An Energy-Aware Mobile Gateway for Bluetooth Low Energy-Powered Internet of Things Devices

Mochamad Dandy Firmansyah

Course of Study: Information Technology/InfoTECH

Examiner: Prof. Dr. Kurt Rothermel
Supervisor: Dr. Mohammed Abdelaal

Commenced: 2018-3-14
Completed: 2018-9-14

CR-Classification: C.2.1,C.2.4
Abstract

The term of Internet of Things (IoT) has currently become a novelty in the Internet as an innovation to connect things from all around the world where various sensors are connected using gateways. However, it is not a straightforward task to design such gateways owing to several problems. For instance, there typically exist severe energy consumption constraints due to the limited power source. In most cases, a gateway has to spend an amount of energy for processing the collected data in the network. Additionally, there are myriad of different user interface functions for various services, which in turn raises the question about the reliability and scalability of such gateways. To support the IoT vision, many people have recently used smart mobile devices, e.g., smartphones, tablets, PDA, and laptops, as a gateway for data acquisition in IoT so that these IoT devices can be used in a broader scope. This concept of exploiting our smart devices emerges thanks to their ability to connect things to the cloud via the Internet. In fact, there exist a communication gap between the things implemented with limited power sources to sense the environmental data and the cloud services.

Fortunately, this gap can be bridged by adopting smartphones for forwarding the collected data using their wireless connection technologies. One of the critical technologies that can be used to bridge this communication gap while also still maintaining low energy consumption is Bluetooth Low Energy (BLE). As leverage from the original Bluetooth technology, BLE or known as Bluetooth Smart was initially designed as a power-friendly wireless technology aimed for some novel applications in many industries. To save energy, BLE can be set in a sleep mode and wake up only to receive or send possible packet periodically. By the usage of BLE in modern smartphones, a mobile gateway system can be made in a way that data from the sensors can be passed to the cloud while also considering the energy efficiency in the mobile gateway itself.

In this thesis, we propose a software architecture of energy-aware mobile gateways for IoT applications. The proposed architecture makes continual and efficient data transmission from a set of predefined devices. Moreover, the gateway architecture implements several scheduling algorithms used to efficiently control the sleep mode operations besides handle the simultaneous connection to several BLE sensors. The presented scheduling algorithms comprise Semaphore, Round Robin, Exhaustive Polling and Fair Exhaustive Polling algorithms. To implement the BLE device priority-based approach, several multi-criteria decision making (MCDM) algorithms are also implemented to prioritize the device based on several criteria, such as device power usage, received signal strength indication and the device state. Examples of such MCDM algorithms that have been implemented in this work are the Analytic Hierarchy Process and the Weighted Sum Model.

Furthermore, the algorithms implemented are then evaluated based on two quality of service (QoS) metrics, including the power consumption of the mobile gateway and the through-
put defined regarding the number of packets received per second. The evaluation results showed that Fair Exhaustive Polling (FEP) consumes the lowest energy consumption compared to all other scheduling algorithms with only 12.79 mW. On the other hand, Exhaustive Polling with Analytical Hierarchy Process (EPAHP) has the worst energy consumption among the examined algorithms with 49.14 mW. Concerning the throughput, the Exhaustive Polling combined with Weighted Sum Model (EPWSM) has the most prominent data throughput compared to all other algorithms with 101.18 packets/s while Fair Exhaustive Polling (FEP) has the lowest throughput value with 50.98 packets/s. To sum up, the proposed mobile gateway architecture is exceptionally efficient for handling data forwarding from multiple BLE sensors to the cloud services with energy awareness.
Acknowledgement

Firstly, I would like to thank God who has given me patient, endurance and strength during the hard time and challenging situations so that I can finish this Master Thesis.

I dedicate this thesis to the loving memory of my father, H. Anam, who has given me proper guidance, motivated me to pursue a higher degree education and inspired me a lot in numerous fruitful discussion. Thank you for everything you have done for me and our family. And to my mother, Hj. Sri Sugriarti, who have never stopped praying for me on the day and night. You are the best woman that I have ever known in my life forever and ever. To my sisters, Mbak Lulu, Mbak Ewik, Mbak Oki, thank you for your sincere care, love and support. Subsequently, I would like to express my sincere gratefulness to “Lembaga Pengelola Dana Pendidikan (LPDP)” who has granted me to pursue a master degree in Stuttgart, Germany.

Furthermore, I would like to thank my best friend since bachelor degree, Bram, for accompanying me throughout the tough time and during my living in Stuttgart. It would have never been the same without you. Also, I would like to thank my friends in Stuttgart, Ganda, Jumadi, Riza, Ahya, Winda, Hajrah, Karlina, Firas, Dinar, and Riska for helping me whenever I need them and for the beautiful memories during our stay in Stuttgart. Then, I would like to thank my teammate for this work, Marwan, for his continuous support and spirit.

Lastly, I would like to say thank you to my supervisor Dr. Mohammed Abdelaal for giving me the opportunity and for his continued support and feedback throughout the work. It was an honor to work with you.
# Contents

**Abstract** i  
**Acknowledgement** iii  

1 **Introduction** 1  
1.1 Motivation 1  
1.2 Problem Statement 3  
1.3 Contributions 4  
1.4 Thesis Overview 5  

2 **Background** 7  
2.1 Internet of Things 7  
2.1.1 Mobile Gateway 9  
2.1.2 Public Sensing 9  
2.2 Android Operating System 11  
2.2.1 Android Software Stack 11  
2.2.2 Android Programming 12  
2.3 Wireless Technology 13  
2.3.1 Bluetooth Wireless Technology 14  
2.3.2 Bluetooth Low Energy 15  
2.4 Related Work 21  
2.4.1 BLE Wireless Technology 21  
2.4.2 BLE-Powered IoT Solutions 22  
2.4.3 Public Sensing 22  
2.4.4 Mobile IoT Gateway 23  

3 **System Model** 25  
3.1 IoT Sensors 26  
3.2 Mobile Gateways 28  
3.3 Cloud Services 29  
3.4 Network Topology 30  

4 **Mobile Gateway Design** 31  
4.1 Design Challenges 31  
4.2 Android Multithreading 32  
4.2.1 Single-Threaded Application 33  
4.2.2 Multi-Threaded Application 34  
4.2.3 Thread Safety 34
List of Figures

2.1 Basic gateway .................................................. 9
2.2 Mobile gateway ................................................. 10
2.3 Android software stack ......................................... 11
2.4 Bluetooth protocol stack ....................................... 14
2.5 The BLE protocol stack ......................................... 16
2.6 Bluetooth versions and device mode ......................... 16
2.7 The BLE channels ................................................ 17
2.8 GATT data hierarchy ............................................. 18
2.9 The BLE packet breakdown ..................................... 19
2.10 The illustration of time allocation for advertising and scanning in the BLE ........ 20
2.11 The BLE operation modes ....................................... 20

3.1 Proposed system model .......................................... 25
3.2 Example of the IoT sensors ..................................... 26
3.3 Example of smart devices ......................................... 27
3.4 nRF connect and a smartphone ................................ 27
3.5 Entity-relationship diagram ..................................... 29
3.6 Network topology for the BLE connection ................... 30

4.1 One CPU executes two threads simultaneously .............. 33
4.2 IPC performed by binder framework ......................... 40
4.3 Remote communication interface using AIDL ................ 40
4.4 Remote procedure calls performed by AIDL .................. 41
4.5 Proposed system mobile gateway design ...................... 43
4.6 The self adaptive system based on MAPE ...................... 45
4.7 Sequence diagram of proposed mobile gateway ............. 46
4.8 Scheduling with semaphore algorithm ....................... 50
4.9 Scheduling with round robin algorithm ...................... 52
4.10 Scheduling with exhaustive polling algorithm ............. 53
4.11 Scheduling with fair exhaustive polling algorithm .......... 55
4.12 Hierarchical structure for the AHP ......................... 58

5.1 Class diagram from the proposed mobile gateway .......... 71
5.2 Screenshot of the system UI ................................... 72

6.1 Screenshot of the Android Apps used for experimental tools . 90
6.2 Wireshark network protocol analyzer tool .................... 90
6.3 Power consumption with number of devices = 2 ............ 93
6.4 Power consumption with number of devices = 4 ............ 93
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5</td>
<td>Power consumption with number of devices = 6</td>
<td>94</td>
</tr>
<tr>
<td>6.6</td>
<td>Power consumption with number of devices = 8</td>
<td>94</td>
</tr>
<tr>
<td>6.7</td>
<td>Power consumption with number of devices = 10</td>
<td>95</td>
</tr>
<tr>
<td>6.8</td>
<td>Throughput with number of devices = 2</td>
<td>96</td>
</tr>
<tr>
<td>6.9</td>
<td>Throughput with number of devices = 4</td>
<td>96</td>
</tr>
<tr>
<td>6.10</td>
<td>Throughput with number of devices = 6</td>
<td>97</td>
</tr>
<tr>
<td>6.11</td>
<td>Throughput with number of devices = 8</td>
<td>97</td>
</tr>
<tr>
<td>6.12</td>
<td>Throughput with number of devices = 10</td>
<td>98</td>
</tr>
<tr>
<td>6.13</td>
<td>Memory consumption comparison</td>
<td>100</td>
</tr>
<tr>
<td>6.14</td>
<td>Energy vs. data comparison</td>
<td>102</td>
</tr>
<tr>
<td>6.15</td>
<td>Latency vs data comparison</td>
<td>103</td>
</tr>
<tr>
<td>6.16</td>
<td>Comparison of power consumption with MAPE vs. without MAPE (dominant power consumption)</td>
<td>105</td>
</tr>
<tr>
<td>6.17</td>
<td>Comparison of throughput with MAPE vs. without MAPE (dominant power consumption)</td>
<td>106</td>
</tr>
<tr>
<td>6.18</td>
<td>Comparison of power consumption with MAPE vs. without MAPE (dominant throughput)</td>
<td>106</td>
</tr>
<tr>
<td>6.19</td>
<td>Comparison of throughput with MAPE vs. without MAPE (dominant throughput)</td>
<td>107</td>
</tr>
<tr>
<td>6.20</td>
<td>Indoor floor plan used for mobility of the smartphone experiment</td>
<td>108</td>
</tr>
<tr>
<td>6.21</td>
<td>The effect of mobility towards throughput</td>
<td>109</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of some wireless technologies</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Scaling for the AHP weight</td>
<td>58</td>
</tr>
<tr>
<td>4.2</td>
<td>Normalized matrix for all 3 criteria</td>
<td>59</td>
</tr>
<tr>
<td>4.3</td>
<td>Normalized matrix for RSSI subcriteria</td>
<td>60</td>
</tr>
<tr>
<td>4.4</td>
<td>Normalized matrix for device state subcriteria</td>
<td>60</td>
</tr>
<tr>
<td>4.5</td>
<td>Normalized matrix for the smartphone’s residual energy subcriteria</td>
<td>60</td>
</tr>
<tr>
<td>4.6</td>
<td>Weighting scale for subcriteria in RSSI criteria</td>
<td>62</td>
</tr>
<tr>
<td>4.7</td>
<td>Weighting scale for subcriteria in smartphone’s residual energy criteria</td>
<td>63</td>
</tr>
<tr>
<td>6.1</td>
<td>Some utilities used for application overhead experiment</td>
<td>91</td>
</tr>
<tr>
<td>6.2</td>
<td>Ranking of scheduling algorithm based on power consumption</td>
<td>95</td>
</tr>
<tr>
<td>6.3</td>
<td>Ranking of scheduling algorithm based on throughput</td>
<td>98</td>
</tr>
<tr>
<td>6.4</td>
<td>Some utilities used for memory consumption experiment</td>
<td>99</td>
</tr>
<tr>
<td>6.5</td>
<td>Some utilities used for specific vs. generic gateway experiment</td>
<td>101</td>
</tr>
<tr>
<td>6.6</td>
<td>Some utilities used for self adaptive system with MAPE experiment</td>
<td>104</td>
</tr>
<tr>
<td>6.7</td>
<td>Some utilities used for mobility of smartphone experiment</td>
<td>108</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter provides an overview of the problem tackled in this thesis by clearly highlighting the proposed solutions. First, we discuss the motivations through which the main problem is described. Next, the issues which are going to be faced in this work will be identified. Then, we list the contributions that have been done in this work. Lastly, a short overview of the thesis’ chapters will also be defined.

