FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Bluetooth Mesh Networks for Home Automation Applications

Francisco Gil Cerqueira Correia

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Supervisor: Paulo José Lopes Machado Portugal

July 6, 2020

Abstract

Internet of Things (IoT), is a concept that has been getting a lot of attention and has seen a lot of applications in devices used in industrial and home environments. Currently in the market, there are a broad amount of IoT solutions for wireless network applications focused on home automation. These solutions however share some common problems, such as high-cost network modules, making them unsuitable for low-cost and simple applications like switches and light bubs, and a reliance on proprietary communication protocols, leading to compatibility problems or even interoperability between different manufacturer devices.

Recently, with the appearance of low-cost modules including Wi-Fi and Bluetooth interfaces plus integrated digital and analogue input and outputs (I/O), the possibility of creating new solutions making use of these modules platforms was made available, heavily reducing costs for these applications.

The increased availability and need for wireless network applications, also leads to the development of standards and protocols specific for this type applications, some of which are open, like the Message Queuing Telemetry Transport (MQTT) protocol and, more recently, Bluetooth Mesh for Bluetooth Low Energy (BLE) enabled devices. These protocols, when paired with low cost wireless modules, can enable low-cost and open solutions for home automation.

Also, due to the variety of solutions and communication protocols involved in home automation, a number of platforms appeared, which can integrate and offer a ways to coordinate and interact with solutions from different manufacturers.

In this dissertation, a network and software architecture solution for home automation is proposed in order to make use of the recently made available open implementations of Bluetooth Mesh for low cost modules, and validate the Bluetooth mesh performance in a home environment including support for the available home automation platforms.

Resumo

Existem atualmente no mercado um conjunto vasto de soluções para redes domóticas sem fios. Contudo, todas estas soluções partilham alguns problemas como o custo elevado dos módulos de rede, o que os torna impraticáveis em soluções de baixo custo como interruptores e iluminação, e serem baseadas em soluções proprietárias, levando a problemas de incompatibilidade e/ou de interoperabilidade entre equipamentos de diferentes fabricantes.

Recentemente, com o aparecimento de módulos de baixo custo que disponibilizam em plataformas baseadas em microcontroladores com interfaces Wi-Fi, Bluetooth, e entradas e saídas digitais e analógicas incorporadas, abre-se o potential de implementação de aplicações para redes domóticas com estas plataformas, reduzindo drasticamente o preço das mesmas.

A crescente disponibilidade e necessidade de aplicações de redes sem fios, leva ao desenvolvimento the protocolos e normas especificas para este tipo de aplicações, algumas das quais são abertas, como o *Message Queuing Telemetry Transport* (MQTT) e, mais recentemente, *Bluetooth Mesh* para dispositivos com *Bluetooth Low Energy* (BLE). Este protocolos, quando unidos com plataformas com comunicação sem fios de baixo custo, possibilitam a criação de soluções, também elas de baixo custo para redes domóticas.

Devido a quantidade de diferentes soluções com diversos protocolos de comunicação envolvidos, plataformas de "Home Automation" começaram a ficar disponíveis como uma maneira de integrar estes diferentes dispositivos, habilitando a coordenação e interação entre os mesmos, independentemente do seu fabricante.

Nesta dissertação é proposta uma solução para redes domóticas e arquitetura de software para aplicações nestas redes, utilizando as recentemente disponíveis implementações de *Bluetooth Mesh* para módulos de baixo custo e validar o seu desempenho num ambiente doméstico, incluindo suporte para plataformas de domótica disponíveis.

Contents

List of Figures

List of Tables

Abbreviations and Symbols

API	Application Programming Interface
App	Application
BLE	Bluetooth Low Energy
Bluetooth SIG	Bluetooth Special Interest Group
ESP-IDF	Espressif IoT Development Framework
GPIO	General Purpose Input and Output
ID	Identification
I/O	Input and Output
IoT	Internet of Things
LAN	Local Area Network
AP	Access Point
LR-WPAN	Low-Rate Wireless Personal AREA Network
MQTT	Message Queuing Telemetry Transport
Opcode	Operation Code
RTOS	Real Time Operating System
SoC	System-on-a-Chip
TTL	Time-To-Live
UI	User Interface

Chapter 1

Introduction

This first chapter presents the motivation, context and objectives for this dissertation and its structure.

1.1 Motivation

The home automation market has recently seen a renewed interest with the advent of IoT systems, a term used to describe systems that can connect to the Internet or even just connect and share information with other systems in the same network[49].

Most solutions for home automation on the market, that try to build on this concept of IoT, are reliant on modules to enable communication which are comparatively expensive within the context of home automation where some systems can be as simple as controlling a lightbulb or enabling wireless switches.

These solutions also suffer from proprietary, manufacturer specific, and closed communication protocols with a reliance on expensive proprietary gateways to allow data to flow to and from the system's local network. This introduces problems with compatibility between solutions from different manufacturers and locks the consumer to a specific manufacturer, or group of manufacturers, solutions unless he acquires a different gateway[48].

When developing new solutions for home automation, developers have to invest time in order to choose an adequate protocol for the system's needs and to implement the same protocol on the chosen platform. If the protocol to be implemented is not open, the developer might also have to invest time, to learn new technologies, and money in order to access essential documentation as seen, for example, from the KNX Association[31].

1.2 Context

The appearance of microcontrollers with wireless communication interfaces, specifically Bluetooth and Wi-Fi, in low-cost modules, makes developing new solutions to be integrated in home automation cheaper and consequently more accessible to both developers and consumers.

Also relevant is the availability of open protocols specifically developed for IoT or home automation with implementations, for an increasing number of modules that can be easily integrated in new systems. Bluetooth Mesh is one of these protocols, with recently made available open implementations in low-cost modules.

Due to the variety of devices and protocols employed in home automation devices, platforms that aim to support these devices have appeared, delivering a common interface for different manufacturers devices and ways to coordinate them, all which can be accessed remotely.

1.3 Objectives

The goal for this thesis is to define a Bluetooth Mesh network and provide an open and modular software architecture, making use of open protocol implementations, which should be able to cover the most common applications for home automation. The software architecture should also abstract the communication covered by these protocols in order to streamline and simplify node development, deployment, and configuration for developers who just want to add communication features to their devices without having to worry about these protocols. This approach allows interested developers to contribute and benefit from each other's work, which should lead to more featureful and accessible home automation devices.

1.4 Document Structure

This document is structured in six different chapters:

- chapter 1 contains the motivation context and objectives for this dissertation;
- chapter 2 presents and describes relevant technologies for this dissertation's objective;
- chapter 3 contains the proposed solution taking into account the discussed technologies;
- chapter 4 goes over how the proposed solution was implemented;
- chapter 5 contains tests and the validation of the proposed solution with the presented implementation;
- chapter 6 describes the achievements made with this work, and presents possible ways to expand or enhance it.

Chapter 2

State of the Art

This chapter covers the state of the art on the most relevant topics for this dissertation. These topics are divided in three sections:

- home automation platforms hosted locally or in the cloud;
- standards and protocols for communication between nodes in a home automation network and to outside the network;
- Network node's hardware and software.

2.1 Home Automation Platforms

Home automation platforms enable users to interact with their IoT devices through their computer or any internet capable devices common browsers or applications. A good home automation platform should be able to provide users with easily understood user interface (UI) elements that behave independently of the underlying technology behind the device their trying to observe or control.

For the purpose of this dissertation, only platforms with tools allowing users to create and run scripts on said platform were considered. This feature makes it so coordination between devices will be left entirely to the platform and the user, circumventing any need for the developer to work on compatibility with devices from different manufacturers or with different communication protocols from the one chosen for the solution. The developer can then focus only on implementing the devices features.

There is an assortment of home automation platforms such as *thinger.io*[26], *ThingsBoard*[28] and *openHab*[23], however the focus of this dissertation will be on *Home Assistant*[12]. *Home Assistant* was chosen due to being completely free and open source, unlike some of the other options which lock features behind a paywall[27], and better integration and documentation for the chosen technologies, covered in Chapter 3. All discussed platforms have similar device and technologies support making it so the solution presented in this dissertation should be compatible with any of them[25][29][24].

2.1.1 Home Assistant

Home Assistant is an open source home automation platform with support for the most common home automation communication protocols such as *Zigbee*, *Z-wave*, and *Message Queuing Telemetry Transport* (MQTT)[13]. It supports device discovery for the supported integrations, automatically creating and configuring devices on the network and creating a suitable UI element for the user to interact with. Integrations are add-ons that can be installed onto the platform and facilitate communication with the devices they were built for. These add-ons can be supplied by either the manufacturers or built by the community. Installation of existing integrations is done directly through *Home Assistant*'s interface[11].

The *Home Assistant* user interface, *Lovelace*[14], operates on a card basis layout, with cards created from integrations without user input, and allows for users to add and manage these cards to their preference and bind them to whatever device they want to control. Lovelace already comes with cards that should cover most users needs, such as cards that can act as buttons and display sensor information to name a few. However, if the developer sees fit, it can create cards to better suit their devices features and make them available for the users, either through sharing the new card's files and and configuration needed to make it available on the interface, or through contributing to the open source project by having it included in an integration or in Home Assistant's core features. An example of the Lovelace UI can be see in figure 2.1

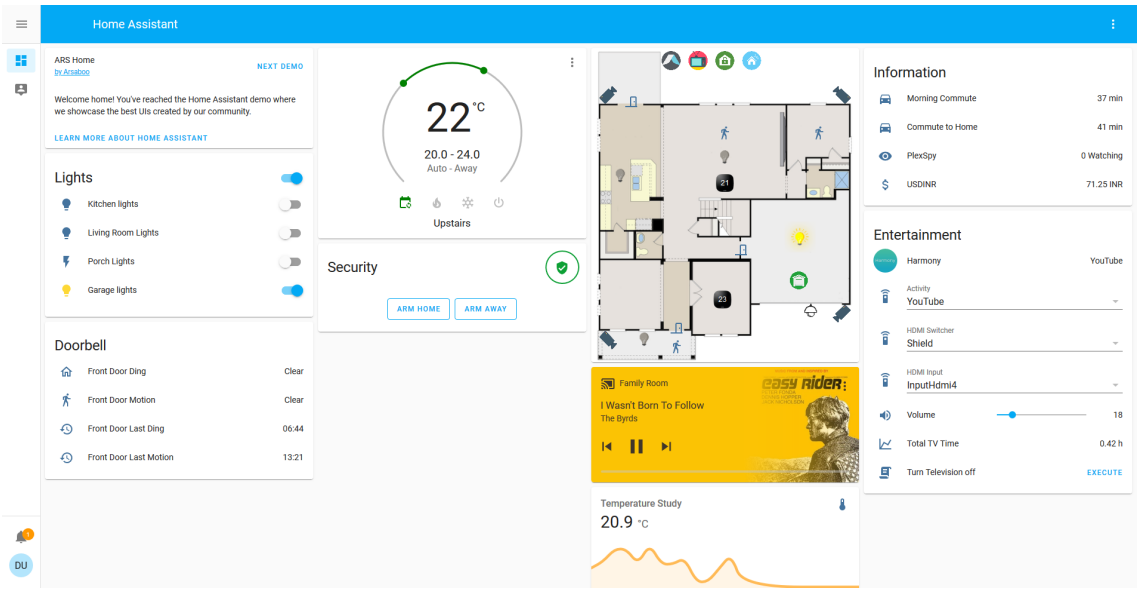

Figure 2.1: Home Assistant demo UI in browser[10]

Home Assistant can be installed and run locally on the home network where most of the devices it interfaces with are, or it can be set to run remotely in the cloud. Setting it up locally can be a better option as some integrations discovery and automatic setup only work on devices within the same local network. The platform can be setup in any device as long as it possesses

internet capabilities and it can run *Python* scripts, although experience may vary depending on the processing power of the device[15].

2.2 Wireless Communication Standards and Protocols

IoT devices are defined by their ability to be connected to the Internet or to communicate with other devices. The standards and protocols covered in this section dictate how these devices can realize their connections between each other wirelessly and to remote services to enable integration with home automation platforms.

2.2.1 IEEE 802.11 (WI-FI)

IEEE 802.11 is part of the *IEEE 802* set of local area network (LAN) protocols published by the IEEE association and defines a set of protocols for implementing a wireless local area network (WLAN) used commonly for Wi-Fi. Wi-Fi communication happens most commonly in the 2.4GHz band of the radio frequency spectrum.

