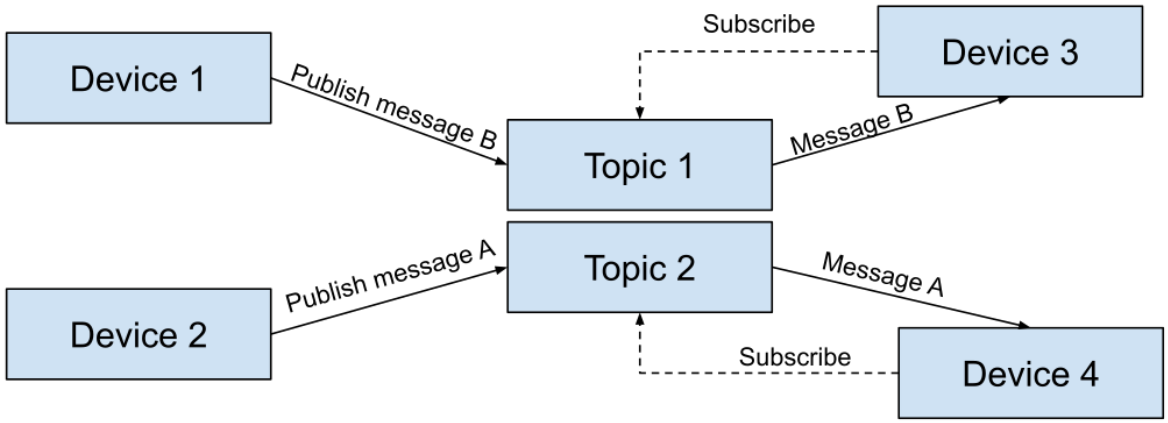

Figure 8: An MQTT broker with two topics and 4 clients.

A broker is used to handle all communication. It stores all messages and subscriptions, and when a new message is revived, it propagates it to all relevant devices. Topics are strings, generally with a schema that resembles paths to files or folders. An example could be "/house1/living_room/light_bulb2". This represents a certain light bulb in the living room of a house. For some applications, it makes sense to keep the topics closely related to physical objects, as in the previous example, while in other more abstract topics are needed. MQTT is used in this project, as it allows messages to be sent both from and to small low-power IoT devices, as most the work is handled by the broker.

To ensure that the connection is secure, the connection to the MQTT broker is established using TLS. TLS ensures that the communication is authenticated and secure.

In addition to TLS, a login is also used by each client connecting to the broker such that each client can be identified.

3.6 MongoDB

MongoDB is the database that will be used for this project. It is document-oriented database, and uses JSON-like objects to store information. It enforces no strict schema, and documents in the same collection can look very different. This requires the developer to either enforce a schema themselves or create code able to handle all the different possibilities. MongoDB has been chosen as it enables rapid development, and many of its downsides, such as being slow, unstable etc. are not an issue in a project of this scale.

3.7 Security

In this project the IEC 62443 [22] standard will be used to evaluate the security of the product. Only a subset of the standard will be used and this section will give an overview of how the standard is structured and what parts will be used and how.

The IEC 62443 is a standard for security in Industrial Automation and Control Systems (IACS). It covers everything from the entire plants down to the individual components, and includes both the development process and operation.[23] In this report the subject of development process will be disregarded and only the hard requirement for individual components will be used.

The first concept from the standard needed is the concept of security levels. Each part of the plant (so called zones) have individual security levels depending on the kinds of attack that needs to be prevented. The security levels (SL) range from 1 to 4.[23, Sec. 3.3.4]

- SL1 is protection against misconfiguration. This means that a level of access control is needed to ensure that a worker cannot accidentally change the configuration of machinery leading to a failure.
- SL2 is protection against intentional attacks using generic skills. This means that a worker that wants to gain access to information of control of machinery will not be able to using standard procedures.
- SL3 is protection against attacks using sophisticated techniques. This means protection against average hackers.
- SL4 is protection against attacks using sophisticated techniques, who have high motivation and patience. This means protection against attacks from large organizations or nation-states.

3.7.1 Security level

The required security level will be determined based on the use case presented earlier. In this use case, only data is made available on the network, and access to the mesh network does not allow an attacker to affect the operation of the machinery. Based on this, a security level of 2 should be sufficient. If Bluetooth mesh would also be used for configuration of machinery, a security level of 3 would be mandatory.

Based on this, the project will be evaluated for both SL2 and SL3

3.7.2 Component requirements

To evaluate if a component reaches a given security level a number of component requirement (CR) are used. These are split into 7 categories, the foundational requirements (FR) [22]

- FR 1: Identification and authentication
- FR 2: Use control
- FR 3: System integrity
- FR 4: Data confidentiality
- FR 5: Restricted data flow
- FR 6: Timely response to events
- FR 7: Resource availability

Each category have between 3 and 14 CRs. A great amount of the CRs are either not relevant to or out of scope for this project, and I have therefor chosen a small subset of CRs to use as the basis for evaluating the security of the solution. The following lists includes all the CRs the project will be evaluated from, including an abridged description. These descriptions do not cover all the requirements of the individual CRs, but are meant to give an overview of the purpose of each CR. The full description can be found in IEC 62443-4-2 section 5 to 11.

The following are needed for SL2:

- CR 1.2: All components shall be able to be identified and authenticated.
- CR 1.14: Symmetric keys need to be exchanged securely.
- CR 2.8: Components shall provide logs containing information about a specific list of events such as request errors, configuration changes etc.
- CR 3.1: All messages shall support message integrity and authentication checks.
- CR 6.2: Components shall be continuously monitored

The following additions are required for SL3:

- CR 1.2: All components shall be able to be uniquely identified and authenticated.
- CR 1.14: Symmetric keys need to be protected using hardware mechanisms.

4 Design

This section will present both the methods used to generate, as well as the final design.

4.1 Use-case

A use-case diagram has been generated based on the top-level requirements laid out in section 1.1. The diagram can be seen in figure 9. This diagram covers both the cloud and the gateway. As can be seen, some very general concepts are needed by the network administrator, which are then translated into some very concrete Bluetooth Mesh concepts.

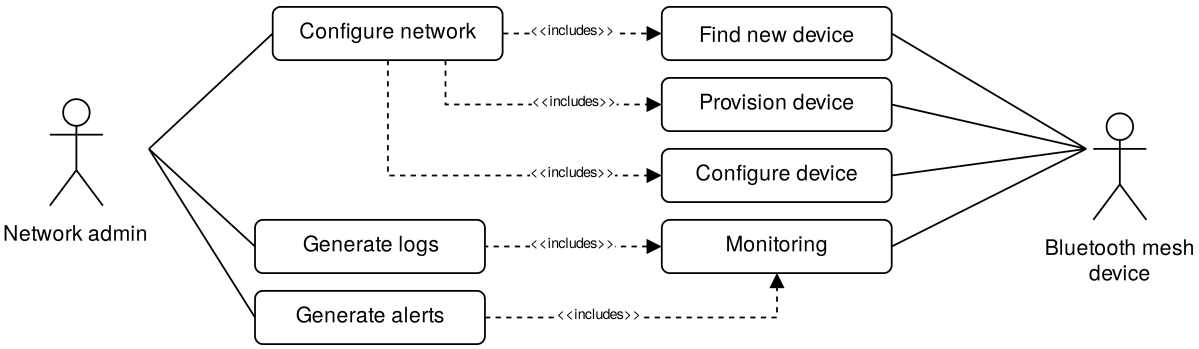

Figure 9: Use-case diagram

4.1.1 Design requirements

Based on the use-case diagram presented in section 4.1, the following architectural requirements were laid out.