1.1 Motivation

In the recent decade, the Internet has rapidly changed our lifestyle. With the Internet, the way of communication and information exchange have been revolutionized. From people to people, people with things and the newest one of things to things, all kinds of communication have become possible with the usage of the Internet. As a new technology paradigm in the Internet, Internet of things (IoT) has recently been talked as a vision in which the data obtained by the things can also be harvested to be used for many applications in many fields by embedding computational capabilities to some objects that have valuable data. According to Cisco [1], these fields are including healthcare, manufacturing, energy, transportation, education, government, smart cities, retail, etc. Moreover, the number of IoT connected devices worldwide, according to statistic [2], has reached 15 billion in 2015 and for 2025, the installed base of IoT devices is expected to grow to almost 75 billion worldwide, or it is increasing triple its number in about ten years. Furthermore, there will be more than one billion U.S. dollars annually from 2017 onwards projected by the overall IoT market [2].

When talking about IoT as a new technology paradigm, one of the most important elements which support the IoT system is the so-called wireless sensor networks (WSN). Specifically, WSNs provide a means for information aggregation from the environment. Thus, WSNs describe the next evolution of systems automation that relies on sensing the environmental data [3]. Information about the physical world can be accessed by several systems. In this manner, WSNs can provide a virtual layer so that heterogeneous information systems can collaborate and deliver shared services [4]. The usage of wireless sensing systems is broadly advantageous thanks to the fact that they are easier and cheaper to use in IoT applications compared to the established infrastructure, e.g., wired, satellite or optical communication. However, not only in WSNs but also in the field of wireless sensor nodes, which the sensors are not connected each other by the network, instead of using intermediary devices to gather
data from the sensors and further the processing in the cloud, are also popular in recent years. To collect as many environmental data as possible, the vast number of sensors are deployed in distributed locations. These sensors are also designed with extremely limited resources, such as limited power, processing, and storage capacity so that the cost of building such an intelligent system based on the sensory data can be suppressed.

In general, building an IoT platform that uses data collected from the collection of sensor nodes has become a common interest. Nevertheless, there is a question regarding data consumption of this platform. In other words, a problem arises about where should we deliver the collected data so that its valuable meanings can be presented to the users? The answer lies in the so-called cloud computing technology. By distributing data to the cloud, we can further process and analyze the collected data so that the value of the data can be obtained. Cloud services are typically equipped with unlimited resources, such as storage, processing power, etc. Thus, it can remedy the problem of limited resources lying on the IoT platform. In fact, consuming the data provided by the WSNs can be a challenge, especially while detecting the relevant quantities, monitoring and collecting the data, assessing and evaluating the information, formulating meaningful user displays, and performing decision-making and alarm functions. To provide solutions to these challenges, cloud services are used to provide a facility for data processing without caring so much about the processing resources.

As aforementioned, the sensors—deployed in the environment to collected data—are typically resource-constrained. Therefore, they can not afford direct connections to the cloud via traditional communication technologies, such as WiFi and cellular networks. In this context, a research question arises about the suitability of our smart mobile devices to negotiate with the IoT sensors while passing by to upload the sensors’ data in an energy-efficient manner, known as opportunistic sensing (OppS). In fact, OppS is a cultivation term from an already known paradigm of public sensing (PS) which obtains sensor data using additional resources of commodity smartphones. In these cases, smartphones are used because they have become a commonly used device for communicating and exchanging information. Additionally, they have an incredibly increasing computing power in a small and compact shape nowadays. Moreover, the utilization of smartphones mostly by everyone has become another critical factor. As leverage from the term of PS, OppS gathers data from sensors by using mobile smart gadgets (e.g., Smartphones, Tablets, etc.) carried by people without their entanglement.

Another critical factor lies in the usage of smartphones for OppS is they provide numerous ways for communicating with the utilization of wireless communication technologies, such as Bluetooth, Bluetooth Low Energy (BLE), 3G / 4G cellular network, wireless fidelity (WiFi), near field communication (NFC), etc. These wireless connections provide a forwarding mechanism of sensor data to the universal and transparent access to the cloud services. Furthermore, as mentioned earlier about the computing power in smartphones, these devices with the usage of the database can be used to build an energy-aware mobile gateway without the need for additional hardware and predefined infrastructure. Mobile gateway is leverage from the gateway term in which it can be carried or moved easily without having the problem of many setups and installation settings.
1.2 Problem Statement

In the previous section, we discussed the gap between the IoT sensing devices and the cloud services. In this section, we elaborate on the problem of data forwarding and the need for reliable gateways between these two components, i.e., IoT devices and the cloud services. In fact, establishing a direct connection—without mediator hops—between the IoT devices and the smartphones highly reduces the bandwidth usage and energy consumption [6]. In most cases, the sensing devices have limited wireless connection technologies due to their energy limitations where these devices are required to independently operate for a long time span [10]. To achieve data exchange between the mobile gateways and the sensor nodes, there should be at least one wireless communication technology which will be used in both sections commonly. According to [11], the implemented software architecture for multi-standard and multi-technology inter-operation presents a reduced use of hardware resources in front of a relatively high-energy consumption, mostly due to the simultaneously active radio interfaces combined with small battery capacity, that limits the smartphone’s lifetime. To achieve an energy-efficient data exchange using wireless technology, the transition between sleep and wake up modes is highly required where the sleep mode usually is to save energy while no data exchange is needed.

One of the wireless communication technologies that implement these modes is known as Bluetooth Low Energy (BLE) [12]. Compared to other wireless connection technology, such as Zigbee, Bluetooth Classic, WiFi and Z-Wave, BLE has the lowest power usage for communication in the short range while still having a moderate throughput value [13, 14]. Because of retrieving environmental data from nearby sensor nodes, a mobile gateway provides PS capabilities. Data exchange using BLE may sometimes be hindered due to whether the smartphone’s proximity is in range with the sensor nodes or due to the data availability from the sensor nodes themselves. This can sometimes reduce the energy efficiency of the smartphones which may also reduce the lifetime of the usage of the smartphone as a mobile gateway. A scheduling method is then needed to set the timing of each BLE-based data exchange between a mobile gateway and a set of IoT sensing devices.

In general, the services offered by a sensor node may differ from one to another sensor node. For instance, a heart rate service provided by a specific sensor node $A$ may vary from a temperature service offered by a sensor node $B$. There exists a case where an application-specific gateway works only for a specific sensor node. Nowadays, many sensor nodes produced by a particular manufacturer employ their particular application gateway. This may isolate from one service to another, or known as siloed versions where instead of requiring nodes to form mesh networks to relay data back to a few Internet-connected gateways, and each node could piggyback on passing smartphones to offload or receive data [9], e.g., Xiaomi MiBand [15] and Fitbit [16].

Furthermore, there is another problem regarding the devices cross-platform connectivity where the model of specific smartphone limits some newer wearable devices to which these devices ought to be able for connecting to another model of smartphone. Even though these wearable devices also provide data using BLE connection, because of this limitation, this data can not be gathered by another model of smartphone. Moreover, to connect a BLE peripheral
device for IoT into precisely one smartphone puts a failure to the aim of this device due to the lack of the existence from the smartphone may lead to loss of functionality of the tethered device to send or receive data [9].

In this thesis, we propose an approach that adopts mobile gateways which supports several usage scenarios. The proposed generic gateway architecture collects data provided from the nearby IoT devices (sensor nodes) and opportunistically works as a relay of data from the adjacent devices to the cloud. To build such a system, as in the case of the sensor nodes, it is also essential to have an energy-aware mechanism due to the limited battery capacity of the smartphones. Furthermore, another vital scenario for our proposed system is the implementation of an open gateway model for authorizing IoT devices to use any nearby smartphones for forwarding or receiving data and acting as the gateway. Thus, this thesis mainly discusses on how a mobile gateway can be implemented in such a way that it performs an energy-efficient architecture and provides an open gateway architecture to tackle any services offered by some already known IoT devices located nearby the designated mobile gateway.

1.3 Contributions

The primary objective of this thesis is to design and implement a software architecture for a generic mobile gateway to serve the myriad of IoT applications. This architecture leverages BLE as the main wireless communication so that the energy consumed while running the application is minimized. To sum up, the contributions of this thesis are summarized as follows:

1. Designing a software architecture of a generic mobile gateway. The developed architecture leverages Android service interface for serving different types of IoT devices. Specifically, the gateway can connect to a range of BLE-powered IoT devices, retrieve data from these devices, and eventually either locally process the collected data or upload it to a cloud service. To this end, the gateway dedicates a specific service fragment for each IoT device, according to the purpose of the collected data. For instance, the gateway can visualize the data coming from a fitness device as well as uploading this data to a specific cloud service.

2. Designing several scheduling algorithms to establish a simultaneous connection between a mobile gateway and several BLE-powered IoT devices. The main goal behind these scheduling algorithms is to improve the energy efficiency of the mobile gateway together with achieving an acceptable level of data throughput defined concerning the number of received packets per second from each IoT device. The developed scheduling algorithms determine the time span for connecting to a certain IoT device. Moreover, scheduling algorithms determine the timing of data exchanging data between the mobile gateway and the IoT devices. To achieve these goals, we implemented and designed a combinations of the following algorithms: Semaphore, Round Robin, Exhaustive Polling, and Fair Exhaustive Polling.
3. Designing a *Priority-based* scheduling algorithm that relies on ranking the IoT devices before establishing the BLE connection. To this end, several algorithms have been used for ranking possible candidate IoT devices using multi-criteria decision making (MCDM) approach. The criteria considered here consist of the device state, the smartphone’s residual energy and the RSSI values. In this context, two MCDM mechanisms have been examined, including the *Analytic Hierarchy Process (AHP)* and the *Weighted Sum Model (WSM)*.

4. Designing a self-adaptive system using the well-known *monitor, analyze, plan and execute (MAPE)* strategy used for determining which scheduling algorithm is suitable during runtime with the use case scenario designed on each possible situations. The main goal here is to adapt the gateway according to the runtime conditions. Therefore, we consider the smartphone’s residual energy as well as the number of nearby IoT devices to decide on which scheduling algorithm to adopt.

5. Implementing the designated scheduling algorithm and MCDM mechanism for decision making on the smartphone running the Android and SQLite database [17]. Additionally, we measured the energy consumption of each scheduling algorithm implemented using a software-based profiling method from Qualcomm, namely *Trepn Power Profiler* [18]. Similarly, we measured the data throughput for each scheduling algorithms using *Android Bluetooth HCI Snoop Log* protocol in Android developer options [19] and then we evaluated these results using the *Wireshark Network Protocol Analyzer* [20].

### 1.4 Thesis Overview

This remainder of the thesis is organized as follows. Chapter 2 mainly introduces the background of the technologies related to this thesis, including public sensing and the BLE wireless technology. In chapter 3 we present the system model and the adopted assumptions. Afterward, we introduce the proposed software architecture in Chapter 4. In Chapter 5 we elaborate on the implementation of the proposed scheduling algorithms, before presenting the measurements and the evaluations of the proposed architecture and algorithms in terms of the smartphone’s energy efficiency and the data throughput in Chapter 6. Finally, Chapter 7 concludes this work with some proposals for future work.
Chapter 2

Background

The main objective of this Chapter is to provide a good understanding of the key concepts that form the basis of this thesis. In the following sections, we discuss the concept of the Internet of things (IoT) followed by the mobile gateway. Afterward, public sensing and opportunistic sensing concepts are explained in detail. Subsequently, the basic principles of the Android operating system are introduced before focusing on the wireless technology implemented in the Android environment. The last section revolves around the related works regarding research in this area.

2.1 Internet of Things

Generally, the history of IoT cannot be separated from the history of the Internet. In 1992, the TCP/IP allowed some Programmable Logic Controls (PLC) to have sorts of reliable connectivity [21]. Furthermore, in the late 1990s, Kevin Ashton introduced the term of IoT where he implemented an idea to use RFID for supply chain of Procter & Gamble (P&G) company [22]. He added that computers must be empowered to gather information so that they can see, hear and smell the world themselves [23]. Therefore, IoT has been known as a world where “things” become connected through the Internet. IoT has also been described as the worldwide network of uniquely addressable interconnected objects based on standard communication protocols [24]. The electronic devices—which can produce data by communicating with the external world and sensing the environmental phenomenon—are referred to as “things” that are related to IoT. Hence, the term of IoT broadens the Internet usability to not only some devices which are generally built to have connectivity, such as PCs, laptops, and smartphones, but also to some objects that are usually built by not having ability for connectivity, such as electronic appliances, healthcare devices, some devices which are deeply rooted in technology (e.g. cars, motorcycles, airplanes) and some foreign objects or even living things (such as woods, livestock, plantations) that have been implemented by embedded computational capabilities [4].