Wi-Fi is available at every home with a router supporting wireless communication, serving as a bridge for the local network and the internet. Most devices will run untethered, without wired communication to other devices, making Wi-Fi support a requirement if there is ever a need to expose a device to remote access and allow users to control and supervise it from outside the home network.

This standard describes two basic modes of operation: *ad-hoc* and *infrastructure*. In *ad-hoc* networks, devices connect directly to each other to communicate. This mode is mostly for spontaneous and temporary connections, making it look like a bad fit for this dissertations goal. As such, this document will focus on *infrastructure* mode of operation which is the default mode of operation for home routers. In this mode, devices do not connect directly to each other. Instead their connections follow a star topology network, meaning it has a central node, the Access Point (AP), to which all other nodes connect, as represented in Figure 2.2, and is responsible for relaying messages between different devices in the network[51] .

With Wi-Fi availability in every home, its reasonable effective range as demonstrated in [44], and the added bonus of IoT devices usually requiring low bandwidth, so they can be reliably deployed in areas with bad coverage, solutions with exclusive reliance on this standard readily available[37].

While Wi-Fi can reach far, it does so with a cost. When compared with some other solutions, such as *IEEE 802.15.4*, Wi-Fi shows higher power consumption for the same range[53]. For devices that will be running without break and devices with a limited power envelope, restricted by their power supply or battery, an increased power consumption will increase energy cost per device, also decreasing battery life for battery powered devices.

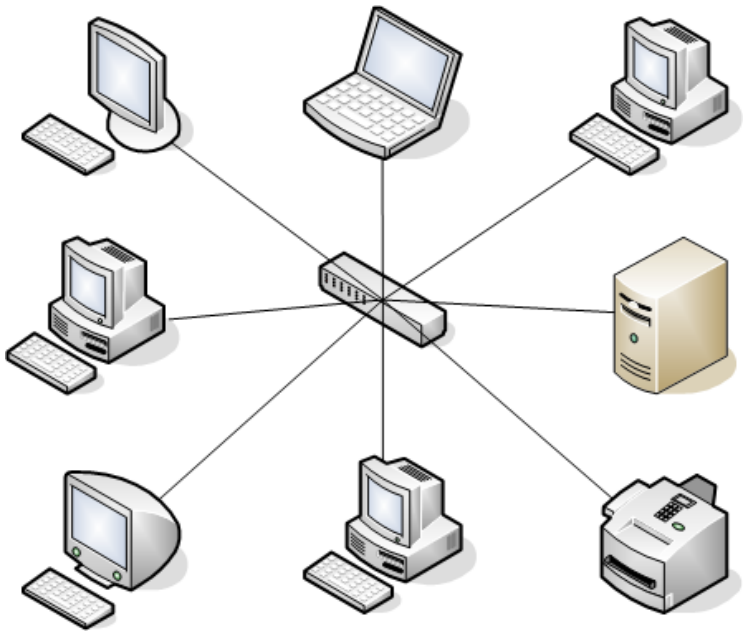

Figure 2.2: Star topology network with a *IEEE 802.11* compliant router at its centre

2.2.2 Bluetooth

Bluetooth is a wireless communication standard, created and managed by the *Bluetooth Special Interest Group* (Bluetooth SIG). Bluetooth networks follow a point-to-point topology, where devices establish a connection and communicate directly. Communication between devices follows a master-slave model, meaning communication in the network by a single device, the master device. One device can connect to several other devices creating piconets, as seen in figure 2.3[35].

It is mainly used to stream data, music streaming or file sharing, and to configure wireless devices. It is widely available on smartphones and a variety of other electronic devices, such as headphones, speaker, smartwatches, medical appliances and some other IoT devices[54][45][39].

Bluetooth communication, like Wi-Fi, also happens in the 2.4 GHz band. Bluetooth splits this band in several channels. These channels can be categorized in either advertising channels, connecting or communication channels.Communication channels are where all communication between master and slaves happen after establishing a connection, while advertisement channels are used to find nearby devices and connecting channel exist to negotiate a connection[35].

Bluetooth connections require *pairing* in order to establish secure connections. To setup a Bluetooth connection, a device must first discover available nearby devices by sending enquiry requests on a dedicated channel and store the responses to the enquiries. After receiving the responses a device can try no initiate a connection with one of the devices found from the enquiry. To enable data exchange between two devices they first must go through a pairing process to establish a connection link key in order to encrypt the data shared. Pairing is usually done through a numeric comparison, where one device display a number and the same number must be input in the second device. However, devices with limited I/O, such as having no screen or keyboard, can't

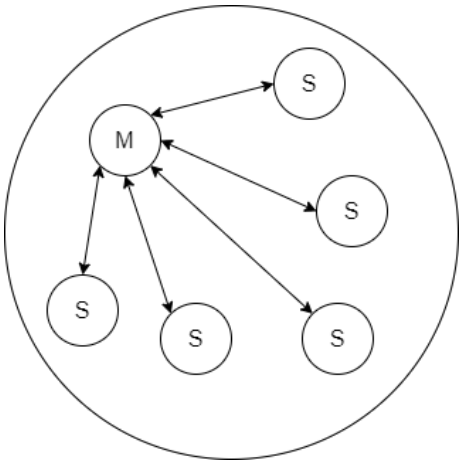

Figure 2.3: Bluetooth Piconet Network

rely on this method. The devices will establish a connection with the first inquiring device once they start advertising their availability[46].

2.2.3 Bluetooth Low Energy (BLE)

BLE comes as an alternative technology from *Bluetooth SIG* for low power devices communication. It supports low power transmissions, with lower transfer rates, and enables different types of network topologies, when compared to standard Bluetooth. Communication between BLE devices happens mostly through the advertisement channel and advertisement packets. This enables devices such as beacons, where their only purpose is to broadcast their state or other unique information, with varied uses such as indoor tracking and others as listed and briefly covered in [40]. The advertisement channel can also be used to initiate and establishing connections creating piconets similar to normal Bluetooth, where one device, acting as a master, can connect and communicate with several other devices in a different channel.

BLE lends itself well to the creation of mesh networks as shown in [56] and [36] leading eventually to the release of a Bluetooth mesh standard published by *Bluetooth SIG*[34] which will be the focus for this dissertation[32].

While BLE shares the same 2.4GHz band as Wi-Fi interference between these two shouldn't be a problem as shown in [47], due to it's use of three spread apart advertisement channels and channel hopping when establishing connections.

These BLE networks should be comparable to *IEEE 802.15.4* networks, another standard which defines a set of protocols for implementing low-rate wireless personal networks (LR-WPANs), enabling the same kind of solutions and a good candidate for establishing connections between IoT devices. BLE however has the advantage of being supported by most wireless enabled devices, covered in Section 2.3.

2.2.4 Bluetooth Mesh

Bluetooth mesh is a specification released by Bluetooth SIG which defines requirements to enable an interoperable mesh networking solution on top of BLE[34]. The exchange of data within the network follows the publisher-subscriber paradigm. Nodes within the network can publish messages to unicast, group, virtual addresses or broadcast them, and nodes wishing to receive messages addressed to group or virtual addresses are required to subscribe to those addresses. In mesh networks, devices can connect to other neighbour devices in order to create paths for messages in the network to travel, as visualized in Figure 2.4.

Due to the mesh nature of this network, a message travel distance is mostly limited by network density and its time to live (TTL). This allows devices to send messages beyond their BLE range, through message relaying, by neighbour nodes.

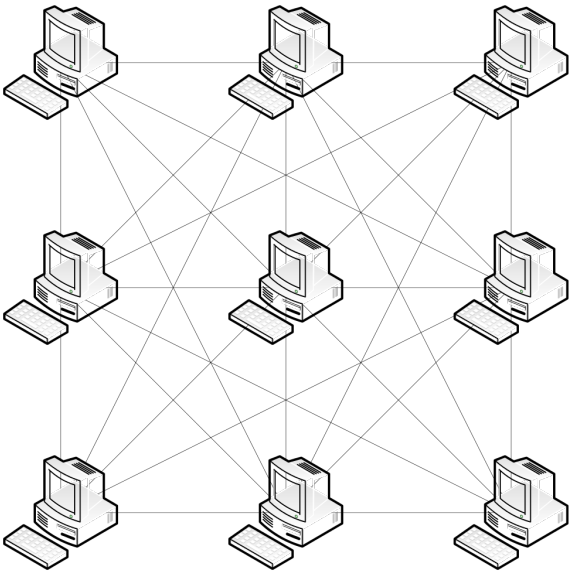

Figure 2.4: Mesh Network

Messages in the network are shared exclusively through the BLE advertisement channels, as such nodes do not rely on BLE connections to communicate with each other, so there are no piconets created, no master-slave communications. Instead of establishing connections to send messages, Bluetooth mesh relies on a managed flooding approach to publish and relay messages. In a managed flooding network published messages will be relayed by nodes who receive them, decreasing their TTL every time it passes through a new node, until the message's TTL reaches 0 or it reaches its destination. At this point, the message is discarded at the receiving node instead of being relayed. Managed flooding has a few advantages and disadvantages. Due to not relying on pre-established routes for message passing, the network is more resistant to node failure since, in a sufficiently dense network, messages can take several path on their way from the source to the

target node or nodes. This means there is no time spent in reorganizing the network after detecting a dead node and there's a smaller chance a message does not get to its destination.

This approach, however, can also backfire in high density networks. Since nodes are relaying repeated messages there is a lot of wasted energy and channel bandwidth to messages that will serve no purpose as they will be ignored by the target nodes after they receive the first one. This can also lead to network degradation, with higher packet loss. In these cases, careful management of message's TTL and selective relay feature support for nodes can help mitigate some of these problems[16].

Messages shared within the network are encrypted with two keys: a network key and an application (app) key. A network key is shared by all nodes within the network and is used to encrypt fields in a message belonging to the upper layers of the Bluetooth mesh stack. These fields include information such as the involved nodes address and the messages TTL. This ensures every node in the network can process these fields, without being able to access the data that might not be directed to them. Application keys are attributed to models in a node and are used to encrypt the fields belonging to the lower layer of the stack which includes the message content. Models part of a specific Bluetooth mesh network application can then share messages with each other without outside interference while still relying on foreign nodes in the network to relay messages. Network keys and application keys are attributed to nodes during node provisioning, however application keys can be added and managed later by the provisioner.

To provision a node to the network, another BLE device is required. It can be a node on the network or an external device, usually a BLE enabled smartphone or personal computer. To provision a new node into an existing network, the device doing the provisioning, the provisioner, requires the network key and assigned addresses. In the case of a new network, a network key must be generated before starting device provisioning. Failure to know all the assigned addresses can result in network nodes sharing the same address leading to undesired behaviour.

Bluetooth mesh specification splits some of its nodes specified features and allows nodes with only partial feature support to participate in the network. Some of these features are:

- **Relay feature** — enable devices to retransmit Bluetooth Mesh messages, enabling larger networks.
- **Low Power feature** — enable low power devices to operate within a Bluetooth Mesh network at reduced receiver cycles. Requires establishing friendship with a nearby node.
- **Friend feature** — it encompasses the ability to support low power feature nodes by establishing friendships and storing messages destined to those nodes, delivering them when the low power node requests for them.

Due to how messages are shared within the network, nodes are required to be constantly listening on the advertisement channels. The low power feature is important in enabling low power devices, by making them be able to put the Bluetooth module to sleep when not in use and ignore messages not addressed to them, saving energy, but still allowing them to send messages and receive them through specific nodes with the friend feature enabled. Nodes with the friend feature

will store messages for nodes with the low power feature enabled, delivering messages only when requested by the low power node.

To better understand how a Bluetooth Mesh works, some concepts need to be introduced as defined in Bluetooth *Mesh Profile* specifications[34]:

- **Models** — Models define the basic functionality of a Bluetooth Mesh node. A node can be composed of several models. For this dissertation, two types of models were addressed: client and server models which when paired together in different nodes enable Bluetooth Mesh applications. Client models define messages in order to set or read the status of states in the corresponding server model. Server models define a collection of states, state transitions and messages the node containing it can send or receive, also defining behaviours related to these collections. Bluetooth Mesh specification already contains some models, called *generic models*. By writing applications with these models, a developer can guarantee interoperability between devices from different manufacturers as long as they also implement these models. *Vendor models* are the other type of models allowed, with the main difference being that there are no vendor models defined in the specification, their definitions are left entirely to the developer and can be made to fit whatever solution is needed.