- Server
  - Store information about networks
  - Store information about network configuration
  - Transform information about network configuration into instructions for setting up a network
  - Transform messages about network activity into a representation of a network

  - Generate alerts based on network monitoring

- Gateway
  - Receive messages, and take one of the following actions:
    - Provision a node
    - Configure a nodes keys
    - Configure a nodes publication address
    - Configure a nodes subscription address
    - Check if node is responsive
  - Send detailed information on network activity to the server, such as:
    - Success or failure to provision a node
    - Success or failure to configure a node
  - Generate logs on network activity, and send them to server

4.1.2 Communication diagram

Based on the architectural requirements, the communication diagram in figure 10 has been created. It represents the internal components of both the gateway and the server, and the lines illustrate the which components communicate. Each component is made up of functions, data and threads that is tightly coupled and have a logical connection.

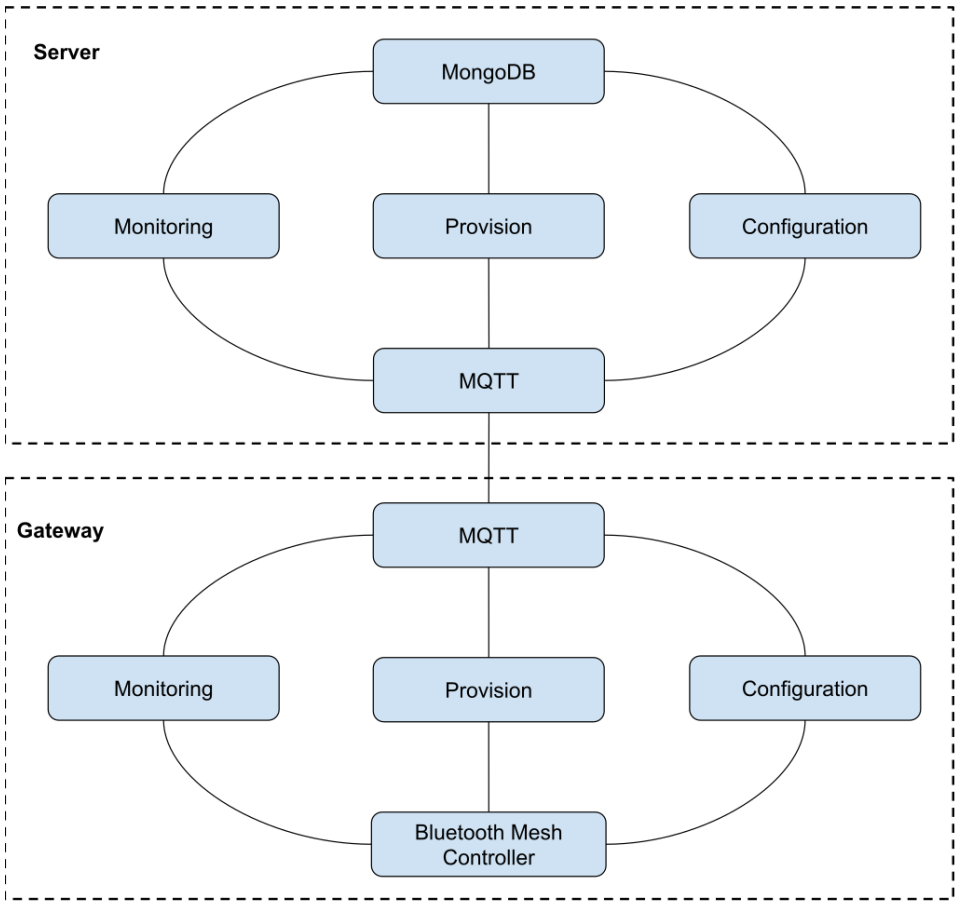

Figure 10: A block diagram illustrating the main components of both the server and gateway.

As can be seen from the image, the server and gateway has a similar layout, with components responsible for the three main tasks: monitoring, provisioning and configuration. These talk to MQTT to exchange information between server and gateway, and on the gateway, the three components communicate with the Bluetooth mesh software stack while the three components on the server communicate with a MongoDB database that store all information about the network setup, including which nodes should be provisioned and how they should be configured.

4.2 Configuration to instructions

The database will contain an abstract representation of the network and this will ultimately need to translated into a set of messages that can be send out via Bluetooth. This translation can either be handled by the server or the gateway. In the first case, the server will take the network representation and generate a single instruction, i.e. provision

a node. This instruction will be transferred to the gateway, which performs the tasks and answers back with a response. In the second case, the entire network will be transferred to the gateway, and the gateway will have to figure out which instructions need to be executed.

The first option was chosen for this project, to simplify the design of the gateway. As the server is typically written in a higher level language, it is more appropriate to leave the responsibility of this duty to the server.

4.3 Security

To ensure no unauthorized nodes are added to the network, all nodes have to use some type of OOB (Out of band) information. Bluetooth mesh supports several kinds, but for this purpose, static OOB have been chosen. This means that during production, a key will be programmed into the node, and this key will be sent to the server. During provisioning, the gateway will ask the server for the keys corresponding to the node it want to provision, and only if the keys match will the device be added to the network.

This relies on the fact that the flash memory containing the keys are encrypted to ensure that no attackers can gain access to it, and compromise the network.

Alternative methods of transferring OOB information, called dynamic OOB, circumvent this issue but presents new ones instead. An example of dynamic OOB is that the gateway generates a 6 digit number and displays it. A worker on site would then have to type that number into the node that should be provisioned.

This means that for each device, a person is needed, and the person should be able to determine if a node has been tampered with.

5 Implementation

5.1 Gateway

In this section the most important implementation details for the gateway will be described.

In figure 11 a picture of the gateway can be seen. It has three LEDs, that indicates that it is booted, that it is connected to the server and if any errors that may occur, respectively.