Accordingly, IoT has opened a gate to a more advanced application of embedded computers to be used as sensor nodes to sense environmental data. An embedded computer is a computer with a dedicated purpose that lies in the electrical or mechanical system. The things that differentiate embedded computer with general purpose computer, e.g., PCs or laptops, are the functional purpose of usage, the ability of low power consumption, low cost per unit, the
ability to be used in a severe operating range, and small or compact. Due to the previously mentioned requirements, an embedded computer has been made with the trade-off in the limited resources, such as limited storage and processing power capability [10]. Such limited resources may burden the embedded computers to store a significant amount of data when they are used as sensor nodes. Furthermore, any processing of data that requires a high computational processing power cannot also be done in these embedded computers. Thus, it is required to further the data processing somewhere that is capable of doing so.

Consequently, sensor nodes in IoT are supposed to be connected to the Internet. The approaches to connect sensor nodes to the Internet can be whether they connect directly to the Internet using their Internet Protocol (IP) connectivity or they use a set of central management systems, e.g., base station or gateway that forwards the data from the sensor nodes to the Internet. There are many challenges which must be taken into consideration when the sensor nodes connect directly to their IP address, such as security, data privacy, for each of the device [1]. Nevertheless, to further the processing of data from sensor nodes, the implementation of cloud computing in IoT is necessitating because the cloud can provide better resources than the embedded computers in the sensor nodes themselves. The latter approach of connection is more preferable because, with one gateway device, data from many nearby sensor nodes can be obtained sequentially while also forwarding the already gathered data to the cloud. By integrating cloud computing into IoT, the vision of connecting “things” to the Internet can be realized so that many applications that implement IoT as smart devices can also be offered.

One of the apparent use of IoT is the wireless sensor networks (WSNs). WSNs comprise the sensor nodes distributed geographically which combine into networks and are equipped with one or more sensors to monitor environmental phenomena and gathering physical measurements [10]. The networked of sensors with different modalities in distributed locations work like a sentient organism that gathers or sensing data of the environment from its surroundings and formulating these data and change it into a valuable data that could create smart environments in WSNs [3].

However, it is not only WSNs that have increased the usage of IoT in everyday’s life, but also the sensing devices which are not inter-networked may increase the use of IoT in the future significantly. These sensor nodes differ in their usage from one to another, e.g., some newer wearable devices which can have different sensors, such as temperature, heart-rate, step counter, etc. The current strategy of commercial sensor nodes that require a limited model of devices or precisely one device which is adequate of connecting, segmented and application-specific approach to wireless connectivity creates the isolation on each of these sensor nodes to a specific device. This phenomenon is referred to as the siloed versions problem [9].

Establishing connections via the wireless medium has incredibly changed the way of connecting among two or more objects. The usage of wireless connectivity has driven some usage of mobile computers, including tablets, laptops, and smartphones into a prosperous era. Some wireless technologies have been widely used, and these technologies provide transparent and universal access to the Internet and cloud services [9]. In IoT, wireless communication technologies, especially the low-energy technologies, play a significant role to connect several sensor nodes and make them visible or accessible through the Internet.
2.1 Internet of Things

2.1.1 Mobile Gateway

In the past few years, the term “mobile gateway” has known in the literature as the cutting edge of the gateway technology. A gateway is known as a system capable of translating data from a low-power link to the Internet at large [9]. Figure 2.1 depicts the settings for a gateway where one gateway can be used to connect several peripheral devices to the cloud via the Internet connection. Whereas, in the mobile gateway, the devices act as gateways through migrating from one place to another. Mobile gateways serve as a basis for the IoT platform in which they forward data from the sensors to the cloud service. However, the usage of mobile gateways presents a non-traditional trade-off between the requirement of a re-routing frequently to ensure the best operation of the network and the urge for minimizing topology management overhead [25].

Figure 2.2 shows the settings of the mobile gateway. As shown in the Figure 2.2, several mobile gateways are used for enabling gateway services for each type of IoT devices. In [9], the authors commented about IoT problems related to gateways. Generally, gateways are used to deliver data to the Internet from a specific sensor node where such gateways are isolated in the form of being not able to gather data from other sensor nodes. Instead of establishing mesh networks to deliver the collected data from sensor nodes to several mobile gateways, it is more convenient to implement an open gateway architecture for multiple types of IoT sensing devices.

2.1.2 Public Sensing

In the recent decade, the trend of harnessing smartphones as data muling device—which gather data from sensor nodes—has broadly increased. Smartphones represent an ideal data muling platform thanks to being highly pervasive, supporting relatively powerful hardware, being equipped with various sensors (e.g., accelerometer, GPS, temperature, etc.), their widespread
use by many people together with an abundance of communication technologies and supporting application development [7–10]. Thereby, the term public sensing (PS) describes the smartphones’ utilization by mobile participants for collecting sensor data. However, many of the applications of PS in smartphones do not consider that smartphones as mobile devices have a limited energy resource based on the size of the battery themselves. Therefore, the usage of PS in many smartphones owned by the participants might have a problem in the case of energy consumption consumed by the installed PS App. Accordingly, it is mandatory to keep the energy consumption of the PS App to minimum [7].

There are two different types of PS systems: opportunistic (OppS) and participatory (PPS). The difference between these types is about the involvement of the smart phone’s owners to decide which data to be sent, which request of the data will be responded, etc. In PPS, the user is actively involved in the decision. In contrast, OppS does not involve the user in sensing related decision. Instead, it moves the decision maker into the sensing system itself. Hence, OppS obtains the sensor data by opportunistically sensing them using smartphones [6–10].

As aforementioned for OppS decision making, the criteria existed in the sensing system automates the decision making from the selection of sensor nodes. It can be done using some mechanisms which known as multi-criteria decision making (MCDM). In our work, we use OppS by implementing two MCDM mechanisms (e.g., Analytical Hierarchy Process (AHP) and Weighted Sum Model (WSM)) to decide the priority of available nearby IoT devices. Such a decision-making mechanism requires some criteria in the input for determining the outcome result of it. Our approach uses 3 different criteria for the input of those MCDM algorithms: the RSSI, the smartphone’s residual energy and the device state. These criteria are given to the MCDM to provide the list of prioritized devices which become candidates for the mobile gateway to connect using BLE and obtain the data provided by the sensors.
2.2 Android Operating System

Android is an open source operating system for smart mobile phones developed by Google. As a novelty in the mobile phone’s operating system, Android has offered many opportunities for building mobile applications. Hence, unlike the other well known mobile operating systems: Microsoft Windows Phone and Apple iPhone OS (IOS) which are built on the proprietary operating system, Android uses an open-source Linux kernel. The usage of open-source Linux kernel has given android support to mobile application development. Furthermore, it also offers a simplified development environment for the mobile application like in Microsoft’s and Apple’s mobile platform [26].

2.2.1 Android Software Stack

There are five main elements in the Android which formed its main system. These elements mainly consist of a collection of C/C++ libraries and a Linux Kernel. As shown in Figure 2.3, these elements are [26]:

1. Linux Kernel: Kernel provides not only an abstraction layer between hardware and the remainder of the stack but also the core services.
2. Libraries: This stack runs on top of the kernel and includes various C++ core libraries from Android. Some other parts from this stack are the database with SQLite, Graphics, Media, SSL & Webkit, and Dalvik Virtual Machine.
libraries, SSL and WebKit for Internet security and web browser, a media library for audio and video playback and a surface manager for providing display management.

3. Android runtime: This stack distinguishes an Android phone with a mobile Linux implementation. This stack consists of Core libraries that provide most of the functionality available in core Java libraries and Android specific libraries. Then it also consists of Dalvik VM which a register-based Virtual Machine to run multiple instances efficiently.

4. Application Framework: The classes provided by the application framework to create an Android application and provide a generic abstraction for hardware access, manage user interfaces and application resources.

5. Application Layer: This stack provides applications in android whether it is a third-party or native application. These applications are built on the application layer employing the same API libraries. Using Android runtime, the application layer runs using services and classes from the application framework.

2.2.2 Android Programming

To build an application layer mobile gateway using an Android, we must program the application written using Kotlin, Java, and C++ language. After building the application, deployment is done by building an apk file (an Android package file) and implement it into a set of smartphones that run the Android.

Similar to the applications that run in Java, Android Apps also run sandbox security that isolates the application code from one to another. This is because one virtual machine (VM) is assigned to one Linux process. So is on the Android, every Android app owns one Linux process. Furthermore, due to the system is inherited from Linux, it assigns a unique Linux ID on each App. This ID is what is known as the process ID (PID) in Linux which is unknown to the App.

Another essential thing to do in Android is a security aspect on each App. The principle of least privilege is implemented by Android system so that to use the components which are required to be used in work, and each Android App must be given access to each component. Hence, the parts of the system that have not been given access to a specific App cannot be used by this application. Furthermore, for building an Android App, the essential building blocks are the App components. There are four types of App components for which each of them is an entry point for the user to enter the App, such as:

- Activities: This component acts as an entry point for interaction with the user.
- Services: This component is used for making a background App that keeps running for all kinds of reasons.
- Broadcast Receivers: This component works as a listener which enables the delivery of events in the system to the App outside of regular user flow.
• Content Providers: A shared set of App data, which can be stored in the file system, is managed by this component.

• Fragment: This component is a portion of the user interface which can be made into a modular section of an activity.

### 2.3 Wireless Technology

Since the last two decades, the connection between two or more objects has been revolutionized by wireless technology. The term of wireless itself means the exchange of information or data without the usage of an electrical conductor. Wireless technology is also an integral part of communication in IoT because of less complexity in deployment and being cheaper than fixed wired infrastructure [10]. There are several wireless technologies which can be used for IoT wireless connection, such as Bluetooth, Bluetooth Low Energy (BLE), WiFi, Zigbee, and Z-Wave.

| Table 2.1: Comparison of some wireless technologies [13] |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| **Wireless Technology** | Bluetooth Classic | Bluetooth 4.0 (BLE) | WiFi | Zigbee | Z-Wave |
| **Promoter** | Bluetooth SIG | Bluetooth SIG | WiFi Alliance | Zigbee Alliance | Z-Wave Alliance |
| **Standard** | IEEE 802.15.1 | IEEE 802.15.1 | IEEE 802.11 | IEEE 802.15.4 | - |
| **Topology** | Star | Star | Star | Mesh, Tree, Star | Mesh |
| **Frequency** | 2.4 GHz | 2.4 GHz | 2.4 GHz | 2.4 GHz | 900 MHz |
| **Data Protection** | 16-bit CRC | 16-bit CRC | 32-bit CRC | 16-bit CRC | 8-bit CRC |
| **Range** | 10 m | 30 m | 10-100 m | 100 m | 100 m |
| **Data Rate** | 1-3 Mbps | 1 Mbps | 11 Mbps | 250 kbps | 10-40 kbps |
| **Energy Usage** | Low | Very Low | High | Very Low | Low |

As listed in Table 2.1, all of these wireless technologies have their criteria which differ from one to another. Nevertheless, the growth of BLE has significantly outperformed the other emerging wireless technologies listed in Table 2.1. The two main factors influencing the BLE growth: the emerged of BLE inherited from the classic Bluetooth and the prior support of BLE on many smart devices (e.g., smartwatches, fitness trackers, etc.). Meanwhile, as mentioned in [13], the selection of a technology for communications relies on the aimed application. Subsequently, IoT deployments according to [14] are likely to use ultra-low power and short-range communication technologies. Therefore, when building an energy-efficient mobile gateway for IoT devices, the last three criteria from Table 2.1 must heavily be considered, namely range, data rate and energy usage. In term of the energy usage, BLE and Zigbee offer significant low-energy usage. Furthermore, these two wireless technologies offer relatively good ranges.
Meanwhile, BLE offers four times higher data rate than Zigbee. Besides, BLE is a simple and secure option for communications [13]. Thus, BLE is one of the suitable wireless technologies that can be used for communication between sensor nodes and IoT mobile gateway.

### 2.3.1 Bluetooth Wireless Technology

The development of emerging wireless technologies had become a trend in the late 1990s. Many wireless technologies had been invented at that time, and one of them is called Bluetooth Technology. Bluetooth was introduced in 1994. Nevertheless, the development of Bluetooth itself has begun since the Bluetooth Special Interest Group (SIG) —a group of more than 30,000 companies that operate in the areas of telecommunication, networking, consumer electronics, and computing— took over and made an open industry specification which specified both application and protocol and a qualification program developed to ensure end-user value for its products.

Since the beginning, Bluetooth has been designed for a short-range connectivity solution for portable, personal and handheld electronic devices. It serves primarily as a substitute of the interconnect cables between personal devices and aims for providing universal user-friendly and low-cost air interface to replace RS 232 serial port protocol. Furthermore, another essential item for Bluetooth as wireless technology is for enabling ad-hoc connectivity among the personal devices for some individuals is allowed. Hence, the exchange of data without the need for the infrastructure for supporting their communication [28].