- **Messages** — There are three types of messages for communication between models which are GET, SET and STATUS. GET messages are used to request the values of a given state. SET aims to trigger state transitions in server models. SET messages can be unacknowledged, meaning they don't require a reply, or acknowledged. STATUS messages contain state values. They can be sent as replies for GET messages, acknowledged SET messages or sent without being requested, usually triggered by a periodic timer for publishing states. The type and purpose of a message can be identified by the included operation code (opcode) in the messages header. Message operation codes are usually defined in sets with a GET, acknowledged SET, unacknowledged SET and STATUS. Models can define several of these sets, each for a specific functionality.

- **Elements** — An element can be described as a constituent part of a node which can be independently controlled. When provisioned, elements in a node are assigned their own unique address in the network, which can be used as target for unicast messages. An element can contain a single or several unique models. Models are unique between each other when there is no overlap of opcodes within the models supported messages.

As Bluetooth Mesh was made to enable IoT devices, some parallels can be drawn with protocols created for the same purpose, namely *Thread*[52] and *Zigbee*[30]. Both run on top of the *IEEE 802.15.4* standard and are widely used in home and industrial automation networks. Table 2.1 compares some of the protocols characteristics.

These *IEEE 802.15.4* protocols build mesh networks relying on routing tables to route packets between devices. This approach differs from the described Bluetooth Mesh flooding by forcing message relaying through a single route. While this increases network complexity, it results in

Table 2.1: Protocols comparison[52][30][34][42].

Protocol	Radio Band	Network Topology	Data Rate	Native IP communication	Partial feature devices
Bluetooth Mesh	2.4GHz	Mesh w/ routing	2Mb/s	No	Yes
Zigbee	2.4GHz	Mesh w/ routing	250kb/s	Possible (Zigbee IP – IPv6)	Yes
Thread	2.4GHz	Mesh flooding	250kb/s	Yes	Yes

more efficient communication within the network, with less wasted packets. *Bluetooth SIG* however claims that by using TTL in messages and the mentioned optional features support in nodes, networks can be created where packet waste is negligible and won't hurt the network's and node devices performance[16].

2.2.5 MQTT

MQTT is a light, open messaging protocol following the publisher-subscriber paradigm[41][55]. MQTT falls under the client-server model for distributed application, where clients need to interact with a server, the MQTT broker, in order to publish and subscribe to topics. A client subscribing to a topic means the client is interested in receiving messages published to the subscribed topic. The broker is responsible for tracking clients and their subscriptions, notify them and deliver published messages.

MQTT was primarily designed to work over TCP/IP, however it can be implemented over any network protocol that can provide ordered, lossless and bi-directional connections[43]. MQTT has support from several platforms, capable of running locally or in the cloud and has seen implementations for most of the devices that can support it making it easy to integrate in new projects[19][18].

2.3 Hardware Platforms

This section will focus on the ESP family of products, specifically the ESP32, as it provides, at the time of writing, the cheapest available development platform with support for the technologies and development tools needed to implement a Bluetooth Mesh network with remote access.

ESP32 developed by *Espressif Systems*, is a series of microcontrollers with support for wireless communication, through Wi-Fi and Bluetooth 4.2, and include several common peripherals for interacting with the world around them[4]. Their low price and features made them quickly become widely adopted for small projects with over eleven thousand public repositories on GitHub alone[9]. Due to their success, development kits are easily available for purchase and development tools are well documented with guides and examples[7]. *Espressif Systems* also provides

the *Espressif IoT Development Framework* (ESP-IDF) which exposes application programming interfaces (API) to facilitate development on the platform. This API covers the networking capabilities of the device, WI-FI and Bluetooth, peripherals and includes support for common IoT protocols. Development is also possible trough the Arduino development platform which is built on top of ESP-IDF, offering a more familiar API for developers already used to said platform[1]. The solution developed in this dissertation will be relying directly in ESP-IDF as it contains a Bluetooth Mesh stack, not present currently in Arduino, which is built on top of Zephyr's, an open source real time operating system (RTOS)[17][5].

ESP-IDF applications run on a modified version of FreeRTOS, another open source RTOS for embedded systems[6].

Another candidate considered for this dissertation's solution was the nRF52 series of systems on a chip (SoC)[50]. These microcontrollers also provide a Bluetooth Mesh stack implementation by their vendor *Nordic Semiconductor*. An overall look into some of the *Espressif Systems* and *Nordic Semiconductor* offerings can be seen in Table 2.2. As shown in the table, nRF52 chips lack Wi-Fi, meaning that for enabling remote access to the Bluetooth Mesh network, a second platform would need to be considered in order to provide a gateway between Wi-Fi networks and the Bluetooth Mesh. This disadvantage is added on top of the superior ESP32 performance and greater cost of these chips and their respective supplied development kits[8][21].

Table 2.2: Hardware platforms features comparison[38][22].

SoC	**ESP32**	**nRF52832**
Processor	Xtensa dual-core 32-bit LX6@80/160/240MHz	32-bit ARM Cortex-M4F@64MHz
RAM	520 kB	64 kB
Bluetooth	4.2	5
BLE	Yes	Yes
802.11 (Wi-Fi)	Yes	No
802.15.4	No	Yes
General-purpose I/O (GPIO)	34, Configurable	32, Configurable
Other digital I/O	SPI, I^2C, I^2S, UART, IR	SPI, I^2S, 2-Wire, UART, PDM, QDEC
Peripherals	ADC, Motor and LED PWM, Hall sensor, touch sensor, Ethernet MAC interface, timer, counter	ADC, RNG, Temperature sensor, general comparator, low power comparator, timer, counter

Chapter 3

Proposed Solution

Chapter 3 is divided in three sections. The requirements the solution must comply with for it to fulfil the objectives defined in Section 1.3; proposed network architecture, solution deployment and choices made taking into account the defined requirements, and the researched software and hardware devices, presented in Chapter 2; The last section is dedicated to the software architecture for the code running in the network nodes.

3.1 Requirements

The list of requirements presented in this section will serve as the foundation for the proposed architecture. As the requirements written were created to meet the needs derived from problem addressed in this dissertation, the validity of the found solution can be confirmed by gauging its performance against the defined requirements.

3.1.1 Node Hardware Requirements

List of hardware and technology requirements the individual nodes in a home network must adhere to:

- node devices must have general general-purpose input/output (GPIO) pins or support peripherals to extend device functionalities by, for example, connecting sensors or actuators;
- node devices must be able to run home automation applications;
- node devices must support BLE 4.2 version or higher.

3.1.2 Network requirements

Requirements for the group of nodes when making up a home network:

- all nodes in a home network must have at least one wireless technology in common;

- at least one node in the network must support Wi-Fi in order to enable remote access to the network.

3.1.3 Node software requirements

Requirements for software and tools used for developing for the chosen hardware platforms:

- node software development tools must be free with accessible documentation;
- node software development tools must provide an API for the chosen technologies.

3.1.4 Home automation platform requirements

The required features the home automation platform must provide in order to be included in the solution as are follows:

- platform must provide a remotely available user interface either through a web browser or dedicated applications;
- platform must be able to run in the cloud, in a different network from the home nodes;
- platform must support at least one communication protocol, common to the home nodes or the node responsible for communication outside the local home network.

3.2 Proposed Architecture

3.2.1 Node network and home automation platform integration

As mentioned in Chapter 1, the focus for this dissertation is Bluetooth Mesh networks. The Bluetooth Mesh networks should enable devices to communicate between themselves inside the network, making up home automation applications. As mentioned in 2.2, Bluetooth Mesh gives way for complex network, spanning a large amount of area, due to message relaying within the Mesh, while also allowing for low power solutions through features aimed at low-power devices. Through Bluetooth Mesh, devices can address each other or groups of nodes and send messages directly, while, through the paths established in the Mesh, guaranteeing that communication always follows the lowest latency path.

Network management, including node provisioning and configuration, is left to the already implemented provisioning and configuration models. The developed ESP32 node applications will have included a provisioning and configuration generic server models, which, as the generic part of the name indicates, are part of the Bluetooth Mesh specification. Because these models are already implemented in the ESP-IDF stack they will no be explored in detail in this dissertation, which will only cover the necessary to set them up on implementation. The provisioning and configuration will be done through the *nRF Mesh* application running on an Android device

with Bluetooth 4.2 and BLE support. This open source application has provisioning and configuration clients implemented, which can communicate with thee respective server models running in a node, with a user interface to enable network management. The configuration models enable binding application keys to other models and manage their address subscriptions and message publishing context, which includes properties like publishing address and period between published messages.

Home Assistant, the home automation platform chosen, will be used to supervise and remotely control *Bluetooth* mesh networks from a browser. This is possible through the MQTT integration. An MQTT broker is required to be installed and running on the same machine or remotely to allow Home Assistant's integration to connect to it and participate in message exchanges as a client. By reading specifically formatted messages, the platform's MQTT integration is able to setup new devices in *Home Assistant*, assert their status and send command messages in order to change the device's status. For seamless integration with Home Assistant, the reported devices features during setup must comply with supported components included in Home Assistant. It is possible for a developer to expand Home Assistant's component library however this was not deemed necessary, as the current components cover the projected devices functionalities, and will not be covered in this dissertation. MQTT's integration message formats and supported components can be found on Home Assistant's MQTT integration online documentation[20].

Since a Bluetooth Mesh network is limited to communication between devices in that same network, there is no way for a device participating exclusively in it to communicate with a remote server. A simple solution to this problem would be to enable network nodes to also participate in a Wi-Fi network. This solution however suffers from some drawbacks:

- **BLE and Wi-FI requirement** — Devices would be required to include both these software stacks in their applications, leading to worse performance, and limiting possible solutions.

- **Crowded 2.4GHz ISM band** — Having both these technologies enabled in every device would further crowd the 2.4GHz band which could lead to worse network performance through lower speeds and packet loss.

- **Limited Wi-Fi range** — For a device to be accessible remotely it would need to be in range of a Wi-Fi access point, wasting the mesh network extended range capabilities.

A compromise to this problem is to depend on gateways, running on devices with support for both technologies, to relay messages between Bluetooth Mesh network nodes and a remote server running the MQTT broker, through a Wi-Fi connection. This solution still adds some restrictions to the network. The gateway nodes must be near a Wi-Fi access point and Bluetooth Mesh nodes need to have a network path to those gateways in order to send and receive messages to and from outside the network. The solution covered in this dissertation will make use of a single gateway in a network to translate and relay supported messages.

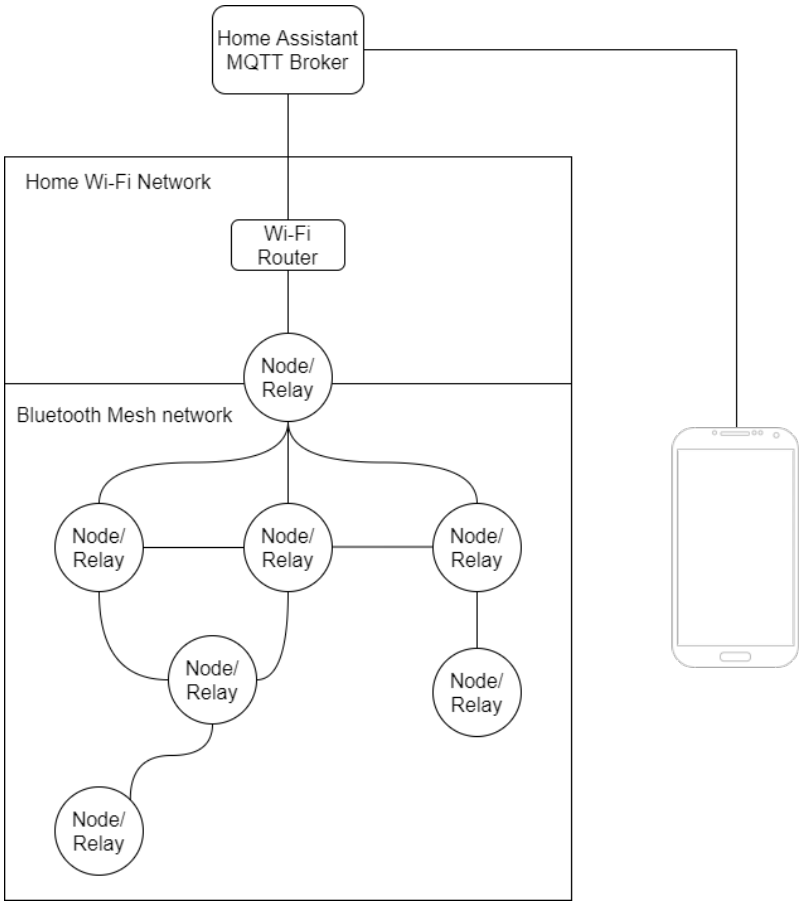

Figure 3.1: Proposed solution architecture

3.2.2 Node hardware

The purpose of the nodes is to enable home automation applications through communication between themselves and the cloud and using their interfaces to read values from or interact with the world around them and their users. An example of a home automation application, would be controlling a an actuator either through another node, with a sensor connected, or remotely through the home automation platform.