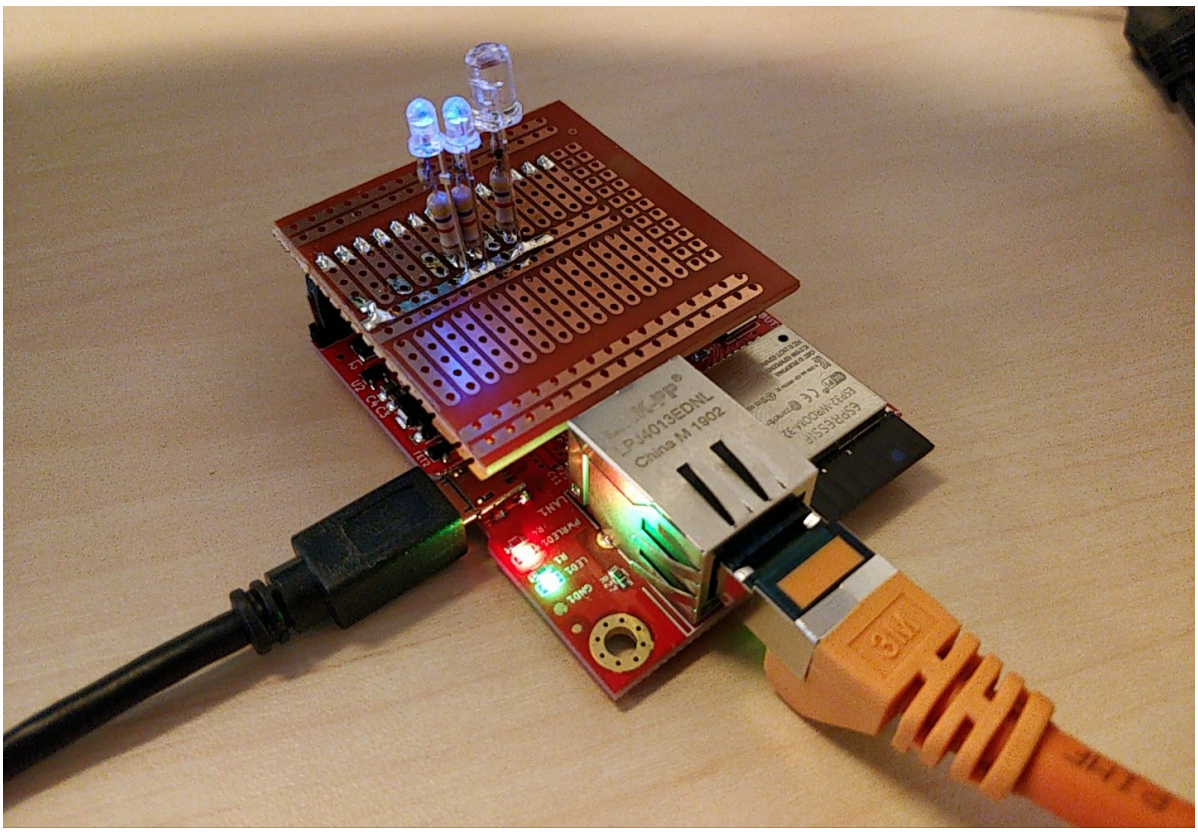

Figure 11: A picture of the gateway.

5.1.1 Task diagram

In figure 12 a communication diagram of the primary tasks inside of the gateway are presented, both application- and system-level.

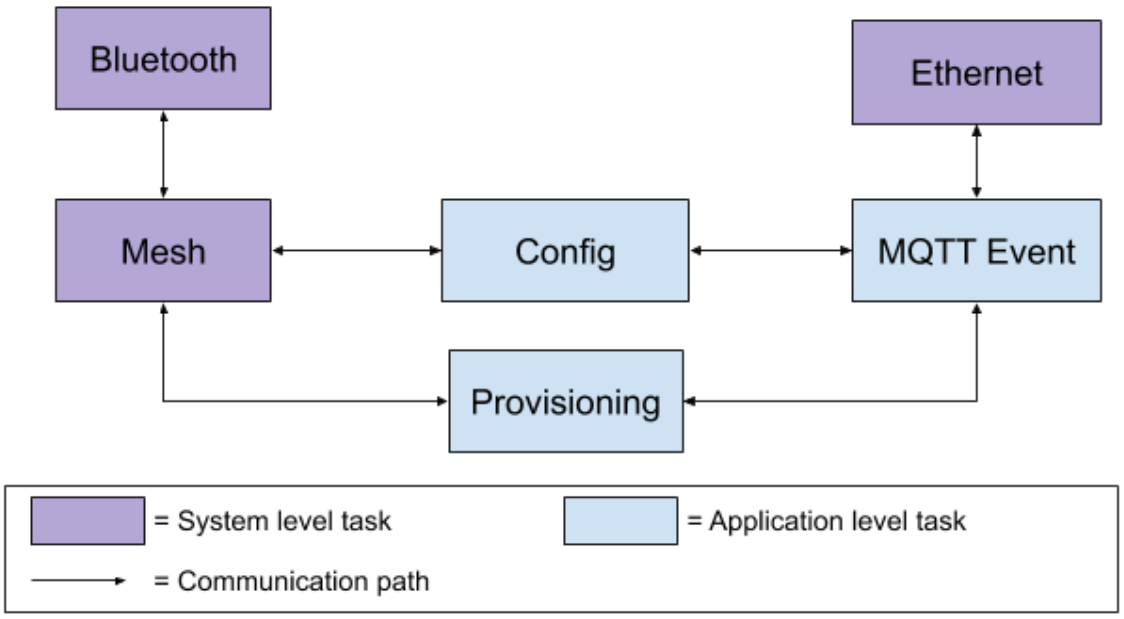

Figure 12: A diagram of the primary tasks used in tasks the gateway, and how they interact.

This diagram is very similar to the diagram presented in 10. It illustrates the flow of information, it can go from Bluetooth, through the mesh subsystem, though either the configuration or provisioning systems and out to Ethernet through the MQTT system, or it can take this route in reverse. One will notice that there is no monitoring task. Monitoring is handled mainly by the server though the configuration task on the gateway.