![Bluetooth protocol stack](image)

Current development of Bluetooth has already reached version 5.0. Since 1999, Bluetooth has received an IEEE standard under 802.15 task groups. The group of 802.15.1 was chosen to provide the base of Bluetooth standard. Meanwhile, 802.15.2 group managed the problems
2.3 Wireless Technology

between 802 wireless technologies. Furthermore, 802.15.3 group developed the standard for high data rate radios (> 20 Mbps), and 802.15.4 group studied the standard for low data rate radios (< 200 kbps) [28].

Figure 2.4 describes the protocol stack of Bluetooth wireless technology. The stack is divided by two main protocols, such as transport and middleware. On top of it lies the Applications or Profiles layer. In the transport protocol layer, some main components are the radio, baseband, link manager, host controller interface and logical link control and adaptation protocol. First, the radio layer provides the technical characteristic of the Bluetooth radios which uses 2.4 GHz frequency and employs Frequency Hopping Spread Spectrum (FHSS) technique. It uses modulation of Gaussian Frequency Shift Keying (GFSK), and the baud rate is 1 Msymbol/s with the raw transmission speed is 1 Mbps. Meanwhile, the baseband layer specifies the essential procedures so that the communication with each other using this technology is enabled. Furthermore, the link manager protocol describes the transactional protocol between the two link management entities on two Bluetooth devices that are responsible for establishing the Bluetooth link properties. The host controller interface (HCI) is used to interface the access to the lower layers of the Bluetooth stack for the host devices. Lastly, the logical link control and adaptation protocol (L2CAP) covers the specifics of lower layer Bluetooth stack and serves the higher layer of Bluetooth stack with a packet interface [28].

In the middleware protocol layer, some crucial protocols are service discovery protocol (SDP), RFCOMM protocol and telephony control signaling (TCS) protocol. SDP protocol is used for Bluetooth device to inquiry the services available on another device. However, this protocol only serves services information, but it does not provide access to them. The RFCOMM protocol exposes a serial interface to the packet-based Bluetooth transport layer so that like in communication using cable, the signaling in Bluetooth wireless technology is determined by this protocol. TCS protocol does the telephony control using the AT command set. Other protocols are also added to the Bluetooth stack, such as point to point protocol (PPTP) which enables IP communication over serial lines, Object Exchange (OBEX) protocol for transporting objects between devices and so forth [28].

2.3.2 Bluetooth Low Energy

The version of Bluetooth 4.0 has been introduced in 2011 that brought some additional new core specifications. One of this addition is the presence of Bluetooth Low Energy (BLE) or known as Bluetooth Smart. At first in 2011, BLE was developed to be used in many monitoring applications and short-range control [12]. Nevertheless, BLE brings an ultra-low power consumption to the Bluetooth wireless technology so that it can be used for communication in many low power computer systems, such as microcontrollers, embedded computers, and Smart Devices. As leverage from the standard Bluetooth, BLE brings some specifications from its predecessor.

Figure 2.5 shows the protocol stack from BLE. As depicted in the picture, some protocols are inherited from classic Bluetooth. Nevertheless, although standard Bluetooth inherits its controller to the BLE, there is an incompatibility for both types of Controller. This brings
2 Background

Figure 2.5: The BLE protocol stack [12]

a shortage of usage that when a device only implements standard Bluetooth cannot connect to the BLE and vice versa (this type of device is known as single-mode device). Meanwhile, the dual-mode device supports both types of Bluetooth so that it can connect to both devices for which each device can only implement single-mode [12]. Figure 2.6 depicts the difference between the Bluetooth device modes and Bluetooth versions.

Figure 2.6: Bluetooth versions and device mode [29]

From the Figure 2.6 we can see the differences between the modes in Bluetooth technology where it can be used as classic Bluetooth, single mode (BLE only) or dual mode (support both BLE and classic Bluetooth). Furthermore, as depicted in the picture that the dual-mode device has both types of the Bluetooth protocol stack (BLE and classic Bluetooth) so that this mode can support both types of Bluetooth modes. Also, since the physical layer is different between the standard Bluetooth and BLE so that on both versions can not directly communicate with each other.

The main difference between classic Bluetooth and BLE is the ability of BLE to provide an ultra-low power consumption that makes it an ideal choice to be used for data exchange in
2.3 Wireless Technology

a short-range communication for an extended period of operation. Thus, BLE is commonly used for many IoT devices and sensor nodes [10]. The other difference is the number of the channel provided by BLE is 40 channels with the size of the bandwidth is 2 MHz. In contrast, classic Bluetooth provides 79 channels with bandwidth 1 MHz each. There are two different types of channel in BLE: data and advertising channels. Data channels are used for communication between connected devices bidirectionally [12]. Meanwhile, advertising channels employ three channels out of 40 channels of BLE which are used for connection establishment, device detection, and broadcast transmission. To minimize the overlapping with other IEEE 802.11 channels, the three advertisement channels are placed at the center frequency of channels 37, 38, and 39 and are centered on 2,402 GHz, 2,426 GHz, and 2,480 GHz.

![Figure 2.7: The BLE channels](image)

Figure 2.7 shows the BLE channels with its two different types of channels. The darkest color channels depict the advertising channels, and light dark color channels represent the data channels. On each of these three advertising channels, it repeats each of advertisement in quick succession. The receiving device [30] adopts a scanning process that circulates over the advertisement channels. On top of the BLE protocol stack, as shown in Figure 2.5, lies some critical application profiles from BLE itself, such as Generic Application Profiles (GAP), Generic Attribute Profile (GATT) and Attribute Protocol (ATT). In the next section, these protocols are explained in more detail.

**Generic Access Profile**

Located on the highest level of the BLE stack, Generic Access Profile (GAP) defines procedures for the discovery of the device, services offered by the device, security, and connection establishment management, device modes and roles. Furthermore, on the underlying controller of BLE GAP, the roles of a device is specified into four types: Peripheral, Central, Observer, and Broadcaster. A device in peripheral type when it only supports a simple single connection whereas central type is complementary of peripheral where a device can manage and initiate multiple connections. Meanwhile, broadcast type means a device can only broadcast data without supporting connection mode. In contrast, observer type means a device only receives the data transmitted by the broadcaster. Moreover, in GAP the application profiles
also specify the interoperability between devices which produce by different manufacturers that are also done by Bluetooth SIG [12, 13].

**Attribute Protocol (ATT) and Generic Attribute Profile (GATT)**

Attribute Protocol (ATT) assigns the role of the communication between two devices whether a device works as a client or server. Meanwhile, the Generic Attribute Profile or GATT is used for managing a data structure which stores information of attributes. GATT sets a framework which utilizes ATT for the detection of the services and exchanging characteristics of the service, which include a set of data that defines values and properties of a service, from one device to another. For GATT, a concept of client/server is applied for the GATT role. GATT server means a BLE device which saves data locally and serves an access method of data access to a remote GATT client, whereas GATT client means a BLE device which accesses the remote GATT server’s data using four GATT operations. These GATT operations are known as notifying, indicate, read and write [13].

One application profile composes one or many services which in general determine the identification of the application in the server. Meanwhile, one service can have several characteristics where the actual value of data is saved in that it withstands some number of descriptors and a value. The descriptor itself means the thing that describes the characteristic value to make it easier for the user to understand. In characteristics, the four GATT operations are applied to each of them. One characteristic may have one or more GATT operations. [10, 29]. Figure 2.8 shows the hierarchy of the GATT profile.

![Figure 2.8: GATT data hierarchy](31)
2.3 Wireless Technology

BLE Operation

As aforementioned before that in BLE, there are two types of channels, advertising, and data. This also the same with the BLE packets. If the packets sent through the advertisement channel, it is called advertisement packets and vice versa for the data. Advertising packets provide the means to scan slaves and connect to them, and to broadcast data for apps for which the overhead of a full connection establishment is not needed. The size of the packets carried in advertising channels can be up to 37 bytes in addition to that the necessary header information. Meanwhile, the size of the packet carried in the data channel contains more data packet. In details, it may be up to 255 bytes. Figure 2.9 shows the BLE packet breakdown with its possible PDU.

The mechanism of the broadcast is proceeded merely over the air by the advertiser blindly or without the knowledge of the presence of any scanning device previously. Afterward, the advertising interval sets the fixed rate of time to send data that ranges from 20 ms to 10.24 s. This advertising interval may determine the frequency of advertisement packets are being sent. This value has a significant influence on the power consumption because it relates to the time the radio must turn on directly. It can be whether the advertiser wants to increase the probability of its packets being received by the scanner or it saves the power consumption by reducing the frequency of its advertising packets. The higher amounts of transmitted packets become, the higher the device power consumption itself. Meanwhile, the scanner can set the scanning interval of time ranging from 5 to 10 s. The more time used for the scanner to turn the radio on, the more significant of power consumption on it. Henceforth, the combination of the scan window and scan interval determine for how long and how often a device acts as the scanner listen for potential packets.

The scanner obtains an advertising packet successfully when they overlap randomly with each other. This is because of scanner and advertiser are not synchronized in any way, and there is the maximum of three channels available for advertising. Figure 2.10 describes the overlap between the scanner timing and the received advertising packet in an illustrated time allocation.

The scanning procedure done by the scanner is defined employing two specifications: active and passive scanning. The active scanning means a scan request packet is issued by the scanner after it receives an advertising packet. On the other hand, passive scanner defines the
2 Background

Figure 2.10: The illustration of time allocation for advertising and scanning in the BLE [29]

scanner to only listen for advertising packets while the advertiser is careless of the fact that there are packets that have been received by a scanner.

BLE Communication Modes

Since there are two different means of the channel in BLE, there are also two different modes for a BLE device to communicate with other devices: connecting and broadcasting mode. In connection mode, the data transmission can be done in two ways (as shown in Figure 2.11b that the arrows are pointed on both directions). A central or master device discovers for connectable advertising packets from nearby devices periodically, and when suitable, the connection is initiated. Meanwhile, slave or peripheral devices advertise packets periodically and accept if there is an incoming connection. Nevertheless, in broadcasting or connectionless mode, data in BLE broadcaster or sensor node are sent out to any nearby observer devices or receiver within the listening range. Thus, the transmission of data in broadcast mode is done in one direction only (as depicted in Figure 2.11a, the arrows are pointed towards observer devices only) [29].

Figure 2.11: The BLE operation modes [29]
One of the significant advantages of connection compared to broadcasting is the capability of organizing data exchange securely. Hence, it is also able to provide the data with much finer-grained control over each property or field through the use of GATT and additional protocol layers. Furthermore, connection mode may offer for much layered and richer data models, and also one of the most important, provide the potential to use much less power than its competitor due to the ability to extend the delay between connection events further out. Moreover, it can also push large chunks of data out only when new values are available. Instead, in broadcast mode, the advertisement of the packet is done in the full payload at a specific rate without the ability to know the subjects that listen to it and how often it has listened.

There are three important parameters for connection mode at master during connection establishment, such as:

- Connection interval defines the time between the start of two successive connection events ranging from 7.5 ms for higher throughput to 4 s for less power consumption.
- Slave latency determines the number of connection occurrence to which a slave device can opt to skip without the risk of disconnection.
- Connection supervision timeout specifies the maximum time between two valid data received before a connection is lost.

Universally Unique Identifier (UUID)

The services provided by a Bluetooth device use a unique identifier namely Universally Unique Identifier (UUIDs) which in detail depict the services and their attributes in a way that a central registration authority for registering service may not require. UUIDs use 128 bits (16 bytes) long to describe a service identity. However, since the data payload which can be sent via Bluetooth Link Layer is only 27 bytes per send, therefore there is a way to make the UUIDs shortened. Nevertheless, these shortened UUIDs can only be used if the UUID has been registered in Bluetooth Specification. Assigned by Bluetooth SIG, these shortened UUIDs can be either 32 bits or 16 bits long. Using the Bluetooth base UUIDs, these two formats of shortened UUIDs can be retrieved back to normal 128 bits UUIDs. The base of Bluetooth UUID is xxxxxxxx-0000-1000-8000-00805F9B34FB.

2.4 Related Work

2.4.1 BLE Wireless Technology

The use of BLE as wireless communication technology has raised the ubiquitous of some embedded computers used for sensor nodes. With the usage of the Internet, this has brought a new world of IoT. There are several works which utilize BLE as ultra-low power communication mechanisms to be used in IoT. Starting with [12] that reviews and evaluates BLE as a novel
low-power wireless technology. In [13], a comparison of several emerging wireless technologies is comprehensively discussed. Furthermore, the work [14] investigates the energy consumption of BLE compared to Zigbee. Moreover, [34] compares BLE with other sensor nodes by its RF power consumption in a cyclic sleep scenario.