The device chosen for validating this solution, as alluded to in section 2.3 will be the ESP32, making use of *ESP-IDF* and its APIs, including *Bluetooth* Mesh and Wi-Fi APIs, for developing, compiling and flashing the resulting binaries on the target devices. Because ESP32 has support for both Wi-Fi and BLE technologies, it can deployed both as a normal node and a gateway, keeping all development in the same platform.

A representation of the proposed architecture can be seen in Figure 3.1.

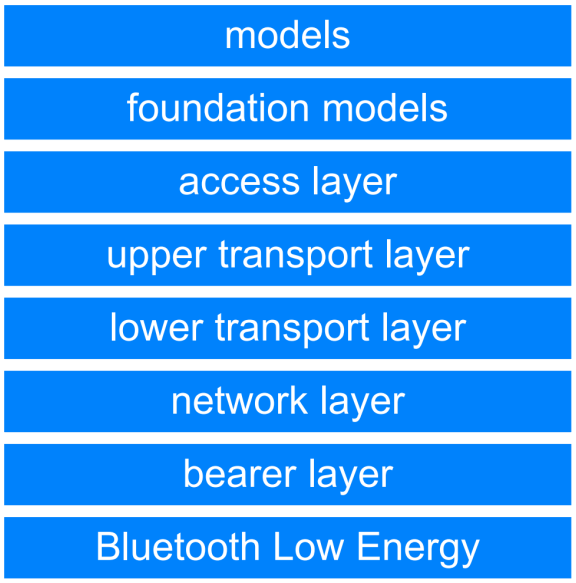

Figure 3.2: Bluetooth Mesh layers from the Bluetooth Mesh Networking - An Introduction for Developers[33]

3.3 Node Software Architecture

The software architecture proposed aims at providing a platform where a developer can include Bluetooth Mesh functionality in their projects and run it alongside existing or new code enabling wireless communication between nodes and control through a home automation platform, while avoiding having to deal the standards details.

While the Bluetooth Mesh architecture is composed of several layers, as shown in Figure 3.2, the focus for this dissertation will be on the model layer. All other layers are already implemented in ESP-IDF stack with a suitable API which this solution will rely on. The Model layer is where the implementation of things such as model behaviours, messages and states are defined.

Software architecture will revolve around software components which will integrate the Bluetooth Mesh model layer API. For this dissertation two types of components will be defined:

- **Bluetooth Mesh common component** — This is a single component that will need to be included in every network node. Its main purpose is to offer a common structure that other components can adhere to, manage initialization and handle message delivery to compatible components models.

- **Bluetooth Mesh element component** — These components define a Bluetooth Mesh element, and its models, that can be included into an application to be run in a node. Certain elements can also include support for relevant peripherals and I/O, streamlining their inclusion in projects. They contain API's to enable their functionalities and macros to further separate the developer from the Bluetooth Mesh details.

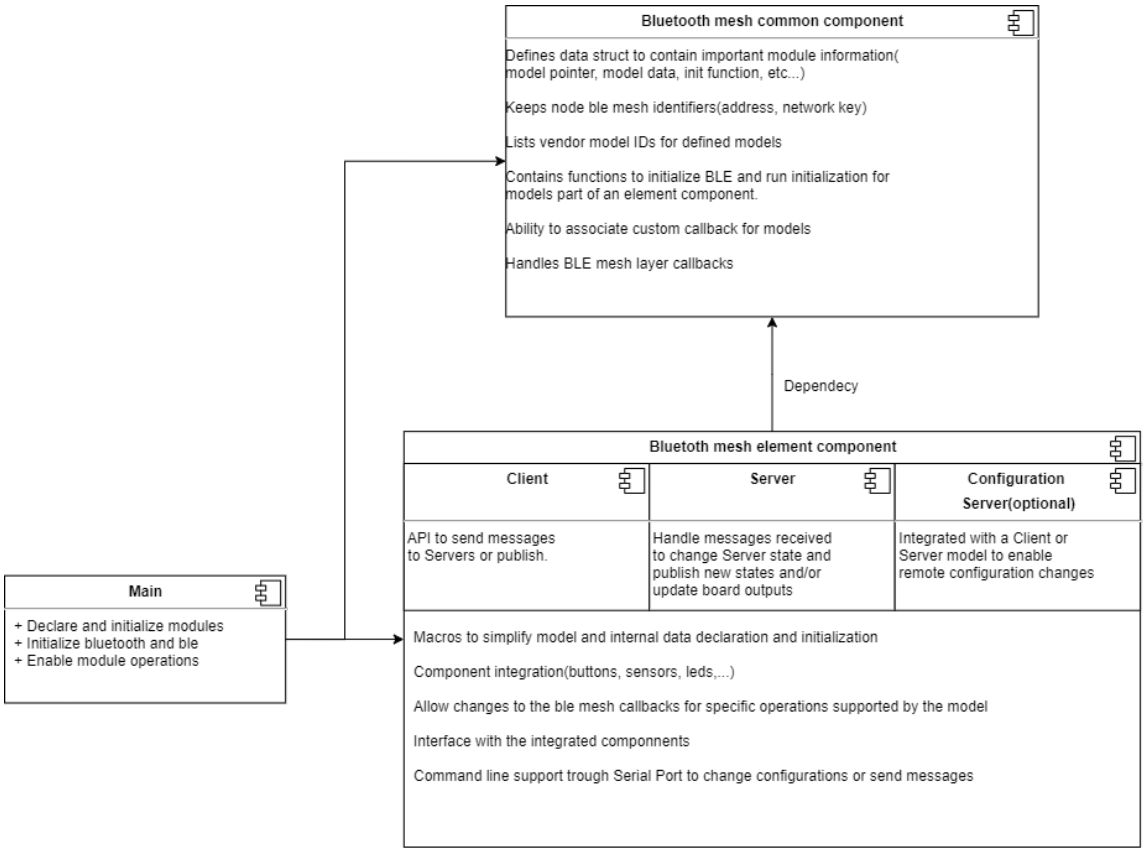

Figure 3.3: Bluetooth Mesh components

Figure 3.3 contains a more detailed feature list for the components with their expected relations.

By relying on this architecture with a common component to manage all other components, nodes should be able to include different element components and even include the same ones several times without worrying about conflicts between them.

The Bluetooth Mesh network and MQTT gateway will be implemented as a Bluetooth Mesh element component. The element will be comprised of a single client model which will relay supported STATUS messages published in the network under subscribed group addresses or the broadcast address to the MQTT broker after proper translation and similarly, translate messages from the MQTT broker into SET messages to send into the network.

A more detailed look into how the gateway component processes messages from new nodes or addresses and links them with Home Assistant components is presented in Chapter 4.

Chapter 4

Implementation

This chapter goes through implementation of the proposed architecture with an ESP32 development kit as Bluetooth mesh network nodes. The applications to be run on these were written for, and built with ESP-IDF Version 4.2.

The first part of this chapter is dedicated to how the components mentioned in Section 3.3, Bluetooth mesh element and Bluetooth mesh common components, can be written as ESP-IDF components, pieces of standalone code which can be compiled and linked into applications to be flashed onto the target devices, and their interactions. The second part covers the gateway component architecture and how it resolves translations between MQTT and Bluetooth mesh network messages.

ESP-IDF components may be written in *C* or *C++* languages. For this dissertation all code was written in C.

Due to message delivery limitations within the ESP-IDF Bluetooth mesh API, where generic model messages are not delivered to vendor models, and an apparent lack of support for creating models extending generic models, all models will be implemented as vendor models. This choice was done in order to facilitate the gateway component. A gateway is required to handle messages with opcodes from several different models and without the possibility to extend generic models it is not possible to create a single model to handle these messages. However, a vendor model can be written to handle as much different messages as needed.

4.1 ESP-IDF Bluetooth Mesh Abstractions and Structures

Within ESP-IDF's Bluetooth mesh stack there are structures that abstract Bluetooth mesh concepts already introduced in Sections 2.2 and 3.2, such as models and elements, which will be required to be incorporated into the ESP-IDF components that will be written.

Besides Bluetooth mesh concepts this section will also cover other structures that need to be handled by the written components for configuration data to be used by the Bluetooth mesh stack.

While some of these structures contain a lot of data, some of that data is initialized and operated exclusively within ESP-IDF's Bluetooth mesh stack and will not be covered in this dissertation.

Structures for model definition:

- **esp_ble_mesh_model_t** — Abstraction describing a Bluetooth mesh model instance;
- **esp_ble_mesh_client_op_pair_t** — To be used in an array containing supported GET and SET message opcodes and their corresponding STATUS message opcode. Only required in client models. Refer to section 3.3 for message opcodes descriptions;
- **esp_ble_mesh_model_op_t** — Abstraction describing model operation context for models. Holds supported incoming message opcodes and their expected minimum length;
- **esp_ble_mesh_client_t** — Client model user data context. Replacement for model user data enforced by ESP-IDF stack in order to handle message publishing and acknowledgment in client models within the Bluetooth mesh stack implemented in ESP-IDF;

Details for data contained within the *esp_ble_mesh_model_t* structure can be found in Table A.1.

Besides model structures, four additional structures require some attention:

- **esp_ble_mesh_cfg_srv_t** — Configuration server model context. An ESP-IDF structure to use with the provided macro *ESP_BLE_MESH_MODEL_CFG_SRV* in order to initialize a configuration server model. On initialization, it is used to set node supported Bluetooth mesh features and configure node specific settings;
- **esp_ble_mesh_prov_t** — Used to define provisioning properties and capabilities.
- **esp_ble_mesh_elem_t** — Abstraction describing a Bluetooth mesh element instance. Holds arrays of models separated by generic models and vendor models paired with their own size information.
- **esp_ble_mesh_comp_t** — Node composition data context. Holds array of elements, their size and internal node identifiers. Usually implemented as a global variable and serves as a point of access for functions to all models in the node.

4.2 Configuration Server Model

The server model, resulting from the *ESP_BLE_MESH_MODEL_CFG_SRV* macro, has its structures and functions defined and implemented in the *ESP-IDF* Bluetooth mesh stack. It handles provisioning requests in order to enable the node to take part in a Bluetooth Mesh network, and models configuration, such as assigning application keys to models and configuring their message publishing, which can be user controlled through the *nRF Mesh* app when acting as the networks

provisioner, through its own configuration client model. While this model is critical to a Bluetooth Mesh node, because all its functionalities are implemented within ESP-IDF, it will not be covered in much detail. Only the attributes related to node features and message transmission are relevant. These attributes are presented in table 4.1.

Table 4.1: *ESP_BLE_MESH_MODEL_CFG_SRV* structure details

Type	Name	Description
uint8_t	net_transmit	Holds amount of retransmissions and delay between them for messages originating from the node.
uint8_t	relay_retransmit	Holds amount of retransmissions and delay between them for message relaying.
uint8_t	default_ttl	Default TTL for messages originating from the node.
uint8_t	relay beacon friend_state	Set state for respective Bluetooth mesh feature described in section 2.2.4. Can be set to enable, disable or not supported.