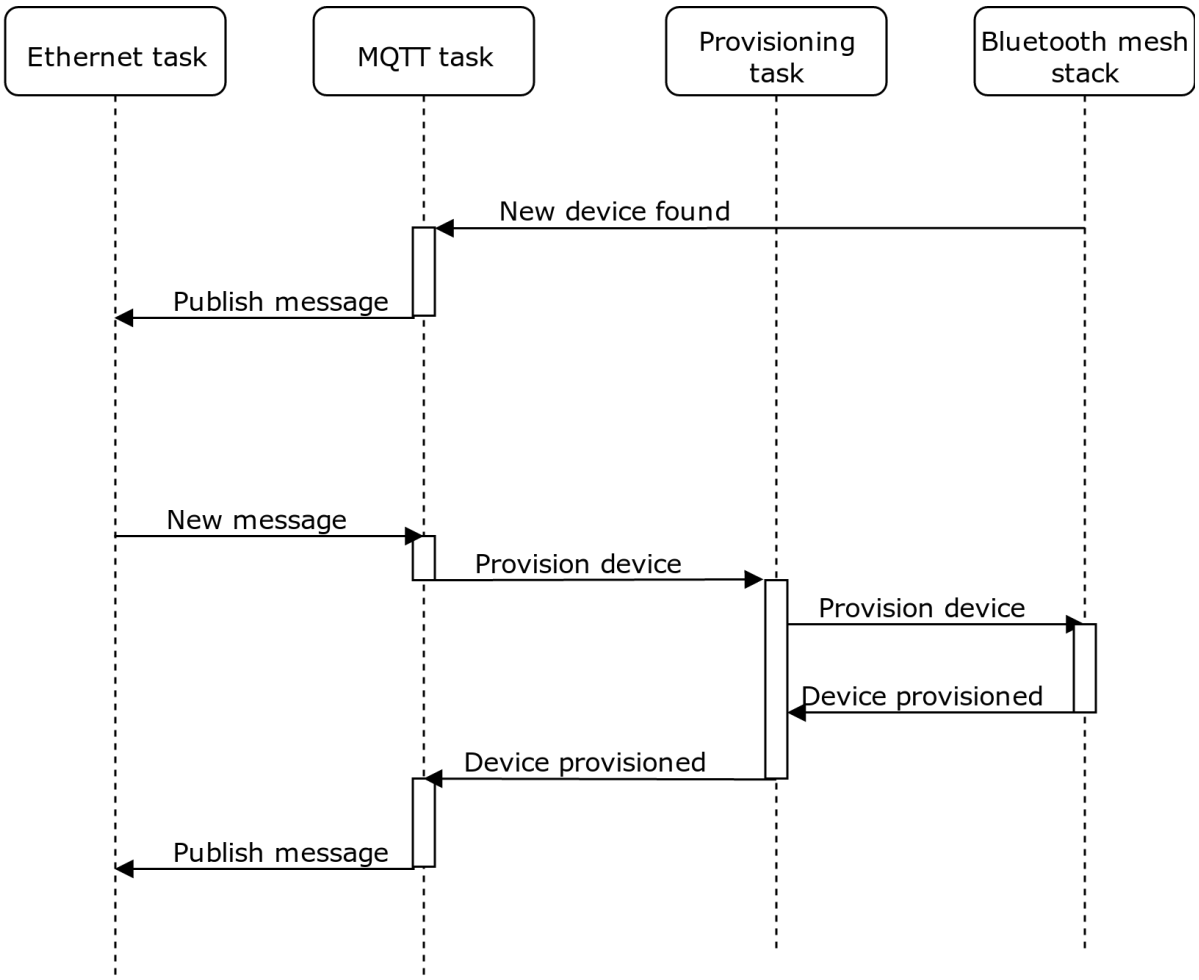

Figure 13: Sequence diagram showing an example of the Bluetooth stack detecting a new device, and that device being provisioned.

Figure 13 shows a sequence diagram illustrating a typical sequence of operations withing the gateway. First the mesh stack detects a new device, this is transmitted via MQTT to the server, which in turns sends a message telling the gateway to provision this device.

5.1.2 Resources restriction

The Bluetooth mesh software stack used in this project is structured such that each mesh model is limited to only one message at a time, and if two messages are sent too close to each other, the outcome is unpredictable from the application level. To overcome this, a very cautions strategy was deployed. Only one task is used to access each mesh model, and no new messages are sent until either a response has been revived from the previous message or a timeout occurs.

This also greatly simplifies the code, as it becomes trivial to connect a response to its corresponding request.

To achieve this, a mutex would usually be used, but in this specific case, the response is handled by a different task in the mesh software stack, and the way mutexes are implemented in FreeRTOS, only the task that locks a mutex can unlock it. This means a binary semaphore was needed.

As only one task has access to each model, other tasks need a mechanism for communicating to that task what messages it should send. This section will examine how this has been achieved for the configuration code.

When any task wants to send a configuration message, they use a configuration queue. Into this queue they put an opcode, along with any arguments needed. Here is an example of the code used to reset at node:

```
void config_reset(uint16_t addr) {

    // The buffer contains a 8bit opcode and 8 and 16 bit data.
    // This means that the first byte is always 8 bit,
    // while the following data can be accessed both as 8 and 16 bit data.
    uint8_t buf[MAX_CONFIG_QUEUE] = {0};
    uint16_t *buf16 = (uint16_t*)(&buf[1]);

    buf[0] = CONFIG_OP_RESET;
    buf16[0] = addr;
    xQueueSend(config_queue, (void * ) &buf, 0);
}
```

When the configuration tasks receives this message, it will use the first byte, the opcode, to determine the structure of the rest of the data and send the requested message. It can be seen below how the configuration task decodes the reset message that was shown above. Some code has been omitted for brevity:

```
void config_task() {
    uint8_t buf[MAX_CONFIG_QUEUE];
    while(true) {
        if(xQueueReceive(config_queue, &buf, 500)) {

            // This code attempts to take the semaphore.
            // If it is unable for 100 ticks,
            // the task will sleep for 100 ticks and attempt again.
            // This makes sure that the tasks let other
            // lower proirity tasks continue while its waiting for
            // the semaphore
            while(xSemaphoreTake(config_sem,100) == false) {
                vTaskDelay(100);
            }

            uint16_t *buf16 = (uint16_t*)(&buf[1]);
```

```
            switch(buf[0]) {
                case CONFIG_OP_APP_BIND:
                    _config_model_app_bind(buf16[0], buf16[1]);
                    break;

                case CONFIG_OP_RESET:
                    _config_reset(buf16[0]);
                    break;
            }
        }
    }
}
```

As can also be seen from the example, the configuration tasks needs the semaphore in order to send out a message. This semaphore is then released in the configuration client callback method. This method is invoked by the Bluetooth mesh subsystem whenever a configuration message has been received (this includes errors and timeouts). This means that the configuration task cannot send any new messages until the response for the previous has been received.

Another thing that can be seen from the above code is a primitive form of generic programming. The buffer that is send using the queue can contain a variety of data. The first 8 bits are always the opcode, but the following data can contain both 16 and 8 bit numbers. This means that there is usually kept both a `uint8_t` and a `uint16_t` pointer into the array. Another way to achieve this is treat the buffer as strictly 8 bits. The `config_reset` example from before would then look as follows.

```
void config_reset(uint16_t addr) {
    uint8_t buf[MAX_CONFIG_QUEUE] = {0};
    buf[0] = CONFIG_OP_RESET;
    buf[1] = addr & 0xFF; // Lower 8 bits
    buf[2] = (addr & 0xFF00) >> 8; // Upper 8 bits
    xQueueSend(config_queue, (void *) &buf, 0);
}
```

When reconstructing the 16 bit numbers more bit-wise operations would have been needed, and instead the current solution was chosen as it seemed more intuitive.

5.1.3 Logging

To enable monitoring of the gateway a logging system has been created. To make logs available remotely, all logs are transferred via MQTT. In the case that gateway cannot connect to MQTT e.g. due to network or MQTT broker issues, these logs are lost and a mechanism for local storage of logs is needed. This has not been implemented.