### 2.4.2 BLE-Powered IoT Solutions

There are many projects demonstrate the implementation of IoT devices as their core services. One of the noticeably essential is [30] has investigated the use of BLE for indoor positioning. Furthermore, BLE is also used for location fingerprinting to increase the location-awareness in the indoor scenario [35]. Then, the work in [6] proposes an indoor air quality using its software built using the Android. Moreover, the project in [36] investigates real-time health monitoring and remote health care services. Moreover, in [13], a prototype of home automation of energy management system has been demonstrated using BLE as the wireless technology.

### 2.4.3 Public Sensing

In the field of WSNs where many sensor nodes gather environmental data in distributed locations are connected using wireless data collection networks to obtain smart environments [7, 3]. PS has gained some interest in a way that the data can be transferred from sensor nodes to the cloud via the Internet. However, there are some challenges in a way that the data can easily be transferred similarly to the scenario as mentioned earlier. These challenges have been mentioned in some works, such as in [3] which states the challenges in some hierarchy of WSNs, e.g., evaluation of information, data collection, and monitoring, UI design to provide meaningful data to the user, decision making, and alarm functions, etc. According to [4], the problem of this can be tackled by forwarding the data from sensor nodes to the cloud. This can be done directly from the sensor nodes by providing IP address or via the usage of an intermediate device as an interface for data with the Internet world known as the gateway. In the recent years, PS has been known as the smartphones’ utilization of mobile participants for obtaining sensor data [7]. An example of the usage of PS using a personal mobile device can be seen in iSense [37]. The iSense is a framework that uses WiFi Access Point and smartphones for participating in crowd-sensing sensory data collection. This work emphasizes more into reducing the overhead and relieving the energy burden on the mobile devices during the operation of sensing and reporting their position information back to the back-end server. Another example is DrOps [7] that remedy the efficiency of data procuration in PS. This work focuses on the use of model-driven data acquisition to keep minimum energy consumption by modeling to decide readings from the unavailable v-sensors. In the latter stage, because of the usage of the smartphone as a gateway in PS, or more known as Opportunistic Public Sensing (OppS), the term of the mobile gateway to connect sensor nodes with the Internet has known widespread. In this side, the smartphone is only used to relay data obtained by sensor nodes to the cloud server, instead of using it for localizing and reporting the position information that might consume more energy. Nevertheless, there are also some challenges to face to use the smartphone as mobile gateways, such as the siloed versions of applications in
the smartphone that is used for a mobile gateway for different types of sensor nodes, energy consumption for the usage of the smartphone as a mobile gateway, etc. The proposed system in our work tries to tackle some of these problems which are cumbersome to utilize PS using smartphone opportunistically.

2.4.4 Mobile IoT Gateway

There are many research works regarding mobile gateway for IoT devices. However, most of them concentrate on the usage of the mobile gateway and the challenges that occur when implementing a mobile gateway with smartphones. [25] is the only work that considers the energy consumption of mobile gateway itself. Despite the similarity, the authors in this work focus on the efficient routing method for data traffic to a mobile gateway in WSNs without considering several design aspects in the mobile gateway which can also be utilized to reduce the energy consumption. Meanwhile, [9] proposes an architecture using a smartphone-centric approach to leverage the number of BLE radios to connect IoT devices to the Internet. The proposed architecture also resolves the problem of siloed versions for connecting the cross-platform low energy devices to the Internet. However, this work does not provide an implicit mechanism for leveraging the mobile gateway to provide an efficient architecture. Furthermore, [6] proposes an algorithm for an approach to discover, negotiate, collect and deliver the sensed data in an energy efficient manner namely Collmule. However, this work mentions only the software prototype of Collmule and implement the algorithm in software simulation. Another discussion about research in this area can also be found in the thesis of [10]. In this work, the author mentions how to implement such a protocol of connecting BLE sensors to the IoT world. However, this work focuses on designing the protocol to improve the security during data transfer from sensor nodes to the mobile gateway and ended up in the cloud.
Chapter 3

System Model

This Chapter mainly discusses the proposed system model that harnesses smartphones to act as mobile gateways. First, the explanation about IoT sensors which are used as nearby IoT devices is given. Then, the discussion is moved to the mobile gateways followed by cloud services. Finally, the network topology of the proposed mobile gateway is also included in the discussion of this Chapter. Figure 3.1 depicts the proposed system model.
3 System Model

3.1 IoT Sensors

IoT sensors are devices which work as a sentinel to sense the environment for obtaining data from the real world. Multiple IoT sensors, which are located in distributed locations with different modalities, obtain these data from the sensory devices [4]. Accordingly, these data can be gathered by the smartphone using the BLE wireless connection periodically. Thus, as the sensors must present in the nearby of the smartphone for connection and data exchange using BLE before these data are sent to the cloud. As shown in Figure 3.1, the IoT sensors consist of many sensors that advertise packets using BLE as wireless connection technology. The usage of BLE for the sensors is critical due to the ability to use ultra-low power consumption. This can be done by switching the BLE modes between active and sleep modes periodically. Therefore, the sensors are capable of being used for a long time operation without having to change the battery routinely [13, 14]. There are several types of sensors that are used as the IoT sensors in this work, including:

![Figure 3.2: Example of the IoT sensors](image)

1. The sensor built using BLE module of nRF8001 and Faros Board as shown in Figure 3.2a. As a single-chip BLE Connectivity IC, nRF8001 combines a fully ready BLE version 4.0 Link Layer, Radio and Host stack with the simple serial interface so that a wide range of external application microcontrollers is supported. This board is also specifically designed for BLE applications operating as Peripheral/Slave role only. Regarding the currents which represents the power consumption in BLE Connectivity, the number of peak currents is as low as 12.5 mA and average currents for 1 s connection interval drops to 9 µA [35]. The sensory data in this type of sensor are stored as characteristics according to the GATT specification. The transmission of data exchange is done using ATT protocol in BLE. In this work, this sensor provides the temperature readings of the surrounding environment.

2. The sensor implemented in a wearable device, e.g., MiBand 2 produced by Xiaomi [15] as shown in Figure 3.2b. This device provides the readings of some characteristics, such as heart rate, steps counter, etc. The data are also exchanged using BLE protocol stack same as previously mentioned in BLE module nRF 8001.
3.1 IoT Sensors

(a) Revogi smartmeter  (b) Beewi temperature sensor  (c) Coospo cycling sensor

Figure 3.3: Example of smart devices

3. The sensor implemented in a smart device, e.g., a smart meter produced by Revogi [39] as shown in Figure 3.3a. This device provides the readings of some characteristics, such as power usage, power meter, timer, etc. Meanwhile, a device produced by BeeWi [40] known as SmartPad is also included in this example of the smart device depicted in Figure 3.3b. This device provides the readings of characteristics temperature and humidity from the surrounding environment. Furthermore, another smart device example is a cycling sensor produced by Coospo [41] as illustrated in Figure 3.3c. This device gives the readings of characteristics speed and cadence of a bicycle. Same as the other sensors, these smart devices also use BLE protocol for the data exchange.

4. The sensor which is simulated using a third party software in a smartphone which is separated from the smartphone used for the application layer mobile gateway. This software is known as nRF Connect which is built by Nordic Semiconductor [42]. This App is used to advertise BLE beacon packets and provide simulated data implemented into the services and characteristics for the BLE connection mode. Figure 3.4a shows the nRF Connect Advertiser screenshot.

(a) nRF connect advertiser  (b) Xperia Z5 smartphone

Figure 3.4: nRF connect and a smartphone
3.2 Mobile Gateways

Because of its abundance of wireless communication technologies and the ability to connect to the Internet, the smartphone is an ideal choice to be used as the mobile gateway. The smartphone which runs a modern operating system (e.g., Android, IOS, Windows Mobile or any similar mobile platform) can be installed by the third party App in which the developer can programme the application. A mobile gateway in this scope of work signifies an application layer built by the developer targeting the Android and installed to it in the form of APK file. This mobile gateway works as a forwarder of data from the sensors or sensor nodes to the cloud using internet connection.

In this work, the smartphone which is used for installing this application layer mobile gateway is a Sony Xperia Z5 as shown in Figure 3.4b. This phone has been introduced in the late of 2015 with an abundance of choices of wireless connection technologies. For this work, we use the BLE in this phone to connect to the nearby sensor nodes and the connection of WiFi or Cellular Network, i.e., 3G or LTE, for connecting to the cloud using the Internet.

Because of the usage of smartphone based on Android system, as aforementioned in Chapter 2 Section Android Software Stack, Android has a data storage using SQLite database—an embedded SQL database engine that contains serverless, a self-contained and transactional SQL engine—embedded in the Libraries in Android Software Stack. The main advantages of the usage of SQLite are SQLite code free for use for any purposes, either private or commercial use and the zero configuration of the server operations, such as configure, start and stop. Then, the most important is that SQLite supports Atomic, Consistent, Isolated, and Durable (ACID) as a transactional database even after power loss. This makes SQLite a robust choice for the database usage. Furthermore, SQLite operates the write and read operations directly to the ordinary disk files without using any separate server processes. The file format produced by SQLite is also a platform friendly which can be copied between 32 or 64 bit systems or between little and big-endian systems. This database also supports many regular database features, such as triggers, views, and indexes. The maximum database size of SQLite is about $2^{47}$ or equals to 140 terabytes. This makes SQLite can provide a quite large number of database storage performance [17].

SQLite is used in this proposed system to save data about the nearby devices information, services available on each device, and also the characteristics provided by the services in these nearby devices. This information is then linked to each other to make the connection between the entities and the relationship between one database to another. As shown in Figure 3.5 about the entity-relationship (ER) diagram used in this proposed system, there are four central databases resided on the smartphone side. First, a database to store information about available nearby IoT devices or BLE Device Data. The key for this database is the device id. The other entities in this database represent the properties of the nearby IoT devices, such as device name, mac address, RSSI, state, etc.

Secondly, a database which stores information about services available in the BLE device namely BLE Service Data. The relationship between this table and BLE Device Data is a 'has' relationship where one BLE device has one or more services. The key in this database
3.3 Cloud Services

is service id, and there is also a foreign key from BLE Device Data called mac address. Meanwhile, the other entities consist of service UUID, and timestamp.

Thirdly, a database that stores information about characteristics from the available services in the nearby IoT devices namely BLE Characteristic Data. The relationship between this table and BLE Service Data is a 'has' relationship where one service has one or more characteristics. The key in this database is characteristic id, and there is also a foreign key from BLE Device Data called mac address. Furthermore, the other entities in this database including characteristic property, value, UUID, and timestamp.

Lastly, a database which stores information about the BLE data in a Javascript Object Notation (JSON) format that is ready to be uploaded to the cloud. This database name is Upload Data. The relationship between this table and BLE Device data is a 'has' relationship where one BLE device will have one or more data based on the running timestamp. The primary key in this database is MAC Address of the device and the timestamp. Moreover, the other entities in this database are data and upload status.

3.3 Cloud Services

A cloud is a representation of this proposed system that will be used to store data from the smartphone and do the further processing of the data. In this work, we use the cloud storage provided by Firebase. Firebase is a web application development platform made by Firebase Inc in 2011 and acquainted by Google in 2014. Firebase provides many services for database post-processing, such as cloud storage, real-time database, and backend service,
Google Analytic for Firebase, etc. For the database infrastructure provided by Firebase, they offer a simple data storage without the setup of the own server and also privacy. However, the usage of the free service for the Firebase database is only until 1000 client accesses.

### 3.4 Network Topology

![Network Topology Diagram](image)

Figure 3.6: Network topology for the BLE connection

For BLE wireless connection, two of the most commonly used network topologies are broadcast and star topology. In broadcast topology, as shown at Figure 3.6 (b), one device acts as an advertiser and the nearby devices work as scanners that scan and obtain data from the advertisement packets sent by the advertiser. The exchange of the data in this topology can be done only in one way in which the advertiser sends data and scanners receive the data. Meanwhile, in the star topology, as shown in Figure 3.6 (a), one device acts as Master and several devices work as Slave. Master works as the central device that connects to several slave devices periodically. The connection to the slave devices is arranged in a way that the scheduling algorithm arranges it. For the data exchange in this topology, both parties can send or receive data, or the connection can be made in two ways.