4.3 Common Component

As described before in Section 3.3, the common component was implemented in order to manage element components. For this it introduces the structure *model_aux_info_t* to be included in implemented element components. Each implemented model will require this structure to be initialized and pointed at by *user_data* in the *esp_ble_mesh_model_t*. A description of this structure's composition can be found in Table 4.2 . It also offers APIs to aid initializing the ESP32 Bluetooth mesh module along with the element components.

Table 4.2: model_aux_info_t structure

Type	Name	Description
esp_ble_mesh_client_t*	client_data	Pointer to struct to be initialized and utilized by the ESP-IDF Bluetooth mesh stack.
esp_ble_mesh_model_t*	model	Pointer to model struct to which this model_aux_info_t belongs to. Null if corresponding model is not a client.
void*	init_func	Pointer to corresponding model initialization function. Can be null if no initialization is required.
void*	cb_func	Pointer to model's specific message events handler. Can be null if the model doesn't receive messages.
void*	model_data	Points to model specific user data.
model_aux_info_t*	neighbour_model	pointer to the next model_aux_info_t in the element component. Null if there none.

4.3.1 Initialization

The Bluetooth mesh module initialization amounts to:

- enabling provisioning for the device, making it advertise its availability to be provisioned into a new network;
- setting up function callbacks to be called after events triggered by Bluetooth mesh messages. Message delivery events will be covered in Section 4.3.2;
- initialize the relevant node composition data to be used by the upper layers of the BLE mesh stack implemented within ESP-IDF.

All of these functions are implemented within ESP-IDF and as such, will not be covered in detail in this dissertation.

Element component initialization tries to look up for compatible models within the elements, by checking if the model's vendor company id matches one of the supported IDs and running the initialization function pointed within *model_aux_info_t*. To make sure this process does not fail all element components must comply to the components interface as shown in Figure 4.1.

The sequence diagram in Figure 4.2 can be used to ascertain the order of events and the layers involved in the nodes initialization.

The functions provided in the common element to initiate these events are listed and described in Table 4.3

Table 4.3: Common element initialization API

Function	**Arguments**	**Return**	**Description**
esp_ble_mesh_init	esp_ble_mesh_prov_t esp_ble_mesh_comp_t	esp_err_t	Initializes provisioning capabilities and composition data. Returns ESP_OK is successful
ble_mesh_model_init		esp_err_t	Initializes models in compatible element components. Returns ESP_OK is successful.

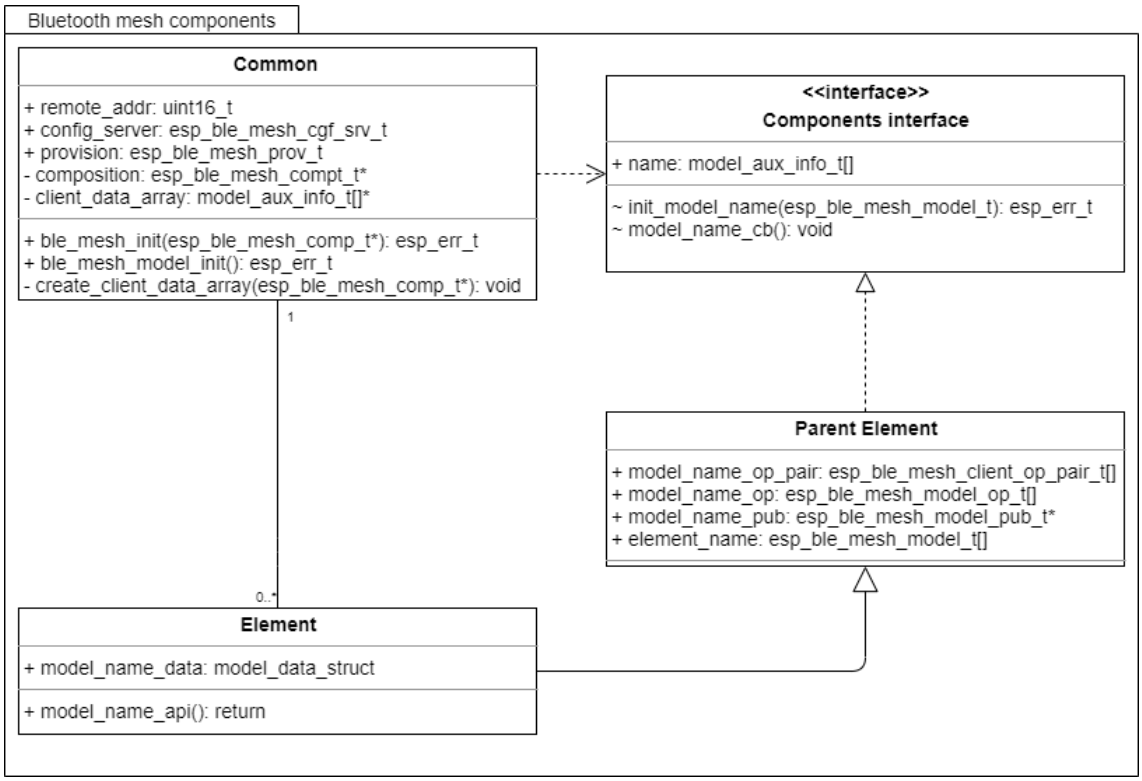

Figure 4.1: Components class diagram

4.3.2 Message delivery to models

As mentioned in the previous section, one of the initialization steps is setting up the callback functions for events involving messages. These events are triggered in the upper layers of the ESP-IDF Bluetooth Mesh stack by either received or sent messages.

For the model layer, which is being implemented in this dissertation, to be notified of these events it must register a function in the stack to be called when these happen. ESP-IDF provides different callbacks for different puporses from which only these there will be covered in this implementation:

- **Provisioning callback** — Reports events and information involved in device provisioning. Since provisioning is already implemented by the ESP-IDF stack, this callback is only used for displaying provisioning events and its state;
- **Configuration server callback** — It is in a similar position as the provisioning callback but for node configuration events.
- **Custom model callback** — A single callback that reports on any message event related to vendor models for either client or server sending, receiving and publishing messages.

While the first two callbacks are only used to report on events happening within the ESP-IDF Bluetooth Mesh stack and could be completely removed from a node without affecting its

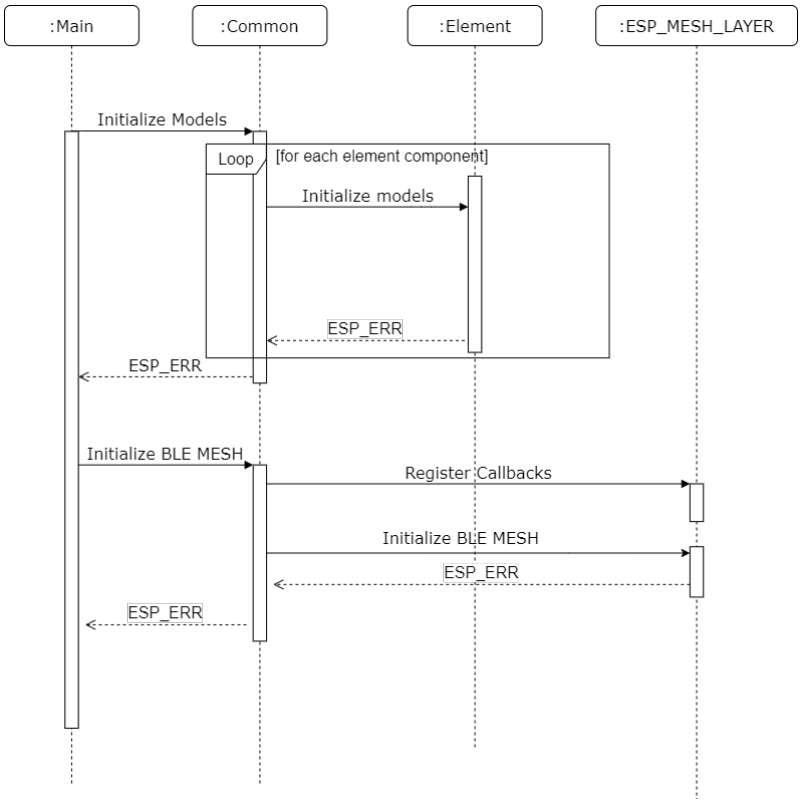

Figure 4.2: Bluetooth mesh initialization message sequence

functionality, the last callback, *custom model* callback, is essential for the node's function and has to be implemented in order to effectively deliver messages passed over from the ESP-IDF stack into the models initialized in the node.

The ESP-IDF stack passes two arguments to the custom model callback when it is called, the event's type and the event's parameters. Different events have different parameters so the callback function must be written to discern events and handle the parameters argument according to the event.

The parameters will usually include a pointer to the model for which the message is to be delivered to or originated from, information about the message, such as its opcode, length and the address from where it came from and error codes when a message fails to be sent. The presence of these parameters varies with the event. The parameters for the message received event, as an example, can be found in Table 4.4.

A single message can only trigger an event once. For messages sent to unicast addresses this poses no problem as the target model for the message is included in the parameters. However for group and virtual addresses this behaviour puts the responsibility of delivering messages to all models subscribed to the address into the callback function. To find target models, it iterates through all node's models looking for the address received in their group array.

Table 4.4: Parameters for message receiving event

Type	Name	Description
uint32_t	opcode	Received message opcode
esp_ble_mesh_model_t*	model	Pointer to the model which will receive this message. For non-unicast addresses, model pointed to is the first one in device node composition with support for opcode received
esp_ble_mesh_msg_ctx_t*	ctx	Pointer to message context. Contains data such as message destination and source address
uint16_t	length	length of the received message
uint8_t*	msg	Value of the received message

To distinguish address types, between unicast, group, virtual and broadcast, a simple comparison is needed. The address space is divided as seen in Table 4.5.

Table 4.5: Bluetooth mesh address table

type	range
Unicast	< 0x8000
Virtual Address	>= 0x8000 < 0xC000
Groups	>= 0xC000 < 0xFF00
Broadcast	= 0xFFFF

Passing messages from the common component to the models relies on element components having properly implemented the *model callback handler* from the interface shown in Figure's 3.3 class diagram. A pointer to the callback handler must be available in the model's *model_aux_info_t* which itself should be pointed at by the user data pointer in the model's structure.

A summarized view of the message delivery sequence can be seen in Figure 4.3.

4.4 Element Component

As mentioned in Section 3.3, these are the components where Bluetooth mesh models are to be implemented and APIs provided in order to enable sending messages and deal with received messages.

For better understanding, this section is divided in four parts: component definition and initialization, and Component APIs and integration with the ESP-IDF Bluetooth mesh stack.

4.4.1 Component definition and initialization

As stated, a component can be made up of several models, each model requiring defining the structures presented for model definition in Section 4.1. All these structure's definitions depend only on the model's purpose.

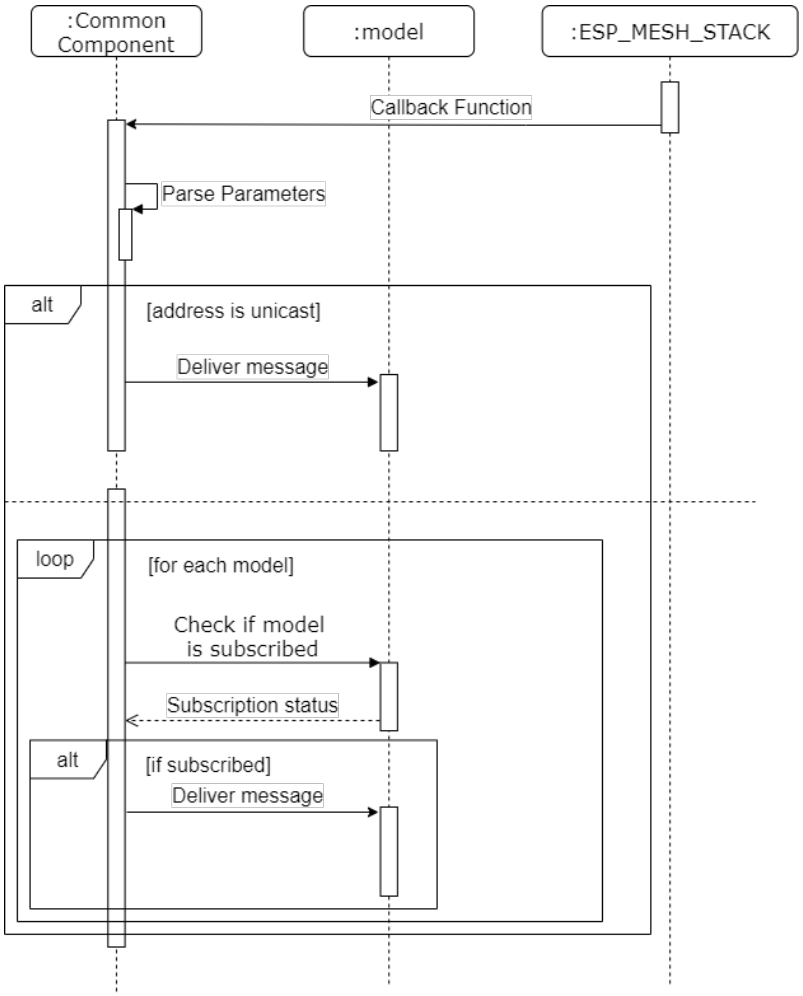

Figure 4.3: Common element message delivery sequence diagram

To avoid having the developer deal with these structures and their repetitiveness, the definitions and initialization will be included in the element component's header as C macros. With these macros, whole components and their data structures can be initialized and referred to by the name used in the macro, which will point to an initialized *model_aux_info_t* struct. The name used in this macro is what will identify the component when using the available APIs.