5.2 Server

In this section the most important implementation details for the server will be described.

The server was developed primarily as a demonstration of the capabilities of the gateway, and was very much secondary to the gateway.

It supports all functionality provided by the gateway, but it has some clear limitations.

The server uses MongoDB to store a representation of all the nodes that are expected in the network. A status is kept for all nodes, indicating if the node is available, if it has been provisioned and if it has been configured. Based on this information, a function will try to either provision or configure the device, if it is available and not already configured.

When the configuration of a device is changed on the server, the status is set to not configured, indicating that the device needs to be reconfigured. This means that all the configuration is overwritten, even if only a small part of the configuration has been changed.

To monitor if all nodes are running as expected, the server sends a message to all nodes. If a node does not respond to the message and alert is generated.

5.2.1 Network representation

The server stores a representation of the Bluetooth mesh network in the database. This section will describe how the network is represented.

The network is represented as a list of nodes. Each node has a name, a UUID, static OOB data, a configuration and a status. The status indicates whether or not the device has been found, provisioned and configured. The configuration contains a list of subscription- and publication addresses.

An example of a node using a JSON representation:

```
{
    "name":"Node-3",
    "uuid": "29651476-394b-46f3-8f97-5c0811437b8c",
    "oob" : "0fxef45",
    "subs":[{'addr':0xC000, 'model':0x1000}],
    "pub" :{'addr':0xC000, 'model':0x1001},
    "status":1
}
```

5.3 Nodes

The nodes used in the project were implemented mainly as a way to test the gateway, and a comparatively very simple. They contain a single mesh on/off client and a single mesh on/off server, and they are represented by a button and a LED respectively. This is enough to demonstrate the basic functionality of a mesh network.

In figure 14 a picture of the node can be seen. As can be seen, it sits on top of a power bank that was used for testing and demonstrations, while a bench power supply was used during development.

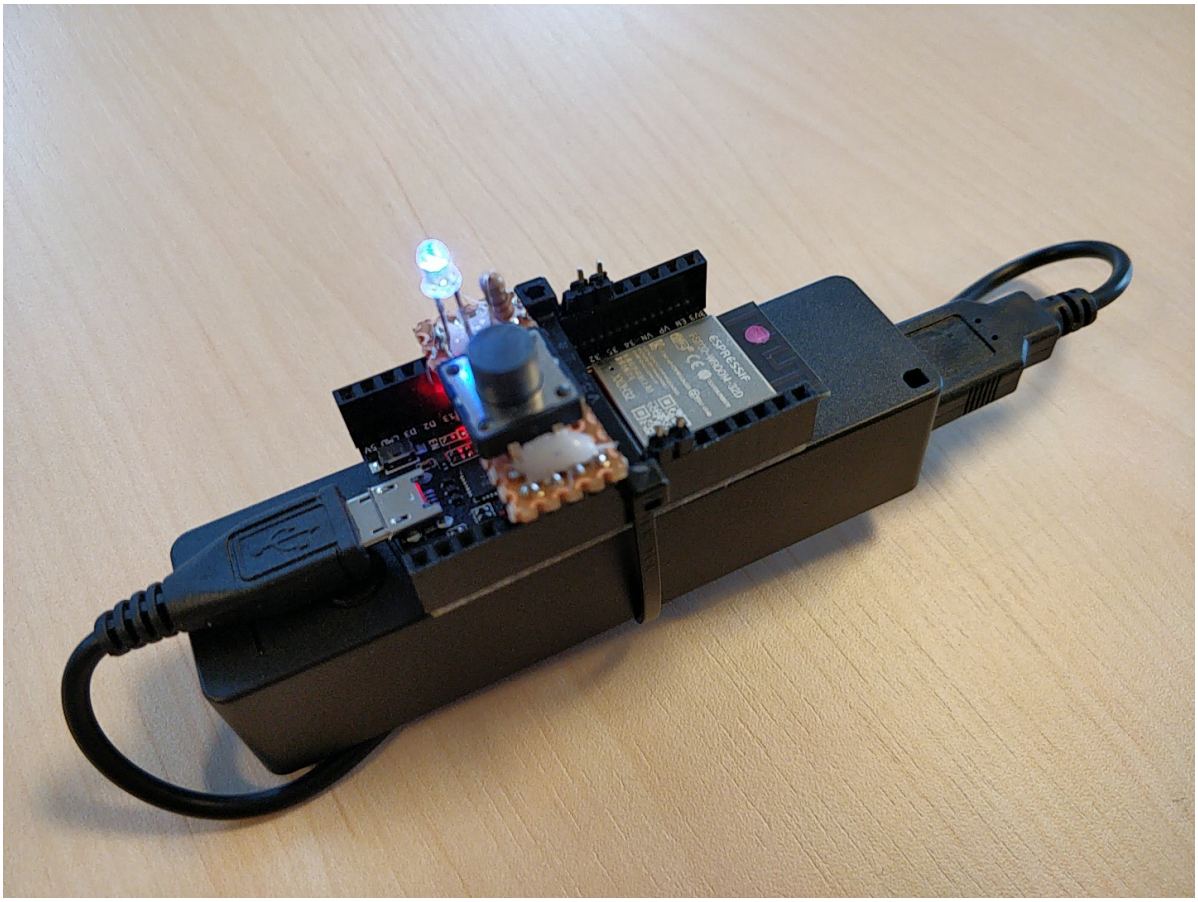

Figure 14: A picture of the node.

5.4 MQTT

MQTT was used to connect the server and the gateway. This section will describe how this was implemented and which topics where chosen.

MQTT uses a broker. This is a separate entity, in this case hosted on its own server, that handles how messages are passed between devices. For this project, a free MQTT broker was hosted using `cloudmqtt.com`. Both the server and client connects to the broker using HTTPS, meaning that the connection is encrypted using TLS. In addition to this, the gateway and server are given the public key beforehand, and it is hard-coded into both. Alternatively the clients can fetch the public key from the server itself, but this is susceptible to attacks where the traffic is routed to a malicious broker.

The MQTT topics are divided into two main categories. Topics that the gateway subscribe to and topics that the server subscribe to. The following are the topics which the gateway subscribe to:

- `/device/prov`: This is used to tell the gateway to provision a device.
- `/device/conf/sub/add`: This is used to add a subscription address to a model
- `/device/conf/sub/del`: This is used to remove a subscription address from a model
- `/device/conf/pub`: This is used to set the publication address for a model
- `/device/conf/key/add` : This is used to transfer an application key to a node
- `/device/conf/key/bind`: This is used to bind an application-key to a model
- `/device/conf/reset` : This is used to reset a node, and thereby removing it from the network
- `/device/conf/comp/get` : This is used to get a list of all the models and elements a node contains
- `/device/error` : This is used to display an error on the gateways error LED.