The proposed system uses the star topology network to which the smartphone acts as the master and the nearby sensors work as the slave. This configuration is due to the usage of BLE connection mode that uses smartphone ability to connect to many nearby sensors sequentially or concurrently.
Chapter 4

Mobile Gateway Design

The primary goal of this Chapter is to give an insight into the system proposed in this thesis. Started with discussing the design challenges followed by the Android multi-threading mechanism. Afterward, the explanation about the Android Services will also be discussed before moving to the Inter-Process Communication. Lastly, the proposed design energy-aware mobile gateway will be discussed in detail in the last section of this Chapter.

4.1 Design Challenges

Compared to a standard personal computers (PC), notebook or desktop computers, mobile devices have many limitations regarding to their operation, such as relatively low processing power, limited random access memory (RAM), limited storage capacity, limited battery lifetime, high cost in data transfer and unreliable data connections, small screen size with lower resolution, etc. However, mobile devices offer many opportunities to be used in many applications due to the ability to make tailored application layers by doing software development.

Accordingly, to make use of the opportunities lies in a mobile device, i.e., a smartphone, by creating a tailored application, the application design have to face many challenges of limitations as mentioned before. One of the difficulties that may have significant influence is the limitation of battery life due to limited battery capacity in a smartphone. Although nowadays many battery manufacturers have successfully produced many batteries with higher capacity, there is still a challenge in the future for a more computing power which may consume more energy in a smartphone. Therefore, one way to suppress the energy consumption to be used in limited battery capacity is by building an energy-aware application so that the App consumes less energy and performs less overhead without sacrificing or reducing the outcome or performance of the application itself.

In accordance with [44], there are several essential methods which bring a better battery optimization in Android application development:

- Using Lazy First approach to build the Android App in which during the application development, the developers should try to optimize the usage of the operations which consume much power by reducing this kind of operations. Several ways to cushion software design using Lazy First are: Coalesce, Defer and Reduce. Coalesce means if it is possible to perform a batch operation from the works. Meanwhile, Defer implies if the
App is necessary to act immediately. Lastly, Reduce means if there are any redundant operations in the App which can be deducted.

- Using the advantages offered by the platform features to manage the consumption of the battery itself. According to [44], there are two clusters in the Android that can be used for optimizing the usage of the battery from the App installed in the Android: Firstly, using several APIs related to the intelligent Job Scheduling, e.g., AlarmManager, SyncAdapter, JobDispatcher, and Services. Secondly, applying several internal ways to the Android to conserve battery life, i.e., Doze and App Standby. A Doze mode means when the devices are left unplugged by the user for long periods of time with the screen off. The battery consumption is reduced using this mode because of the background CPU, and network activity is deferred if the App is obsolete for an extended period.

- Using some tools provided by Android Developer Tools, i.e., Battery Historian and Profile GPU Rendering, to identify some areas in the measuring device which can be optimized for longer battery life.

Meanwhile, the Android provides some use of threads so that the performance of the App in Android can be encouraged. Several aspects when working with threads may leverage the performance by describing some strategies to avoid the snare and some potential traps which may influence the performance. In [45], some discussion about the link between thread priority and the App lifecycle, working with main thread, and working with some methods to handle thread complexity is essential in connection with the performance for the Apps running in the Android. Better performance and responsiveness which are needed for good user experience can be achieved by making use of the threads or doing the multithreading mechanism [46].

In this thesis work, the combination of the multithreading mechanism and battery optimization approach are used to obtain an application which considers the low energy consumption and the best performance suitable to be used as a mobile gateway for IoT devices.

### 4.2 Android Multithreading

A software programming is aimed to work as an instruction to the hardware to undertake an action. In a computer, there are many actions performed to make use of it, and these actions are instructed by the codes which will be translated into instructions that are written using the software programming. These codes are processed by a Central Processing Unit (CPU) in sequential order or known as the high-level definition of threads [46]. Because of the underlying environment in Android is made from Java, then the thread in the Android environment is inherited from java.lang.Thread class. This is the most basic execution environment in the Android where it executes tasks when it starts and terminates when the task finished, the thread is interrupted or the tasks to be performed are void [29, 46].

In a Java-based programming environment, a thread provides execution for tasks using the interface of java.langRunnable implementation where the actual tasks will be wrapped up in a class inside this interface. The instructions written in the software programming can be
processed in the CPU from one thread at a time. However, there is also a system that runs multiple applications which run simultaneously. Such a system requires multiple threads to process various instructions at the same time. As shown in Figure 4.1, two threads execute four tasks simultaneously. From the user point of view, the applications can run in parallel by sharing the CPU’s processing time within the application threads by using scheduler.

![Figure 4.1: One CPU executes two threads simultaneously](image)

Furthermore, the other important aspect in the thread is its priority which determines the timing for execution where a higher priority-thread will have CPU allocation before a low priority-thread. In Java, this thread priority is assigned by number where 10 is the most top priority, and 1 is the lowest priority. Meanwhile, an Execution Interval defines the time used by a task to be finished and context switch explains about a time needed by two threads to change on each other. On Figure 4.1, the context switch is represented by C. The different use of the threads leads to the different type of application behavior that can be made using software programming.

In the case of the number of threads to be used, the application can be differentiated by the term of single-threaded application and multi-threaded application. These terms will be explained in detail in the following subsections.

### 4.2.1 Single-Threaded Application

A single-threaded application means the application that has at least one thread which establishes the code path of execution. Along with the same code path, all of the codes will be executed if there is no more threads are created. This means the execution is done sequentially where an instruction has to wait for the CPU to finish all previous instructions before it can be executed. The advantage of using this type of code is the simple approach of programming model with the execution that can be determined readily or deterministic of the execution order. Nevertheless, for an application that requires the instructions not to be postponed by preceding instructions, this type of code approach is not enough. Therefore, the multi-
threaded application will remedy the case of an application that requires the instructions to run almost at the same time without being postponed by the preceding instructions.

4.2.2 Multi-Threaded Application

Performing a multi-threaded execution to the application means the code of the application is split into several paths of code. Thus, the execution of the operations is done concurrently. This leads to a better performance of the App compared to the single-threaded application code. However, there is a price as the increase of the performance using this method. There are several disadvantages regarding the usage of multi-threaded mechanism, including increased resource consumption, a non-deterministic order of execution, increased complexity, and data inconsistency [26, 46].

Firstly, the usage of multi-threaded mechanism leads to the overhead in memory consumption due to the usage of the private memory area for saving local variables in method and exchanging parameters during the execution of it. Meanwhile, the processor usage is also increased because of the startup and demolish of the threads and for saving and loading the threads during the context switches. This means the more threads are created, and the more overhead and increasing of energy consumption will happen.

Secondly, the order of the code execution in the single-threaded code mechanism is done in a sequential order which means the pointer will go to each code line by line. This may lead to the easiness of debugging the program to fix the bugs and check the code order. In contrast, the multi-threaded mechanism may jump from one code in one side of the method to another side indeterminately. This leads to the confusion and unpredictable behavior and results obtained by the codes. Thus, the complexity of analyzing the behavior of the code will also increase, and multiple threads present uncertainty in execution.

Lastly, the multi-threaded mechanism also gives non-deterministic resource access when two or more threads share the same resource. Hence, when the execution is done, it is not known in which arrangement the threads will process and reach the resource. Moreover, if there are two threads, i.e., Thread 1 or t1 and Thread 2 or t2, running almost at the same time, there is no guarantee that t1 always runs before t2. And the term of context switch may happen in between the bytecode instructions so that it causes the shared resource dependent on the order of the execution which is in this case non-deterministic. Therefore, the shared resource may have an inconsistent outcome at the end of the execution. To remedy the non-deterministic result caused by multi-threaded mechanism, a method known as Thread Safety can be performed.

4.2.3 Thread Safety

As aforementioned in the previous section about the multi-threaded application that it is not an easy task to do multi-threaded programming because of several reasons. One of the main disadvantages of it is the threatens of correctness which leads to data inconsistency. The cause of it is because of concurrent access to the state in shared memory due to the same instance
of an object is being executed by multiple threads simultaneously. The result is either looking at the value of the state before it has been renewed or defecting the value. Hence, the term thread safety means maintaining the correctness of the state of an object when the object is accessed by multiple threads.

To obtain such a thread safety mechanism, when the same resource is accessed by multiple threads, it is essential to make atomic regions of code instructions which are always executed in sequence without mixing the other threads. This can be done by blocking other threads when there is a thread accessing an atomic region. This blocking can be released for another one thread if the current thread has finished accessing the atomic region. Thus, this atomic region is known as mutually exclusive due to only one thread can access this region at a time.

Such a mechanism can be done by doing the object’s state synchronization. By applying synchronization to a code that writes or reads any variable, in which it can be accessed by one thread while being changed by another thread — this is known as critical sections, the access to the state can be controlled. Accordingly, a locking mechanism to check if the critical section has been locked by another thread is a way to do the synchronization method. Afterward, this locking mechanism can guarantee the execution in a specific region, which is being locked by it, is done atomically so that one thread accessing a shared object at a time condition can be achieved.

In Java programming language, two locking methods can be used to guarantee an atomic execution is performed [46]:

- **Object intrinsic lock** means a locking mechanism using the keyword synchronized. By using an intrinsic object lock, the lock can be implemented in a specific section of the code, such as a method or an object. The execution in the critical section is exclusively done for only a specific thread while other threads — the threads which try to access the critical region while being used — are not allowed to use it until the lock has been detached using keyword notify.

- **Explicit Locks** mean a locking mechanism that is using either ReentrantLock or ReentrantReadWriteLock keyword. By using ReentrantLock, the critical sections are shielded by locking and unlocking regions in the code explicitly. It blocks all threads that are trying to execute a critical section if there is still another thread has already locked that region. This case is almost the same as the synchronized keyword. Meanwhile, ReentrantReadWriteLock is slightly different with ReentrantLock where ReentrantReadWriteLock does the blocking for writers versus writers and readers versus writers while it permits the reading threads to execute concurrently.

Providing a thread safe during parallel programming is very important. However, when designing a thread safety mechanism using keyword synchronized, it is also important to consider the possibility of deadlock. A deadlock means two or more tasks block themselves mutually. Such condition happens because of one or more tasks block each other to use the resource due to one task does not finish at the right time causing other tasks to wait for it for an indefinite amount of time. To avoid such a deadlock, it is essential to set a timeout time...
to ensure that the task will not be left running for such an indefinite amount of time which may cause the application to crash.

### 4.2.4 Android Main Thread

The Android performs the operations of UI objects of the App, e.g., establish, utilize and demolish the UI objects, on the main thread or known as UI thread by design. Thereby, the Main thread controls everything that occurs onscreen. However, Android View objects are not thread-safe by design due to many operations are performed together on the main thread. Furthermore, once the App is launched, a new Linux process therewith with an execution task is also created. Therefore, for designing an Android App with better performance, the knowledge of the usage of the threads, especially on understanding how the main thread works, is critical [45]. The misuse of the main thread leads to an App crash or unexpected outcome shown on the screen.

Specifically, it is vital to move tasks which require longer execution time from the main thread so that these tasks do not disturb with fast responsiveness to user input and UI rendering. The cause of this is because the Android system attempts to execute the drawing of the screen every 16 ms in accordance to comply with 60 frames per second requirement for smooth UI rendering. This is done on the main thread. Hence, if the main thread cannot finish executing blocks of rendering work within 16ms, a user may experience lagging, lack of UI input responsiveness, and hitching. Moreover, if the blocking of the main thread happens for more than 5 seconds, the Android system shows a dialog of Application Not Responding (ANR) [26, 45].

### 4.2.5 Android Helper Classes of Thread

As aforementioned in the previous section that the Android framework is inherited from Java software programming. Therefore, some of the primitives classes and interfaces, e.g., Thread, Executor, Runnable, etc., are also available. Afterward, the Android framework provides also some helper classes of Thread which are designed to reduce the cognitive load regarding the development of multi-threaded application in Android, such as AsyncTask, HandlerThread, and ThreadPool. Such helper classes will be discussed in the following subsections.

**ThreadPoolExecutor**

This class is used when there is a work that the workload can be reduced if the tasks inside this work are highly parallel and distributed. This class provides the management of a group of threads creation, set the threads priorities, and the management of distributing works throughout those threads. Before using this class, it is required to set an optimum number of threads that will be used. In the constructor of a ThreadPoolExecutor object, the number of minimum and maximum threads are requested.
Then, this class adjusts the required number of threads based on the input number of minimum and maximum threads with consideration to the workload needed for the tasks to be done. In the beginning, ThreadPoolExecutor initializes the threads with a minimum number of threads given in the constructor. As the workload increases and the number of pending tasks, the threads number will then be adjusted. To adjust the threads in considering to the change of the workload, this class manages to establish or demolish more threads automatically without the requirement of the programmer to manually load balancing the threads [10]. Using ThreadPool serves not only many significant advantages but also several problems, such as a number of threads that are used can significantly affect the performance and power consumption of the application. Therefore, it is essential to give a right amount of maximum threads (not too many but also not too less).