In the main section of the code, after using the macros to initialize all needed element components, the resulting models from this macro need to be added to the node's composition. For this each component will also contain a macro to extract the model pointers from the given name, which will be organized in an *esp_ble_mesh_elem_t* array with the help of a macro present in ESP-IDF's Bluetooth Mesh stack *ESP_BLE_MESH_ELEMENT*. This array can then be linked to the *esp_ble_mesh_comp_t* which will be used for the initialization mentioned in Section 4.3.1.

4.4.2 Sending messages between models

These components biggest purpose is to enable message trading with nodes in the same network. In this solution three functions provided by the ESP-IDF API were utilized in this implementation. These functions cover sending and publishing messages from client models and sending messages from server models. For a model to be able to send messages, the node must have been successfully provisioned into a network, and an application key must be bound to the model.

The biggest difference between sending and publishing messages in ESP-IDF's Bluetooth mesh stack, is that sending messages is a single event which sends the message once, the message can then be retransmitted shortly after, according to the *net_transmit* parameter set in the configuration model, mentioned in Table 4.1.

Publishing messages sends messages according to the publishing configuration for the respective model which is held in the pub parameter within the model's structure.A.1 The publishing configuration can be changed remotely through the configuration server model. Table 4.6 contains the relevant arguments from the send message function as an example. The arguments for publishing are similar except no message context is needed as it is already defined in the model.

Table 4.6: Arguments required for sending a message trough ESP-IDF's Bluetooth mesh API

Type	Name	Description
esp_ble_mesh_model_t*	model	Pointer to the model which will receive this message. For non-unicast addresses, model pointed to is the first one in device node composition with support for opcode received
esp_ble_mesh_msg_ctx_t*	ctx	Pointer to message context. Contains data such as target address, the message TTL, and application and network keys
uint32_t	opcode	opcode of the message to be sent
uint16_t	length	length of the message to be sent
uint8_t*	msg	Value of the message to be sent

To standardize communication between client and server models serving the same application, two types of structures need to be specified:

- **SET structures** — SET data structures are to be carried in SET messages sent by client models. They carry data to be received and processed by a server model which might cause the model's state to change;
- **STATUS structures** — STATUS data structures are to be carried by STATUS messages. These can be sent as messages by servers, to report their state, or published by clients with the same intent.

A SET or STATUS data structure can be associated with multiple SET and STATUS opcodes within the same model. However the opposite is not allowed, as opcodes serve to identify not only the operation requested from the model which received the message, but also the content structure of the message.

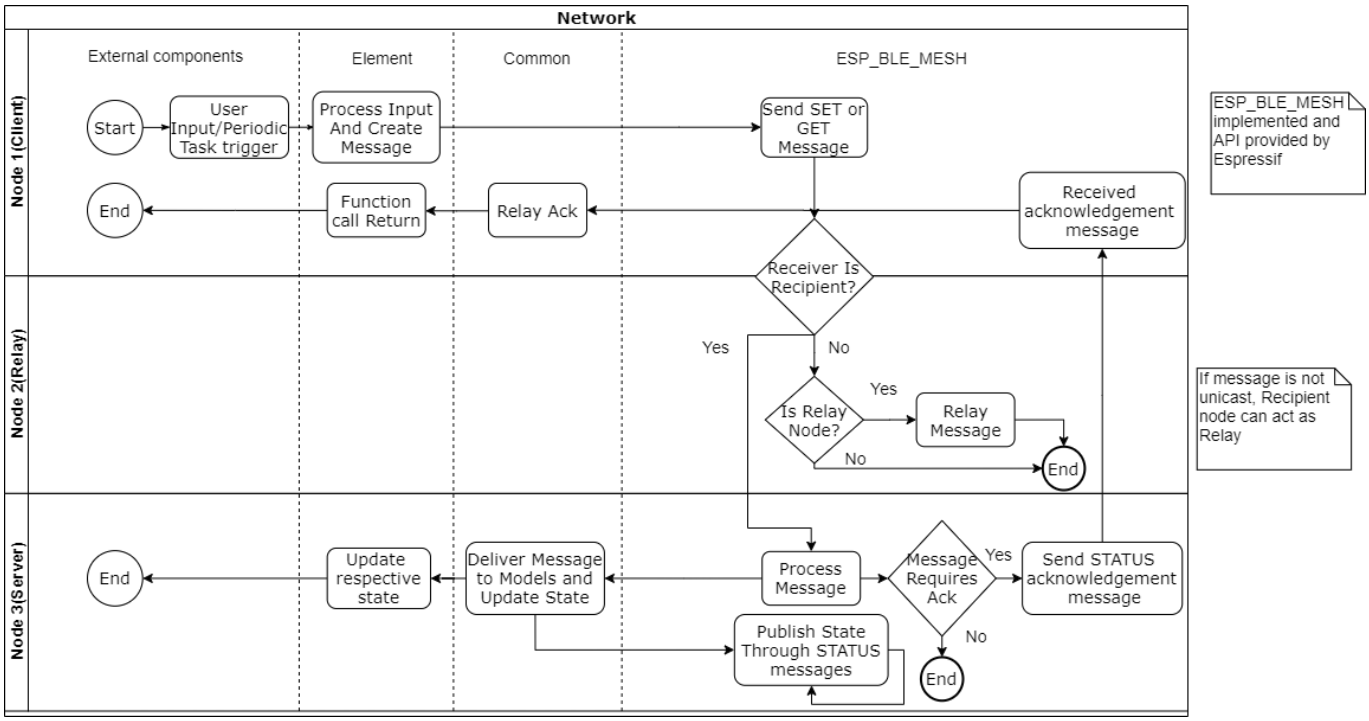

Figure 4.4: Client-server models interaction

The APIs provided by the components to send messages will be heavily reliant on the element component's features. As a baseline, the element components should have a function for each SET opcode which can be called with an initialized corresponding SET structure as an argument.

4.4.3 Receiving model's messages

The method by which a message is delivered to a model is already covered in section 4.3.2. This section focuses on how models deal with the delivered message and how developer can tap into this messages through the element component's API.

Similar to how ESP-IDF allows binding functions to events in its Bluetooth mesh stack, element components should also offer an API to register functions to these events as an extension of the callback function already implemented inside the element component.

Figure 4.4 represent the expected messaging behaviour as described in the previous sections.

4.4.4 ESP-IDF component integration

It is encouraged for element components to be integrated with other ESP-IDF components which cover its use cases as it can further simplify their integration in a project. By integrating other components, developers can rely on element components to interface with the device's GPIO and peripherals by themselves only requiring an initial setup and offering its state information through

the component's API, allowing also to setup helpful tasks to report, for example, sensor readings internally and to the network.

4.5 Developed Bluetooth Mesh Application

Besides the gateway, two other element components were written in order to validate the solution presented in this dissertation. An element component integrated with an IoT Button component, from *Espressif Systems* provided components[3], which is able to communicate with the other written component which outputs to a pin, in the device's GPIO, its state, either on or off, making up an On/Off application.

Each of these element components require defining a single model for each, a client model for the first component and a server model for the second one, with a single set of SET, GET and STATUS messages to communicate between themselves. Since these will be vendor models, the opcodes used are arbitrary and just need to be different from each other. Vendor opcodes are identified by their respective vendor ID and unique ID within the same vendor. To declare vendor opcode's the macro *BLE_MESH_MODEL_OP_3(b0, cid)* was used, where 'b0' is the unique opcode's ID and 'cid' the vendor ID, which for this proof of concept will be Espressif Incorporated's ID, 0x02E5[2].

Table 4.7: On/Off application message set

Opcode	Type	Content	Return opcode
0x01	GET	empty	0x04
0x02	SET(Acknowledged)	uint8_t new_value	0x04
0x03	SET(Unacknowledged)	uint8_t new_value	0x04
0x04	STATUS	uint8_t current_state	

Table 4.7 shows the simple set of messages used for this application and their SET and STATUS payloads.

In this application the client model sends SET messages with the value the server model will update to. To trigger this message the *IoT button* component is used. This button can be set up to read from an input pin and trigger events on button presses or releases, calling the function which will send a SET message into the Bluetooth Mesh network, the value carried in this message will be different from the known server's model state value. The API provided by the element component in order to setup this behaviour and enable developer to directly send messages is described in Table 4.8. Similarly an API is provided in the element component with the server model, in order to bind a port to it which, when connected to an LED, can reveal the model's state.

All messages sent are addressed to the model's first subscribed group address which can be set through the nRF Mesh application.

Table 4.8: On/Off client element component API

Function	Arguments	Return	Description
on_off_button_setup	uint8_t	esp_err_t	Initializes IoT button component to send new SET messages on button press.
get_current_state		uint8_t	Returns server model known state.
send_set_message	uint8_t	esp_err_t	Send new SET message carrying the value in argument.

4.6 Gateway

As discussed in Section 3.3, the gateway between the Bluetooth mesh network and the home automation platform, through an MQTT broker, will be implemented as a Bluetooth mesh component to be included in a normal network node.

The gateway component integrates ESP-IDF components for interfacing with the Wi-Fi module and enable support for sending and receiving MQTT messages. This component will rely on a single client model to receive Bluetooth mesh messages to be translated into MQTT and vice-versa, sending Bluetooth mesh messages into the network translated from MQTT, and a configuration server model which will be used to remotely enable Wi-Fi and MQTT in the node, through the nRF Mesh application.

4.6.1 Home Assistant MQTT devices

MQTT devices in Home Assistant have three properties which are divided as MQTT topics:

- **Configuration Topic** — configuration topic is what enables a device to be discovered by the MQTT integration and added as a Home Assistant device. This is the first topic to which the gateway will send a message when a STATUS with no precedent is received. The message will contain a name for the device, which will be the address to which the message is addressed plus the corresponding model identifier, the name of the corresponding device type from the available MQTT devices and the topic locations to where STATUS messages will be sent with another where SET messages will be read from.
- **State topic** — the topic to which the gateway will report the device's state obtained from STATUS messages shared within the network.
- **Command topic** — topic which reflects user interaction within Home Assistant. The gateway must be subscribed to these topics in order to receive its messages, translating them into Bluetooth mesh SET messages and diffusing them in the network.

4.6.2 Adding specific support for Bluetooth mesh applications

The approach to add support for specific Bluetooth mesh applications to the gateway, was to have a client model to mimic the SET and STATUS messages from the respective applications

client models. This matches well with the MQTT integration as STATUS can usually be directly translated to a state topic message, and SET messages can be easily derived from command topic messages. For each pair of STATUS and SET messages a table can be built which contains all required information for translation. Table 4.9 shows an example for a light with brightness control application. The entries on this table also have to be associated with the respective STATUS and SET opcode and model id.

Table 4.9: Translation table example for light control gateway entry

STATUS struct	**Topic names**	**Command names**	**SET struct**	**Translation type**	**Device name**
uint8_t state **uint8_t** brightness	"status" "brightness"	"status" "brightness"	**uint8_t** state **uint8_t** brightness	onoff direct	"light"

In this specific example the STATUS and SET structures can be the same. The translation type is also introduced for values which have different representations within Home Assistant and the Bluetooth mesh model, such as in the case of the light state. In the Bluetooth mesh model, light state is either "1" or "0" while for Home Assistant, the values have to be either "ON" or "OFF". Each of these types is associated with a function to convert these values, including direct translation, as they all at least need to be converted to, and from strings, when sending and and receiving MQTT messages respectively.