As can be seen, most topics have to do with the configuration of nodes and their models. The following are the topics which the server subscribe to:

- `/server/new`: This is used to inform the server of a unprovisioned device
- `/server/prov/callback` : This is used to return information about the most recent provision to the server
- `/server/conf/callback` : This is used to return information about the most recent configuration to the server
- `/server/log` : This is used to add things to the server log from the gateway

As can be seen from both list of topics, the topics start with either `device` or `server`. This indicates that there is only one server and one gateway. This is a limitation in the implementation that stems from the fact that this project was only developed with one of each. In the future, these could be replaced with a device- and server id.

5.5 Limitations

This implementation has some limitations compared to what is possible using Bluetooth mesh, and a few of these will be outlined in this section.

- The implementation does not support multiple application-keys. This means that all device on the network can read the messages of all other nodes. This is not in itself a security risk, given that all devices are trusted. If a breach does happens, and a malicious device gains access to the network, that device has access to more data than if multiple application-keys.
- The implementation only supports one gateway per server. It multiple gateways are placed in different locations, new servers are needed, including a new MQTT broker.
- The current implementation of the gateway does not store network configuration in non-volatile memory, and thus is looses all knowledge of the network during power-off. There is also no way to transfer this knowledge to the gateway.

6 Testing

Testing is a vital part of software development. Its primary purpose is to catch bugs, and this can lead to faster development times and increased confidence in the software.

Testing was done both at the end of the development cycle, but was also used during development. The two primary methods of testing deployed during this project was unit testing and functional tests.

6.1 Unit testing

Parts of the code was unit-tested and this section will explain how unit-testing and mocking was done.

Unit testing is the act of testing individual functions of the code. It is the lowest level of testing and aims to test as small a unit of code as possible at a time. It is done to ensure the basic level of functionality is correct and catch bugs before they surface in higher level tests.

Each functions should be tested multiple times using different inputs to ensure as many cases as possible are tested, and to ensure that the function handles errors correctly.

The framework, ESP-IDF, provides a library for unit testing called Unity [24]. As Unity in itself provided no ability to mock functions, a library called Fast Fake Functions, FFF, was used [25].

When unit testing embedded software that interacts with hardware and libraries, it is necessary to mock many of the function calls to test the functionality. The purpose of mocking is to create a fake version of a function that is called instead or in conjunction with the real counterpart[26]. The unit test can then examine how many times the function was called, what parameters it was called with etc. This is also done to isolate the test from actual hardware. As an example, a test case for the MQTT subsystem will be shown.

First the mocks for the function `mqtt_publish` were created.

```
FAKE_VALUE_FUNC(int, esp_mqtt_client_publish, esp_mqtt_client_handle_t,
    const char*, const char*, int, int, int);
```

Here, the macro `FAKE_VALUE_FUNC` takes as parameters the return type of the function to mock, the name of the function to mock and then the type of all arguments to the function. This creates the mock function.

Then the function `mqtt_send` can be tested.

```
TEST_CASE("mqtt_send", "[mqtt]")
{
  connected = true;
  mqtt_send("topic", "data", 5);
  TEST_ASSERT_EQUAL(1, esp_mqtt_client_publish_fake.call_count);
  TEST_ASSERT_EQUAL(0, strncmp(esp_mqtt_client_publish_fake.arg1_val, "
      topic", 6));
  TEST_ASSERT_EQUAL(0, strncmp(esp_mqtt_client_publish_fake.arg2_val, "
      data", 6));
  TEST_ASSERT_EQUAL(5, esp_mqtt_client_publish_fake.arg3_val);
}
```

This calls the function `mqtt_send`. This function in turn calls `mqtt_publish`. Afterwards, the macro `TEST_ASSERT_EQUAL` is used to test if the mock of `mqtt_publish` was called as expected with the expected parameters.

6.2 Functional testing

In this section, the functional tests that has been done will be discussed. Functional testing is a test where the system is tested in its entirety as a black box. This means that the gateway was tested in conjunction with the nodes and server. This was the primary type of test, as any test that did not involve either nodes or server would require a large amount of mocking [26] of external interfaces to achieve.

This test was used to test both functionality and later performance. The test was run for several hours during development to catch intermittent issues as a result of package loss or other network issues. At the end of the project, the test was run using different setups to gauge how the system performed under different circumstances.

One aspect of functional testing that was not used sufficiently was forced failure. Giving input to the gateway that were incorrect in some way, to make sure that the gateway could handle failure. This was mostly used at the end of the development cycle, which means that a lot of bugs in the gateway was not caught as early as they could have been.

6.2.1 Setup

The gateway and 4 nodes were placed according to several different layouts. The layouts were chosen in order to show the performance and resilience of the system at different levels of difficulty.

Then a script instructed the server to the following: 1. Provision all nodes 2. Configure all nodes to a certain configuration 3. Reconfigure all nodes to a new configuration 4. Reset all nodes

After the steps had been completed, the time was recorded and the test was run again.

The test was repeated at 20 times for each setup, both to reveal intermittent issues, stemming from e.g. lost or delayed packages, and to get a clear picture of the performance of the network at each level.

6.2.2 The layouts

6.2.2.1 Desk All devices was placed on a desk, within 30 cm of each other. There was a clear line-of-sight between all devices. This was used as a baseline test.

6.2.2.2 Small office Devices was placed around a small office. There was partial line-of-sight between most devices and clear line-of-sight between a few devices.

6.2.2.3 Two offices The gateway was placed in one office, while the nodes were placed in a second larger connecting office. Devices has no line-of-sight, and there is a wall or window between the gateway and 3 of the devices. There is no direct line-of-sight between each node and the gateway. The view is blocker by either a wall or a window, and often also several boxes and people. As both offices are in use this changes over time.

There is also several Bluetooth products in use in both offices (mice, headset etc.) that generate Bluetooth noise.

6.2.3 Results

The full results can be found in appendix B. Figure 15 show the box plot of the tests, where a few extreme outliers has been removed for clarity. As can be seen, the average time goes up slightly as the devices are spread out, but more importantly is that the variation goes up with distance.