**HandlerThread**

The HandlerThread class is used for a long-running thread which obtains work from a queue and performs some operation on it. However, it is essential to set the thread’s priority concerning the type of work that the App does. This is due to the limitation of CPUs in Android devices that are only able to handle a slight number of threads in parallel so that by the determining the priority, the Android system knows the right way to schedule the work despite all other threads are also struggling to get attention.

**AsyncTask**

This class is a useful and straightforward primitive for the Apps which require a quick move to worker threads from the main thread before giving back the result to the main thread to be displayed at the UI. The AsyncTask objects use a single thread and serially execute the tasks so as the main thread. Thus, when it does a long work packet, this class can block the queue resulting in an undesirable outcome that leads to ANR. According to [15], it is essential to use this class only for handling the work items with a duration not more than 5 ms.

4.2.6 Thread Design Strategies

For creating an application for the mobile gateway with good performance while is also still maintaining low energy consumption is the main goal of this thesis work. Therefore, the application should be planned with consideration of thread creation and task execution. According to [16], there are two suboptimal extremes and designs, including:

- **One Thread per task** design means a new thread is always created and terminated for each task executed from work. The overhead of the thread establishment and demolishing may degrade performance while is also increasing the energy consumption, especially if the tasks have a short lifetime and are established frequently.
• **One Thread for all tasks** design means the execution of all tasks are done on the same thread which often fails to use available processors and resulting in an unresponsive application.

Another application design consideration is about the execution scheme which can be divided into two methods:

• Concurrent execution means the execution of tasks are done in interleaved and parallel resulting the design of an application to become not a thread-safe so that it is required to use synchronization. However, the best performance can be achieved with many tasks are executed in parallel. This also leads to consuming more energy due to the usage of more memory and CPU power.

• Sequential execution means the execution of tasks is done in a sequence. This brings the tasks’ execution interval does not overlap resulting in a thread-safe design and less energy consumption due to the execution is done in only one thread. Nevertheless, the throughput resulted from this method is comparably low to concurrent execution and the needed for dependency on previously executed tasks.

The strategy here to obtain our primary goal is to make an effective multi-threading mechanism. This can be achieved by utilizing both sequential and concurrent execution together depends on the tasks given to the program. The key is the tasks which require an ordering or share a common resource can be executed using sequential execution while the tasks that are isolated and independent should be performed using concurrent execution [26].

According to [46], true concurrency cannot be obtained if a number of executing threads surpasses the number of processors. Although, the code in an application can be separated into several code paths so that the concurrent execution sight can be achieved with the multi-threaded mechanism. To perceive concurrent execution, one of the less-stressful ways is using ThreadPoolExecutor and set the number of lower bound of pool size by N —known as a number of the available processor from the runtime class in the Android— so that it is sufficient for independent and nonblocking tasks. However, for the upper bound of pool size, several suggestions have been proven theoretically and empirically, such as in [47] which suggests N+1 threads for compute-intensive tasks and in [46] that gives 2*N threads for well-performing computation.

### 4.3 Android Service

In the Android, to perform a long running operation without having to utilize the user interface (UI), there is an application component, namely **Android Service** which can be used by extending the intended class to the Service class which has been provided by the Android. According to [44], Android Services is one of the API included into intelligent Job Scheduling which helps to reduce the energy consumption. When a component in Android wants to interact with the class extended with a Service, it uses the term of binding to this service
4.4 Interprocess Communication

or performs an *Inter Process Communication* (IPC). Specifically, there are three types of Services in Android, including [26][27]:

- **Background Services** means the operation is not noticeable by the user or it is run with a limited interaction to which most of the lifetime is spent hidden. Instead of waiting to the user interaction, a background service listens to the actions caused by the hardware, system, other Apps or messaging. The common example for it is the App that operates with the database transaction which is not visible to the user directly.

- **Foreground Services** means the operation is noticeable to the user by displaying a notification, data and performing some graphics on the UI. When the application is not visible, then it is effectively suspended. For instance, the App of music player, games, etc.

- **Bound Services** means a service which is bound by an application component by calling `bindService()` and allowing the components to interact with it, send requests, receive results, and performing IPC. There are multiple ways to bind to a service: by using bind to serve as one component or various components bind to the service at once depending on the aim of the operation itself.

### 4.4 Interprocess Communication

An interprocess communication (IPC) is a communication happened across the process boundaries. An application in Java or Android is usually using threads that often establish communication within a process while also sharing the process’ memory. In the Android, IPC is also supported by using *binder framework*. The difference using this framework is there is no shared memory area between threads while also still carries out the data transactions. There are several mechanisms for threads in the Android to do IPC, such as Binder Framework and AIDL [26][48][46]. In the following section, both mechanisms will be explained in detail.

#### 4.4.1 Binder Framework

In Android, the usage of the *Binder Framework* is to enable the movement of data and functions employing method calls between threads running in different processes. In the client process, the threads may access the remote interface through the remote object in the server process. The class `android.os.Binder` in the Android supports the remote interface which is defined by the server process.

A transaction means a remote procedure call which displaces data and a function. Figure 4.2 shows the process of the transaction using IPC through binder framework. There are some processes that encompass a transaction mechanism: Firstly, when the client calls method `transact()`, the thread is blocked until `onTransact()` has completed executing on the remote thread. In this method, the arguments and return values are moved as `Parcel` objects — known
4 Mobile Gateway Design

Figure 4.2: IPC performed by binder framework

as `android.os.Parcelable` interface in Android which does Marshall and unmarshalling efficiently—.

In Binder Framework, the IPC can be done in bidirectional (or a two-way communication mechanism) means that the server process releases a transaction to the client process and returns the flow. Thus, the former server process changes into the client process and performs a transaction on another binder framework which is implemented by the previous client process. Moreover, an asynchronous transaction is also supported by binder framework where `transact` is called and is returned right away by client process thread. In this case, `onTransact()` will be called by a binder framework continuously on binder thread in the server process. Nevertheless, it is not possible to return any data to the client thread synchronously.

4.4.2 Android Interface Definition Language

As an IPC implemented in the Android, an Android Interface Definition Language (AIDL) permits the developer to stipulate a programming interface to which both server and client agree to communicate each other. It is the server that defines an interface of methods in which the clients can call. AIDL is described in a .aidl file in which the compilation of it produces a Java code that supports IPC. The application only requires to consider the
4.4 Interprocess Communication

interface, while Android Apps intercommunicate with the generated Java code [10]. Figure 4.3 shows the mechanism of the remote communication interface using AIDL.

AIDL looks like the other usual IPC that has been implemented in Java environment. AIDL is also used to create a bound to Android Services. However, the difference of AIDL and Bound Service using Binder Framework is the Android Service that uses AIDL can be used in many other clients, instead of using Bound Service for which only one client can connect to the Android Service.

To perform a remote procedure call, AIDL generates two inner classes namely Stub and Proxy. A stub is located inside the server process for which the marshaling and unmarshaling of the data and the transaction in server side is handled. Meanwhile, in the client side, Proxy handles the operations same as Stub does in server side. In the end, Java code is produced from the creation of AIDL in which the code establishes the communication contract and encases the binder framework. This process is illustrated in Figure 4.4.

According to [48], the method calls to an AIDL interface are direct function calls. Therefore, there are several thread usages in which the call occurs influenced by whether the call is from a thread in the remote or local process. To be detailed:

- Calls from a remote process are sent from a thread pool in which the platform holds inside its process. It may make some incoming calls from unknown threads for which there is a possibility of multiple calls happening at the same time. Therefore, it is important to keep the implementation of AIDL to be thread-safe completely.

- Calls created from the local process are performed in the same thread which is establishing the call. Hence, if it is done in the main thread, this thread keeps on to perform in the AIDL interface.
The behavior of remote calls are modified by the `oneway` keyword that a remote call does not block, instead it simply dispatches the transaction data and gets back immediately. Thus, if `oneway` keyword is used with a local call, no impact is detected, and the call is done synchronously.

Another important thing when using AIDL is the supported data types which can be used for RPC. The data types in AIDL support only several primitive types of Java programming language, e.g. `integer`, `long`, `character`, `boolean`, and `string`. Moreover, several special data types in Java, such as `CharSequence`, `List` and `Map` are partly supported. For List and Map, all elements must be one of the primitive data types which are also supported by AIDL as previously mentioned [28 48]. However, for an additional data type which wants to be included in AIDL which is not listed as primitive Java programming language, an import statement must be contained in the .aidl file, and this data type must also be made into a parcelable object [48].

In our implementation of energy-aware mobile gateway for IoT devices, AIDL is used to access the Android Service for the class of GatewayService.class in which this class is implemented for making a Service Interface design for an open gateway model. This will make the primary services offered by our proposed model can be accessed remotely by using IPC. Moreover, in this thesis, there are several models of implementations of AIDL which include the calls sent from the thread pool and local process.

### 4.5 Energy-Aware Mobile Gateway Design

Our proposed system design for an energy-aware mobile gateway for IoT devices can be seen in Figure 4.5. The components that are underlying our proposed design, including Service Interface; System Settings; Gateway Controller; Self Adaptation System; Scheduling & Multi-Criteria Decision Making (MCDM); Gateway Service; Discovery & Connection Manager; Node Repository; Power Estimator. All of this components will be explained in detail in the following section.
4.5 Energy-Aware Mobile Gateway Design

4.5.1 Service Interface

A Service Interface is a component which provides a UI layer of services and characteristics offered by nearby IoT devices. This layer will be triggered when there are at least one or more services that qualify the data from the list of the supported manufacturer in the system settings. This means that in the service interface, there should be a cooperation between the manufacturer and our proposed system so that the known services available in the BLE device produced by a manufacturer can be displayed in the services tab UI correspond to the service interface component.

4.5.2 System Settings

In this proposed mobile gateway, the settings are also provided on the component System Settings. This settings are written in XML files namely Settings.xml and Manufacturer.xml. These files are located inside the Android system asset folder. There are several settings of the system that are parameterized using these XML files, such as:

- **Default scheduling algorithm** provides a data of the default scheduling algorithm that can be used in the first cycle of the program and the latter case, it might be changed concerning several parameters defined by the MAPE algorithm.
• **Time based constants** provides time parameter data which is used to run the scheduling algorithm’s main operation, e.g., Processing Time, Scanning Time, Scanning Time 2 and Time Unit.

  – Processing Time is used for a timing constraint used in several scheduling algorithms to define time required for one cycle of the program to be done.

  – Scanning Time and Scanning Time 2 define the time required for discovering nearby IoT devices. The difference between these two entities is that Scanning Time is used globally (most of the scheduling algorithms’) on their first cycle. Meanwhile, Scanning Time 2 is used for the 2nd iteration of several scheduling algorithms which employ a database of Node Repository to suppress the energy consumption.

  – Unit is used to define the unit of the time from the given parameters written in this system settings.

  – Mape time start determines the first delay of starting the self-adaptive system using MAPE.

  – Mape time defines the time used for another run of the self-adaptive system using MAPE after the first delay which is written in the MAPE time start.

• **Power measurement constraint** provides the constraint of power usage, which is measured during the connection to a BLE device, used by the MCDM algorithm to calculate cost efficiency of a connection to the certain BLE device. This parameter consists of Battery Level, Battery Level Upper, Threshold 1, 2 and 3. Battery Level and Battery Level Upper show the minimum and maximum constraint of the battery parameter. Meanwhile, Threshold 1, 2 and 3 determine the values of the power consumption that become the constraint which is measured using Power Estimator component.

• **Self Adaptive System** provides the parameter for turning on or off the self-adaptive system using MAPE data and the upload data to the cloud.

• **Supported manufacturer data** provides the data of the manufacturer from the nearby IoT devices that use BLE as their wireless communication. For this settings, the Manufacturer.xml is used as the XML file. The manufacturer data consist of manufacturer id, manufacturer name, and services offered by the IoT device.

### 4.5.3 Self Adaptive System using MAPE

To guarantee a low energy consumption while also still maintaining a high throughput value, a self-adaptive system is also implemented in our proposed system model. The self-adaptive system will ensure that the intended system behavior will be controlled by several criteria which determine the current gateway system status. These criteria consist of the number of available BLE devices based on the BLE scan result and the percentage of remaining battery power at the device. Based on these criteria, the self-adaptive algorithm will decide two things, including which scheduling algorithm is suitable for the current situation and whether
4.5 Energy-Aware Mobile Gateway Design

the system should upload the data to the cloud —if there is still a data in the database that has not been uploaded— or not.