4.6.3 Gateway device management

To manage devices between the Home Assistant and the Bluetooth mesh network a list is used to keep track of all registered devices and their corresponding Bluetooth Mesh applications. Before the gateway can start adding new devices, after it is provisioned into the network, it is required to subscribe it to the group address being used for the application, and add also share the application's application key. This, like the other element components, is enabled through the nRF Mesh Android application and the configuration server model running on the node.

Entries in this list are distinguished by the pair of STATUS opcode and mesh group address. When a message without a registered pair arrives to the gateway it will trigger the function to append a new entry if the opcode for the message received has a corresponding entry in the translation table mentioned in 4.6.2. This entries contain the mentioned STATUS opcode, mesh group address, pointer to the corresponding translation table entry and also the application's application key, which will be required to send messages back to the nodes running the Bluetooth Mesh application.

The steps taken when registering a new device are represented in the flowchart 4.5.

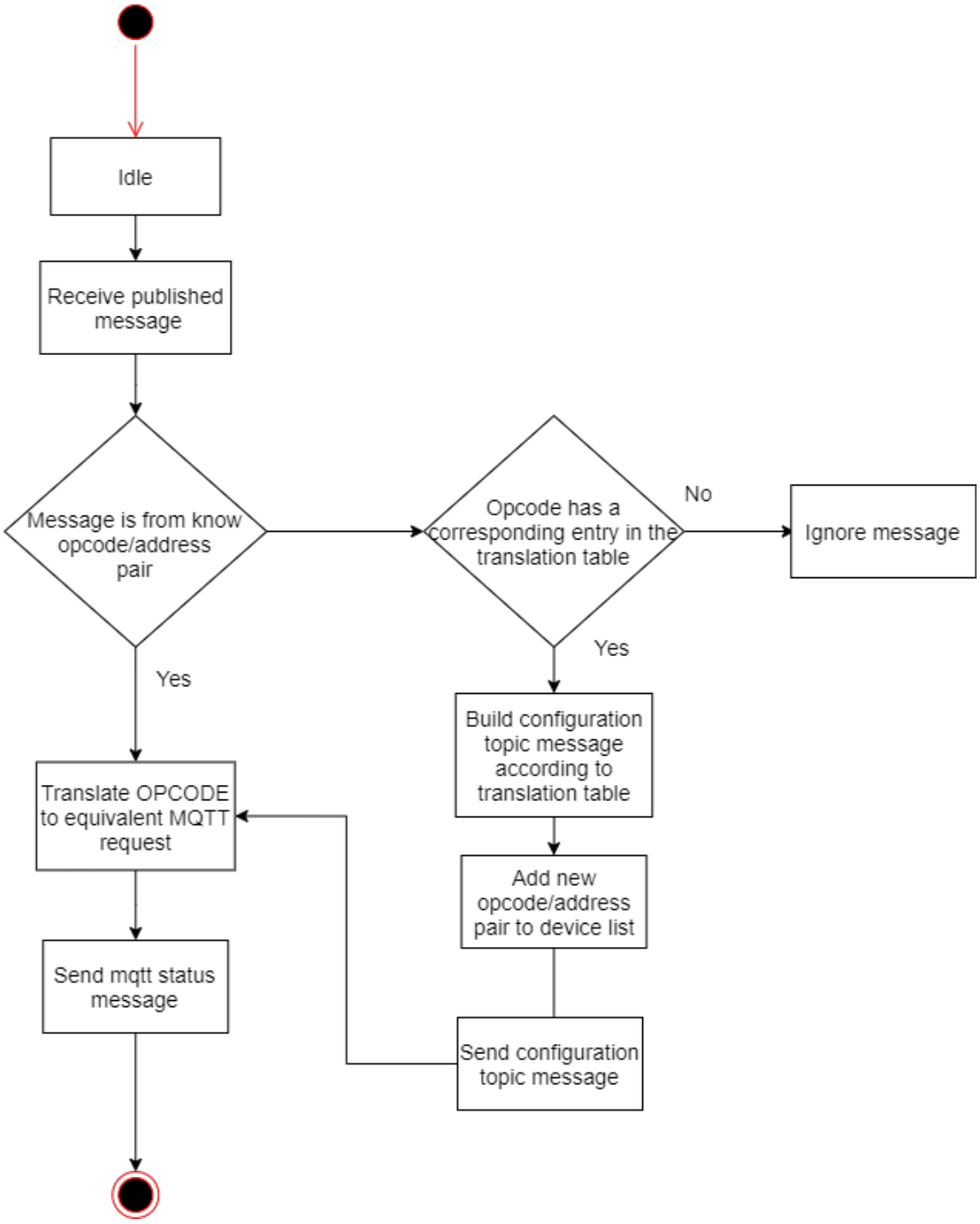

Figure 4.5: Bluetooth mesh gateway flowchart

Chapter 5

Tests and Validation

To validate and test this solution, besides de common element, three element components were written, as mentioned in the previous chapter. The On/Off application element components described in Section 4.5 and a gateway component, with a single entry to support the On/Off application. The first half of this chapter goes over tests done in order to validate the proposed solution, while the second half is dedicated to some network performance tests.

5.1 Validation

The solution was validated in two parts:

- **Bluetooth Mesh** — a network was setup with one On/Off client component node, one On/Off server component node both out of range from each other and another node in between without any components, just to serve as a message relay;
- **Home automation platform** — to the previous network, an extra node with the gateway component was added and setup to connect to a remove server running an MQTT broker and Home Assistant, with the Home Assistant's web page opened in a desktop browser.

This setup covers the minimum requirements presented in Section 3.1.

Figure 5.1 represents the nodes network position used for validation, with all messages requiring relaying in order to reach the other nodes, gateway included.

With the exception of the relay node, all other nodes shared the same application key for their models and were subscribed or publishing to the same group address using the nRF Mesh application running on an Android device.

Wi-Fi and MQTT broker connection details were included in the gateway node code, as the node did not possess a suitable interface to input the details.

The switch node was connected to a physical switch and code was included in order to send a message whenever the switch changed state. Similarly, an LED was connected to the client node, with code to update the LED state whenever a new message arrived.

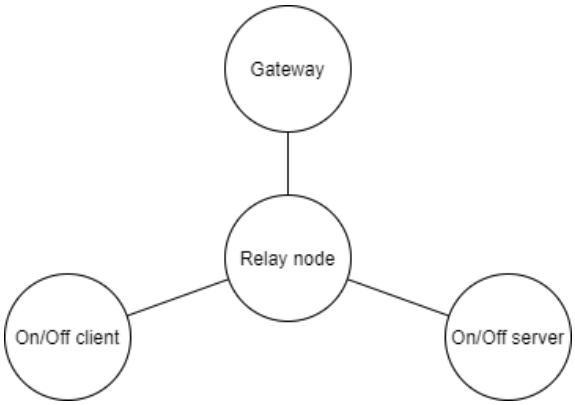

Figure 5.1: Validation network

After setting up and provisioning the nodes, the behaviour observed when interacting with the switch was as expected. In the first network, the LED on the node with a client model reflected the switches state with no noticeable delay, showing that direct communication between Bluetooth Mesh nodes is working correctly. Adding the gateway node made it so that after the first interaction, an UI element was created in Home Assistant which also reflected the switches state. Interacting with the element was also reflected correctly in the LED.

5.2 Performance

Testing was made in order to verify the network performance in a home environment, with walls and other obstacles between the nodes. These tests can serve as a reference for what to expect of a Bluetooth Mesh network in regards to packet loss, latency between nodes and how these metrics are affected by their distance, obstacles and network density.

To measure these metrics within the Bluetooth mesh network, two nodes were setup in a network to communicate with each other in a simple request reply configuration.

Measurement were made at the requesting node. *Package loss* represents number of requests that got a reply and not the number of messages the reply node received. *Delay*, is the time between sending a request and receiving a reply. Values were read at the request node through the node's log output via serial port.

The nodes were setup in five different tests, by moving the node sending the request messages into different parts of a house, with different distances between itself and the other nodes, as shown in Figure 5.2. Only tests 4 and 5 had the relay node included in the network. Each test included 200 message requests.

Through these tests results, seen in Table 5.1, it is possible to see how the Bluetooth Mesh network starts to fail at longer communication distances, with higher *Packet Loss*. Even when the nodes are close to each other some packet loss can be expected which is not surprising since Bluetooth Mesh by itself does not guarantee message delivery. *Latency* seems to be mostly affected by the addition of the relay node and not much with distance, since some messages are now only

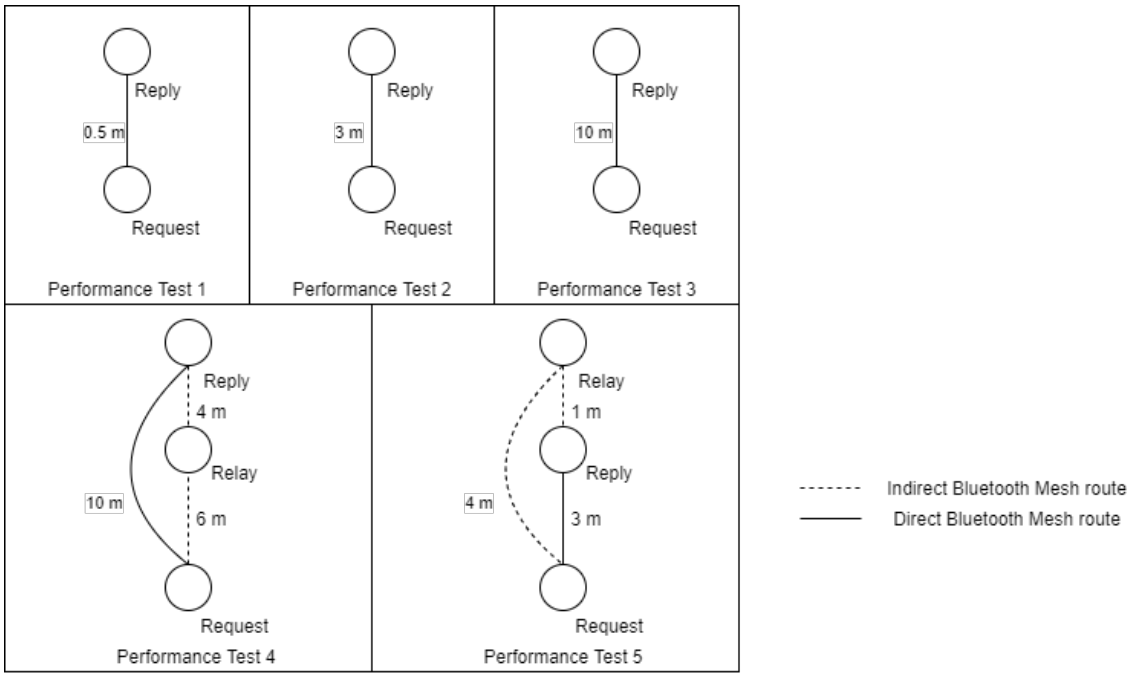

Figure 5.2: Performance Tests network

Table 5.1: Results from Bluetooth Mesh network test

Test Number	**Package Loss(%)**	**Average delay(us)**	**Min delay(us)**	**Max delay(us)**
1	10.5	71 053	31 395	198 939
2	11.5	76 497	32 424	193 574
3	53.5	93 259	36 724	199 118
4	1.5	138 470	28 817	230 335
5	0.5	73 082	27 555	210 449

being delivered after being routed through the relay which adds latency, when compared to a more direct route between the nodes. Although the relay node seems to add latency, it also increases the network reliability by greatly reducing *Packet Loss* as can be seen by comparing tests 2 and 3 with tests 5 and 4, respectively.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Bluetooth Mesh presents itself as an open solution for IoT devices to be integrated in home automation and industrial settings. The work done in this dissertation displays the ability for Bluetooth Mesh network to work in home environments and be included in already available and widespread devices.