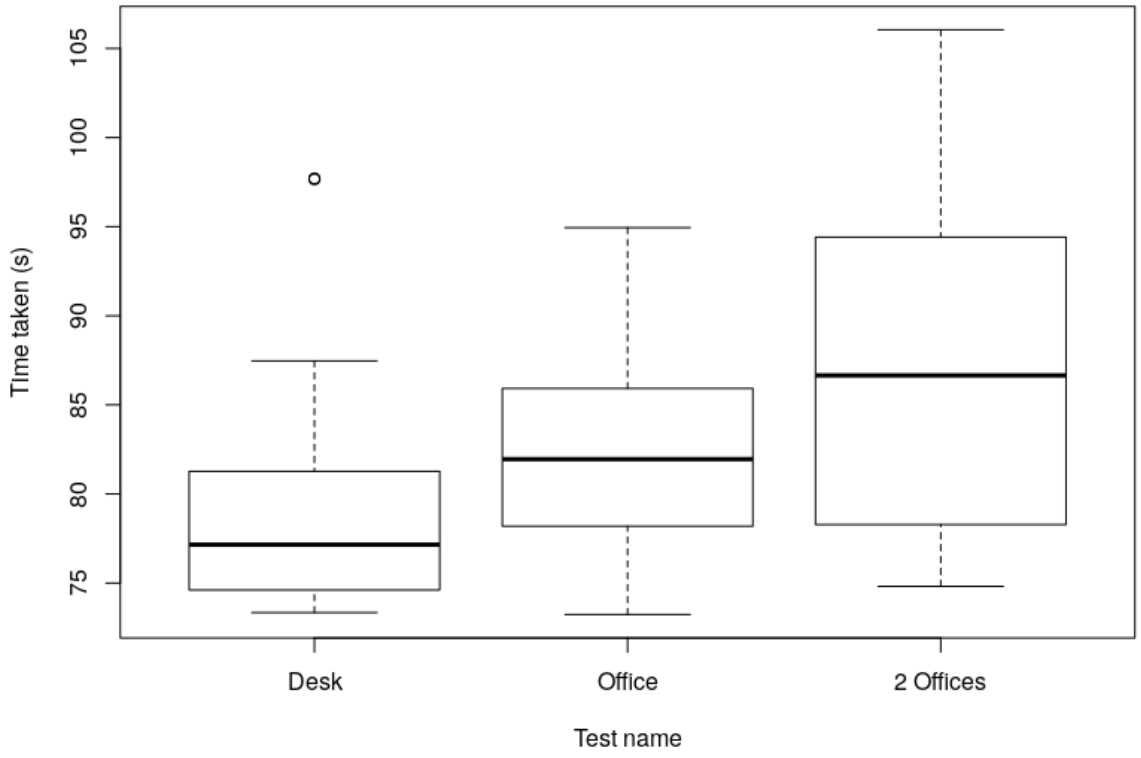

Figure 15: Boxplot of the test results

It can be concluded that the setup is viable in all three setups, but due to package loss and propagation time, performance is degraded with distance, as expected.

7 Evaluation

This section will examine if and how the project fulfills the requirements laid out in beginning, including the security requirements.

The initial requirements were:

- Enable remote provisioning of Bluetooth mesh networks
- Enable remote configuration of Bluetooth mesh networks
- Monitor a Bluetooth mesh network, and report missing devices to a central server
- Demonstrate the above points using at least 4 Bluetooth mesh devices
- The solution should live up to the industrial security standard IEC 62443, to a reasonable degree

As has been described in the implementation and test sections of the report, the first three requirements are fulfilled. The fourth requirement, the demonstration, was performed for the company, and has thus also been fulfilled.

The fifth requirement was that the solution should live up the industrial security standard IEC 62443. It was determined that a security level of 2 was necessary, while a security level of 3 was desirable.

A table containing a summary of the security evaluation can be seen in figure 16, and the argument for each CR goes as follows:

- CR 1.2: The gateway only communicates with the server and the Bluetooth mesh nodes. It can be identified and authenticated by the server using its MQTT login, and identified and authenticated by the nodes using its MAC address and device key.

- CR 1.14:The symmetric keys in the system are the key used for MQTT login and the keys used to authenticate the nodes. The first key is programmed directly into the device during production and the second key is transferred using MQTT. The communication with MQTT is encrypted and authenticated using TLS. The gateway can use secure boot and flash encryption to as a hardware mechanisms to secure all stored keys.

- CR 2.8: The gateway generates logs that are sent to the server via MQTT and stored locally in volatile memory.

- CR 3.1: The messages to the server use TLS to ensure message integrity and authentication. The messages to the nodes use the built in mechanisms from Bluetooth mesh to ensure message integrity and authentication.

- CR 6.2:The gateway enables continuously monitoring of nodes in the network and the server can monitor the gateway.

	Level 2 requirements	Level 3 requirements	Passed?
CR 1.2	All components shall be able to be identified and authenticated	All components shall be able to be uniquely identified and authenticated.	✔
CR 1.14	Symmetric keys need to be exchanged securely	Symmetric keys need to be protected using hardware mechanisms.	✔
CR 2.8	Components shall provide logs containing information about a specific list of events such as request errors,configuration changes etc		✔
CR 3.1	All messages shall support message integrity and authentication checks		✔
CR 6.2	Components shall be continuously monitored		✔

Figure 16: Summary of the security evaluation

Thus the project fulfills both the requirements for a rating of SL2 and SL3

It can then be concluded that the project lives up to all the requirements laid out at the beginning.

8 Further work

This sections outlines some of the work that is to be done, to bring the product into the field.

- There are the limitations mentioned in the implementation sections. These limitations are not inherent to the product, but are a result of the focus during this project.
- A more robust server implementation is needed. The current server is for demonstration purposes only.
- More unit- and functional testing is needed. As was outlined in the testing section only small parts of the system was unit tested, and the functional testing lacked a lot of forced failure tests. A more robust test setup using a specialized test-node that can be used to force certain Bluetooth mesh errors needs to be developed.
- The Bluetooth mesh health client was not utilized during this project. The health server allows a Bluetooth mesh node to make available its error status codes etc. A health client can then be used to read and report on this. This functionality would be a obvious addition to the product.

9 Conclusion

This project has examined the challenges presented with the use of Bluetooth mesh in Industrial IoT. It concluded that the main problems that needed to be solved was remote provision of networks, remote configuration of nodes and remote monitoring of nodes.

These issues was solved by creating a Bluetooth mesh gateway that allowed for all three, and could be controlled via a remote connection. A server and some nodes were developed to demonstrate the functionality and for use during testing.

Testing showed that the product fulfilled the requirements laid out.

The development process used several aspects of agile development. Especially the focus of first getting to a working minimal viable product, and then slowly adding features was utilized, instead of specifying and then developing the entire product in one cycle.

Both unit testing and functional testing was utilized to ensure that the product was functional and provide a level of confidence in the product.

The security of the gateway was evaluated using a standard for industrial security, IEC 62443. It was concluded that the product lived up to the security requirements, based on the security level needed for this type of product.

The initial requirements were:

- Enable remote provisioning of Bluetooth mesh networks
- Enable remote configuration of Bluetooth mesh networks
- Monitor a Bluetooth mesh network, and report missing devices to a central server
- Demonstrate the above points using at least 4 Bluetooth mesh devices
- The solution should live up to the industrial security standard IEC 62443, to a reasonable degree

And all four points were fulfilled.

Appendix A

The table below shows a gannt chart of the initial time estimates.

		Week														
Topic	Hours pr topic	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Project planning	23	10	1	1	1	1	1		1	1	1	1	1	1	1	1
Research Bluetooth mesh	15	10	5													
Choose a platform	5	5														
Setup development enviroement	5		5													
Research security	15		10	5												
Create nodes	15		5	10												
Develop architecture	20			5	15											
Implement provisioning	65				10	15	15		15	10						
Implement configuration	60					10	10		10	15	15					
Implement cloud connection	20											10	10			
Implement monitoring	50										10	15	15	10		
testing	15													10	5	
Report writing	120	5	5	5	5	5	5		5	5	5	5	5	10	25	30

Figure 17: The initial estimate

The next table shows the actual hours spend on each task.

		Week														
Topic	Hours pr topic	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Project planning	28	6	1	3	2	1	2		3	2	1	2	3	1		1
Research Bluetooth mesh	20	15	5													
Choose a platform	3	3														
Setup development enviroement	4		4													
Research security	14		6	8												
Create nodes	10		10													
Develop architecture	8			3	5											
Implement provisioning	25,5			12	10	4										
Implement configuration	20,5					11	6		4							
Implement cloud connection	33					13	6		7	5		2				
Implement monitoring	25						2		3	5	5	5		5		
Testing	19										3	5	4	5	2	
Report writing	141	6	6	6	5	3	9		11	9	5	2	20	19	22	18
	Hous pr week	30	32	32	22	32	25		28	21	14	16	27	30	24	19

Figure 18: The actual hours spend

Appendix B

During the functional testing, three test were conducted, in which the gateway provisions and configures 4 nodes. Each test was conducted 20 times, and the table below shows the raw results in seconds.

Desk (s)	Office (s)	Two offices (s)
97.706809	94.944276	93.033778
79.618475	86.723564	84.706552
83.526377	85.118591	95.789524
79.771042	73.235523	92.858088
77.631949	79.797049	77.491426
74.436912	83.029327	97.113968
74.533125	92.59411	106.044642
76.861116	81.130528	91.655788
75.216834	80.239783	192.30744
74.596964	78.81809	78.189492
73.345627	81.382325	98.391345
74.630055	90.520968	74.816668
75.552774	84.457689	78.785827
77.447282	77.112109	75.069986
82.776443	91.786655	82.69083
87.466056	82.884836	76.96064
97.655167	74.833284	85.451784
79.663504	82.526282	78.375093
75.459828	77.564868	89.902658
74.129568	76.94819	87.853493

References

[1] DPTechnics. (). Bluetooth® Mesh gateway and platform, available soon - DPTechnics, [Online]. Available: https://www.dptechnics.com/en/products/ble-mesh-gateway.html (visited on 10/07/2019).

[2] BLE-MESH. (). BLE-MESH.com, [Online]. Available: http://ble-mesh.com/ (visited on 10/07/2019).

[3] Fanstel. (). BWG832F WiFi BLE 5 IoT Gateway, [Online]. Available: https://www.fanstel.com/open-sources-gateway (visited on 10/07/2019).

[4] Telink Semiconductor. (). Telink Bluetooth Mesh, [Online]. Available: http://www.telink-semi.com/archives/applications/bluetoothmesh (visited on 10/07/2019).

[5] BlueRange. (). Technology - Our BLE Mesh and Smart Beacon technology | BlueRange, [Online]. Available: https://www.bluerange.io/technology/ (visited on 10/07/2019).

[6] Ken Kolderup. (). Introducing BluetoothMesh Networking, [Online]. Available: https://www.bluetooth.com/blog/introducing-bluetooth-mesh-networking/ (visited on 09/20/2019).

[7] Bluetooth SIG, *Mesh Profile Specification 1.0.1*, Jan. 21, 2019.

[8] ——, *Mesh Model Specification 1.0.1*, Jan. 21, 2019.

[9] Nordic Semiconductor. (). S140 SoftDevice Scheduling, [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsds_s140%2FSDS%2Fs1xx%2Fmultilink_scheduling%2Fmultilink_scheduling.html (visited on 09/20/2019).

[10] Zephyr Project. (). About Zephyr, [Online]. Available: https://www.zephyrproject.org/about/ (visited on 09/20/2019).

[11] Z. Zephyr Project. (Aug. 21, 2017). Announcing Bluetooth Mesh support in Zephyr Project, [Online]. Available: https://www.zephyrproject.org/announcing-bluetooth-mesh-support-in-zephyr-project/ (visited on 09/20/2019).

[12] T. Svehagen. (). Bluetooth: Mesh: Add support for provisioning remote devices by tsvehagen · Pull Request #17729 · zephyrproject-rtos/zephyr, [Online]. Available: https://github.com/zephyrproject-rtos/zephyr/pull/17729 (visited on 09/20/2019).

[13] Espressif Systems. (). Espressif's BLE Mesh SDK Is Now Bluetooth-SIG-certified, [Online]. Available: https://www.espressif.com/en/news/Espressifs_BLE_Mesh_SDK_Is_Now_Bluetooth_SIG_certified (visited on 09/20/2019).

[14] BlueZ. (). BlueZ mesh-api, [Online]. Available: https://github.com/hadess/bluez (visited on 09/20/2019).

[15] Olimex. (). ESP32-GATEWAY - Open Source Hardware Board, [Online]. Available: https://www.olimex.com/Products/IoT/ESP32/ESP32-GATEWAY/open-source-hardware (visited on 11/20/2019).

[16] Espressif Systems, *ESP32 Series Datasheet Version 3.1*.

[17] ——, (). Flash Encryption — ESP-IDF Programming Guide v4.1-dev-279-g96b96ae24 documentation, [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/security/flash-encryption.html (visited on 09/23/2019).

[18] ——, (). Secure Boot — ESP-IDF Programming Guide v4.1-dev-279-g96b96ae24 documentation, [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/security/secure-boot.html (visited on 09/23/2019).

[19] ——, *ESP32Technical Reference Manual Version 4.0*. 2018.

[20] ——, (). Security Advisory concerning fault injection and eFuse protections (CVE-2019-17391) | Espressif Systems, [Online]. Available: https://www.espressif.com/en/news/Security_Advisory_Concerning_Fault_Injection_and_eFuse_Protections (visited on 11/20/2019).

[21] E. Staff. (). 2017 Embedded Market Survey, [Online]. Available: https://www.embedded.com/electronics-blogs/embedded-market-surveys/4458724/2017-Embedded-Market-Survey (visited on 09/26/2019).

[22] IEC, *Security for Industrial Automation and Control Systems – Part 4-2: Technical Security Requirements for IACS Components*. Feb. 2019, ISBN: 978-2-8322-6597-0.

[23] TÜViT, *Whitepaper Industrial Security Based on IEC 62443*.

[24] Espressif Systems. (). Unit Testing in ESP32 — ESP-IDF Programming Guide v4.1-dev-975-gd57890cdf documentation, [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/unit-tests.html (visited on 11/18/2019).

[25] M. Long, *Meekrosoft/fff*, Nov. 13, 2019. [Online]. Available: https://github.com/meekrosoft/fff (visited on 11/18/2019).

[26] Embedded.com. (Oct. 16, 2012). The mock object approach to test-driven development, [Online]. Available: https://www.embedded.com/the-mock-object-approach-to-test-driven-development/ (visited on 11/01/2019).