The validation shows how the proposed architecture can make use of current Bluetooth Mesh implementations to enable home automation applications through including Bluetooth Mesh enabled components in device applications and integrate these applications with available home automation platforms.

The experimental tests done however also show this technology limitations when it comes to reliability. Since Bluetooth Mesh does not offer reliable communication channels, an effort must be done when developing applications for these applications to include message acknowledging and routines to compensate package loss when package loss is not tolerated. A dense enough network can make reliability a non problem, but developers cannot trust nodes to always be deployed in dense networks, with high enough reliability for the purpose they were built for.

6.2 Future Work

The Components developed in this dissertation were limited to the ones used for validation. Expanding the element components library and the gateway translation table will enable this solution to be used for a more varied number of home automation solutions.

The gateway component's MQTT can be made to work seamlessly with other home automation platforms besides the one covered, Home Assistant. The translation libraries created for this component can be replaced, or settings can be added to choose which home platform to target.

The current solution only allows for a single gateway in a Bluetooth Mesh network. Gateway coordination in order to allow coexistence of several gateways in the same network would enable

nodes that would be far away from the single gateway to rely on shorter TTL for its messages, diminishing network load, and reducing latency between the home automation platform and nodes.

In order to further decrease network load, performance measurement tools, or Bluetooth Mesh models, could be used in order to actively monitor and evaluate message activity between nodes and change the messages TTL, according to the needs of the node, or disable the relay feature according to the needs of the network.

Appendix A

Tables

A.1 C structures

Table A.1: esp_ble_mesh_model_t structure details

Type	Name	Description
uint16_t	vnd.company_id	ID for the company to which the model belongs to. For this dissertation the Espressif Incorporated's ID was used: 0x02E5[2].
uint16_t	vnd.model_id	ID for the model within the company.
esp_ble_mesh_model_pub_t*	pub	Abstraction for model publication context. Holds data such as publish address, message's ttl, period between message and messages retransmissions.
uint16_t[]	keys	Holds keys assigned to model. Managed by configuration server model.
uint16_t[]	groups	Holds subscribed groups and virtual addresses. Managed by configuration server model.
void*	user_data	Pointer to model specific user data. Handled within ESP-IDF Bluetooth mesh stack in client models.

References

[1] Arduino esp32 github. Available at https://github.com/espressif/arduino-esp32, last accessed in 23 May 2020.

[2] Blueooth sig company identifiers. Available at https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers/, last accessed in 27 June 2020.

[3] Button component from esp32 iot solutions. Available at https://github.com/espressif/esp-iot-solution/tree/master/components/general/button/button, last accessed in 1 July 2020.

[4] Eesp32 technical reference manual. Available at https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf, last accessed in 5 June 2020.

[5] Esp ble mesh api guide. Available at https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/esp-ble-mesh/ble-mesh-index.html, last accessed in 24 June 2020.

[6] Esp freertos api guide. Available at https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html, last accessed in 23 May 2020.

[7] Esp-idf get started webpage. Available at https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/, last accessed in 5 July 2020.

[8] Esp32 esp32-devkitc-32u mouser product page. Available at https://pt.mouser.com/ProductDetail/Espressif-Systems/ESP32-DevKitC-32U?qs=%252BEew9%252B0nqrCEVvpkdH%2FG5Q%3D%3D, last accessed in 4 July 2020.

[9] Esp32 github repositories. Available at https://github.com/search?q=esp32, last accessed in 05 February 2020.

[10] Home assistant demo. Available at https://demo.home-assistant.io/, last accessed in 16 June 2020.

[11] Home assistant discovery. Available at https://www.home-assistant.io/integrations/discovery/, last accessed in 15 June 2020.

[12] Home assistant documentation. Available at https://www.home-assistant.io/hassio/, last accessed in 28 June 2020.

[13] Home assistant integrations. Available at https://www.home-assistant.io/integrations/, last accessed in 15 June 2020.

[14] Home assistant lovelace. Available at https://www.home-assistant.io/lovelace/, last accessed in 15 June 2020.

[15] Home assistant setup. Available at https://www.home-assistant.io/getting-started/, last accessed in 15 June 2020.

[16] An intro to bluetooth mesh part 2. Available at https://www.bluetooth.com/blog/an-intro-to-bluetooth-mesh-part2/, last accessed in 16 June 2020.

[17] Introduction - zephyr project documentation. Available at https://docs.zephyrproject.org/latest/introduction/index.html, last accessed in 23 May 2020.

[18] Mqtt arduino implementation. Available at https://github.com/256dpi/arduino-mqtt, last accessed in 16 June 2020.

[19] Mqtt esp32 documentaion. Available at https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/mqtt.html, last accessed in 16 June 2020.

[20] Mqtt integration for home assistant. Available at https://www.home-assistant.io/docs/mqtt/discovery/, last accessed in 16 June 2020.

[21] Nrf52-dk mouser product page. Available at https://pt.mouser.com/ProductDetail/Nordic-Semiconductor/NRF52-DK?qs=79dOc3%2F91%2FccrafuGv4fOw%3D%3D, last accessed in 4 July 2020.

[22] nrf52832 product brief. Available at https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF52832-product-brief.pdf?la=en&hash=2F9D995F754BA2F2EA944A2C4351E682AB7CB0B9, last accessed in 23 May 2020.

[23] openhab documentation. Available at https://www.openhab.org/docs/, last accessed in 28 June 2020.

[24] openhab mqtt documentation. Available at https://www.openhab.org/addons/bindings/mqtt/, last accessed in 28 June 2020.

[25] thinger.io mqtt documentation. Available at https://docs.thinger.io/quick-sart/devices/mqtt, last accessed in 28 June 2020.

[26] Thinger.io overview. Available at https://docs.thinger.io/, last accessed in 28 June 2020.

[27] thinger.io pricing. Available at https://pricing.thinger.io/#!/on-premise, last accessed in 16 June 2020.

[28] Thingsboard documentation. Available at https://thingsboard.io/docs/, last accessed in 28 June 2020.

[29] thingsboard mqtt documentation. Available at https://thingsboard.io/docs/reference/mqtt-api/, last accessed in 28 June 2020.

[30] ZigBee Alliance. ZigBee Specificarion. *Standard, Oct*, 2015.

[31] KNX Association. Joining / Fees - KNX Association. Available at https://www2.knx.org/za/knx/technology/developing/software/joining/index.php, last accessed in 07 February 2020.

[32] Mathias Baert, Jen Rossey, Adnan Shahid, and Jeroen Hoebeke. The bluetooth mesh standard: An overview and experimental evaluation. *Sensors (Switzerland)*, 18(8), 2018.

[33] Bluetooth SIG. Bluetooth Mesh Networking - An Introduction for Developers. pages 1–28, 2017.

[34] Bluetooth SIG. Mesh Profile 1.0.1. 2019.

[35] Bluetooth Special Interest Group. Specification of the Bluetooth System Covered Core Package Version 4.2. *History*, 0(April):2272, 2014.

[36] Alessandro Chiumento, Brecht Reynders, Yuri Murillo, and Sofie Pollin. Building a connected BLE mesh: A network inference study. *2018 IEEE Wireless Communications and Networking Conference Workshops, WCNCW 2018*, pages 296–301, 2018.

[37] Jie Ding, Tian Ran Li, and Xue Li Chen. The Application of Wifi Technology in Smart Home. *Journal of Physics: Conference Series*, 1061(1), 2018.

[38] Espressif. ESP32 Series Datasheet. *ESP32 Series Datasheet*, pages 1–61, 2020.

[39] Ebaa Fayyoumi, Sahar Idwan, Hiba A. Muhared, Izzeddin Matar, and Obaidah A. Rawashdeh. L2CAP-based prototype media streaming via Bluetooth technology. *International Journal of Networking and Virtual Organisations*, 14(3):221–238, 2014.

[40] Kang Eun Jeon, James She, Perm Soonsawad, and Pai Chet Ng. BLE Beacons for Internet of Things Applications: Survey, Challenges, and Opportunities. *IEEE Internet of Things Journal*, 5(2):811–828, 2018.

[41] OASIS. MQTT Version 5.0 OASIS Standard. (March), 2019.

[42] Wojciech Rzepecki and Piotr Ryba. IoTSP: Thread mesh vs other widely used wireless protocols - Comparison and use cases study. *Proceedings - 2019 International Conference on Future Internet of Things and Cloud, FiCloud 2019*, pages 291–295, 2019.

[43] Erwin Sacoto-Cabrera, Jorge Rodriguez-Bustamante, Pablo Gallegos-Segovia, Gabriela Arevalo-Quishpi, and Gabriel León-Paredes. Internet of things: Informatic system for metering with communications MQTT over GPRS for smart meters. *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies, CHILECON 2017 - Proceedings*, 2017-January:1–6, 2017.

[44] Sandra Sendra, Miguel Garcia, Carlos Turro, and Jaime Lloret. WLAN IEEE 802.11 a/b/g/n Indoor Coverage and Interference Performance Study. *International Journal On Advances in Networks and Services, volume 4, numbers 1 and 2, 2011*, 4(1):209–222, 2011.

[45] T. Shanthi, S. V. Anushree, S. U. Prabha, and D. Rajalakshmi. DAC to monitor solar powered home appliances and usage control using bluetooth enabled mobile application and IoT. *Proceedings of 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017*, 2018-January:1–4, 2018.

[46] Bluetooth SIG. Bluetooth low energy. Available at https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/.

[47] Sergio Silva, Salviano Soares, Telmo Fernandes, Antonio Valente, and Antonio Moreira. Coexistence and interference tests on a Bluetooth Low Energy front-end. *Proceedings of 2014 Science and Information Conference, SAI 2014*, pages 1014–1018, 2014.

[48] Vaibhav Pratap Singh, V. Tulasi Dwarakanath, P. Haribabu, and N. Sarat Chandra Babu. IoT standardization efforts-An analysis. *Proceedings of the 2017 International Conference On Smart Technology for Smart Nation, SmartTechCon 2017*, pages 1083–1088, 2018.

[49] P. Suresh, J. Vijay Daniel, V. Parthasarathy, and R. H. Aswathy. A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. In *2014 International Conference on Science Engineering and Management Research (ICSEMR)*, volume 8, pages 1–8. IEEE, nov 2014.

[50] Comparison Table. nRF52 Series SoftDevices. Available at https://datasheet.octopart.com/NRF52840-QIAA-R-Nordic-Semiconductor-datasheet-136760423.pdf, last accessed in 28 June 2020.

[51] The Institute of Electrical and Electronics Engineers. IEEE Standard for Information technology–Telecommunications and information exchange between systems LAN and MAN–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-2016. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, 2016:1–3534, 2016.

[52] T H E Thread, Group Disclaims, A L L Warranties, Express Or, U S E Of, T H E Information, Herein Will, N O T Infringe, A N Y Rights, Property Rights, Including Patent, Copyright O R Trademark, F O R A Particular Purpose, I N No, Event Will, T H E Thread, Group Be, Liable For, A N Y Loss, Loss O F Business, Loss Of, U S E Of, Interruption Of, O R For, A N Y Other, Special Or, Punitive Or, Consequential Damages, A N Y Kind, I N Contract, O R In, I N Connection, With This, Document Or, T H E Information, Contained Herein, and Even I F Advised. Thread 1 . 2 in Commercial White Paper Thread 1 . 2 in Commercial White Paper. (September):1–20, 2019.

[53] Kok Seng Ting, Gee Keng Ee, Chee Kyun Ng, Nor Kamariah Noordin, and Borhanuddin Mohd Ali. The performance evaluation of IEEE 802.11 against IEEE 802.15.4 with low transmission power. *17th Asia-Pacific Conference on Communications, APCC 2011*, (October):850–855, 2011.

[54] Li Yi, Lei Shiyao, and Chen Deyuan. Mobile Medical Information Acquisition System. pages 498–505, 2016.

[55] Tetsuya Yokotani and Yuya Sasaki. Comparison with HTTP and MQTT on required network resources for IoT. *ICCEREC 2016 - International Conference on Control, Electronics, Renewable Energy, and Communications 2016, Conference Proceedings*, pages 1–6, 2017.

[56] Bo Zhang, Yufeng Wang, Li Wei, Qun Jin, and Athanasios V. Vasilakos. BLE mesh: A practical mesh networking development framework for public safety communications. *Tsinghua Science and Technology*, 23(3):333–346, 2018.