There are two approaches of the self-adaptive system that can be presented to the software system [49], including internal and external approach. The internal approach uses the adaptive logic for managing system that is be tied together with the core application as a managed system. In contrary, the external approach uses the adaptive logic which is detached from the core application. Thus, the external approach brings through the possibility of some software qualities, e.g., modifiability and reusability, without causing the adaptive logic challenging to be evolved, preserved and reused because of its adaptive engine is dependent to the core application.

The external self adaptive used in our proposed system is based on the Monitoring, Analyze, Plan and Execute (MAPE) which was proposed by IBM architecture blueprint for modelling the feedback control loops [50, 49].

Figure 4.6: The self adaptive system based on MAPE [49]

Figure 4.6 shows the self-adaptive system based on MAPE which is implemented in our proposed work. As shown in Figure 4.6, the system differentiates the loop into four parts which share knowledge, such as [50]:

- **Monitor function** serves the mechanisms to gather, aggregate, refine and inform the details obtained from a managed resource. The resources in our proposed design contain the number of nearby IoT devices and current battery percentage in the Android device.
4 Mobile Gateway Design

- **Analyze function** gives the mechanism to relate and model complex situation. In this function, learning of current situations and prediction for future decision are done comprehensively. For this system, our proposed design uses a fuzzy set in prediction.

- **Plan function** sets the mechanisms to build actions which are required to obtain objectives and goals. In our work, this function includes the action of selecting a suitable scheduling algorithm and the action of selecting to upload the data in the database to the cloud or not.

- **Execute function** provides the mechanisms which supervise the execution of a plan in consideration of the dynamic updates.

4.5.4 Gateway Controller

![Sequence diagram of proposed mobile gateway](image)

Figure 4.7: Sequence diagram of proposed mobile gateway

As depicted in Figure 4.5, a Gateway controller works as the starting point of our proposed mobile gateway application design. This component controls the implementation of the scheduling algorithm which may also be combined with the Multi Criteria Decision Making (MCDM), and a self-adaptive system using Monitoring, Analyze, Plan and Execute (MAPE) in order to select which scheduling algorithm that suits the current situations based on some criteria &
also decide whether the data in the database should be uploaded to the cloud or not. Furthermore, gateway controller works also to control the start and stop of the Android Service implemented on the Gateway Service component, before the interface in gateway service using AIDL can be used.

Figure 4.7 illustrates the overall picture of a sequence diagram of our proposed design. Firstly, a user opens the application of the mobile gateway. Then, the UI layer of the application will be shown while also the system checks the requirements of some peripherals to be activated. Afterward, the UI layer triggers the gateway controller automatically to enable the gateway service component for running the IPC using AIDL. Once the onServiceConnected() method is triggered, which means a sign that the Android Service has been connected, a new thread that runs scheduling algorithm and also a new Thread using ScheduleThreadPool for the self-adaptive system using MAPE are created. This means the multi-threading mechanism is implemented to perform both tasks almost at the same time. The scheduling algorithm runs with the thread priority high to ensure it is prioritized more than the other threads. Meanwhile, the self-adaptive system runs periodically based on the given certain time on the system settings. Once a new algorithm is obtained, the thread for running the scheduling algorithm will be interrupted, and a new algorithm will be executed by the gateway controller.

Afterward, the service interface component is triggered to operate when at least one BLE device has been connected, and the data has been obtained from it by the use of the scheduling mechanism. This will trigger the UI to show the services tab and when the user click this tab, the list of connected devices, in which the services available in these devices are known or listed in manufacturer list in the settings.xml, will be shown. Afterward, when the user clicks one of the selections on the list, a generic UI gateway will be demonstrated in that it displays the data gathered from the connection to a nearby IoT device. Nevertheless, if the service interface does not have a generic UI to view for a known device, an option to upload the data to the cloud will be offered.

4.5.5 Node Repository

This component is used as a data storage from the nearby sensor nodes which are obtained using BLE scanning and connection. The data is stored using the SQLite database implemented inside the Android. As previously mentioned in Chapter 3 Section Data Storage in Smartphone, SQLite database is an embedded SQL database engine that contains serverless, a self-contained and transactional SQL engine which is integrated into the Libraries in Android Software Stack.

As shown in Figure 4.5, for doing create, read, update and delete (CRUD) operations of the database, there are several gateway interfaces which are used as an abstraction for the peripheral communication which is done by the Gateway Service. Thus, for doing CRUD operations —which are later to be wrapped into Java code of methods as listed in the interface of the Gateway Service .aidl file— the interface of IGatewayService should be used.
4.5.6 Gateway Service

A Gateway Service is used to run the IPC mechanism for providing a programming interface in the proposed system by means of using the AIDL method. As shown in Figure 4.5, this component is located as the heart of our proposed system model where many of the interfaces for services, e.g., the necessary database operations, hardware properties reading, start and stop scheduling system’s, etc., that are used throughout the system are programmed using AIDL so that the other processes or threads may also use these services and bind to the Gateway Service via AIDL. The interface in the Gateway Service also works as an abstraction of the many Android peripheral operations, such as scanning BLE devices under the component Discovery Manager, connection to BLE devices using Connection Manager, connection to the Firebase cloud using WiFi or Cellular Connection, and database operations under the component Node Repository.

The Gateway Service also uses a component in the Android namely Android Service to perform a long-running operation in the background without having to utilize the user interface (UI). Thus, this Gateway Service class also complies with the intelligent Job Scheduler for optimizing the battery usage as mentioned in [44].

4.5.7 Discovery & Connection Manager

Discovery Manager component is used for discovering the available nearby IoT devices, scanning and obtaining some data about these BLE devices information, i.e., device RSSI, device MAC Address, services available, device manufacturer, etc., resulted from the scanning operation. This scanning operation is one of the sources for the energy consumption due to quite intensive use of the Bluetooth radio. Therefore, it is essential to reduce the timing usage of it to reduce the energy consumption of the proposed application layer design [29, 34].

Meanwhile, Connection Manager is used to connecting to the nearby IoT devices and gather the services and characteristics data during this connection event. The kinds of data that are collected during this connection event consist of service UUID, characteristic UUID, characteristic value and characteristic property. These data are available on the GATT Profile located on the top section of the BLE protocol stack as mentioned in Chapter 2 Section BLE.

4.5.8 Power Estimator

Power Estimator is a component of our proposed design which gives the information about the real-time power consumption consumed by the Android device. This component makes use of the battery level measurement and charging state which are available on the Android [51]. Several parts of the measure can be performed using the Android battery level measurement, such as battery remaining state, instantaneous battery voltage, instantaneous battery current, and many more.

The usage of this component is mainly used by Scheduling Algorithm and MCDM component. This is because, in this component, there is a measurement of real-time battery state
4.5 Energy-Aware Mobile Gateway Design

and power usage constraint which later will be used to measure current power consumption consumed by the system during the connection to a specific BLE device. These criteria will then become one of the factors for the decision making of the connection priority for this particular BLE device. The data of power consumption measured by this Power Estimator component will then be saved in the Node Repository so that if the Scheduling & MCDM is accessing the database of BLE Device Data using IGatewayService interface, the power consumption data has already available in there.

4.5.9 Scheduling

The Scheduling in our proposed design system as depicted in Figure 4.5 is a component that is located under the Gateway Controller component. It binds to the Gateway Service and uses methods which are wrapped into Java code from the AIDL implementation in the Gateway Service. These methods consist of the scanning and connecting to the BLE devices which are abstracted by the Gateway Service from the components namely Discovery and Connection Manager. These operations are then arranged in a way that the allocation of each task, e.g., scanning, connecting, connection timer, disconnecting, etc., are based on a predefined algorithm which is known as a scheduling algorithm.

There are several kinds of connection which are underlying the base of the scheduling algorithms which are used in our proposed system namely synchronous and asynchronous communication. Synchronous communication is the communication where the processes of receive and sending data to communicate at a particular predefined position in the flow of the program resulted in a connection that waits through blocking mechanism. In contrary, asynchronous communication is a communication that uses a buffer of data during the connection resulted in a non-waiting of times connection.

The scheduling algorithm based on these kinds of communication is more merely used for some applications. Lately, the synchronous communication is known as semaphore algorithm. Moreover, the asynchronous communication reserves many ways to be used as the scheduling algorithm. As mentioned in the work of [52], the usage of asynchronous communication for a better Quality of Service (QoS) capabilities that assist latency control and time allocation. In the asynchronous communication, there are several forms of intelligent scheduling algorithm which may bolster the QoS capabilities, such as Round Robin, Exhaustive Polling, and Fair Exhaustive Polling. These scheduling algorithms will be explained in more details on the following subsections.

Semaphore

Semaphore algorithm or abbreviated as SEM is a scheduling algorithm which uses the basic of synchronous communication. This communication waits for the previous communication until the previous one is terminated by the client device. Thus, the time that is used for communication depends on the connection time allowed in the client device.
Figure 4.8 shows the sequence diagram from the scheduling mechanism using semaphore algorithm. As depicted in Figure 4.8, the semaphore algorithm will first turn on the scan to gather the available nearby IoT devices (the sensors). This signifies the first cycle of the scheduling mechanism. The scan duration will be based on the system settings’ Scanning Time parameter. During the scan duration, scan callback will be received by the smartphone which resulting in a BluetoothDevice object. This object contains some information about the BLE device, such as MAC Address, RSSI, Tx Power, etc. Then, after the Scanning Time has been reached, the stop scan is triggered followed by populating the scan result into a list of BluetoothDevice.

After that, the connection to the sensors is made with BluetoothDevice data in the list one at a time. As shown in Figure 4.8, the smartphone will first send the connect packet to Sensor1. Because of the usage of semaphore algorithm, which utilizes Object.lock() to lock the current thread and prevent it to perform the next code execution, the smartphone will wait until the Object.notify() is triggered. The Object.notify() will be triggered after the callback or disconnect packet is received by the smartphone.

After Object.notify() is triggered, the cursor of the program moves to the next code execution. In the Figure 4.8, the next code executes the next connection to Sensor2. The previously mentioned mechanisms will be repeated in here. After some time, the smartphone will make the service discovery which resulting the smartphone to receive data from the
Sensor1 and Sensor2 concurrently. These data will then be stored into the database by considering that if a data has not been stored in the database, then the insert operation will be done. In contrast, if the data has been stored, the update operation will then be performed. After a while, the connections may then be terminated once one or more sensors send disconnect packet to the smartphone which indicates the limit of the connection time using BLE has been reached. After all, connections have been terminated, a new cycle will be executed, and the same processes will again be performed for the semaphore algorithm repeatedly.

The advantage of this algorithm is that the data gathered by the system may be obtained as many as possible due to the concurrent connection resulted in the simultaneously collecting data from several IoT devices. Thus, a high amount of throughput can be achieved by using this algorithm. However, there are some disadvantages of the usage for this algorithm that because of the concurrent approach used in this algorithm, the energy usage may be slightly higher than the sequential way and the fairness of the connections to BLE devices cannot be guaranteed.

**Round Robin**

According to [52], Round Robin algorithm or abbreviated as RR algorithm is also known as a limited service system where the slaves are polled in consecutive order by the master. The RR algorithm assures the master that the different bandwidths to different units can be done by allowing different timing to the various devices which are resulting in a fair share of the excess bandwidth so that the maximum fairness can be guaranteed.

Figure 4.9 shows the sequence diagram from the scheduling mechanism using a round robin algorithm. As what the semaphore algorithm does at the beginning, the RR algorithm turns on the scan for obtaining the available nearby IoT devices (the sensors) which will be performed for a specific duration as written in the Scanning Time parameter in the system settings. The result of the scan is the BluetoothDevice object, and this object will be populated on a list of the scan result after the stop scan is triggered.

Nevertheless, the difference between semaphore and RR algorithm is the usage of a timer for each of its cycle (or later known as cycle time) to be determined in system settings. Then, from the list of BluetoothDevice, the number of available BLE devices may be identified and from this number, a connection time for each device can be calculated by:

\[
\text{ConnectionTime} = \frac{(\text{CycleTime} - \text{ScanTime})}{\text{NumberOfDevices}}
\]  \hspace{1cm} (4.1)

Thus, if the cycle time is 60 seconds, scan time is 10 seconds, and the number of available BLE devices is 5, then the connection time for each device can be calculated using Equation 4.1 as follow:

\[
\text{ConnectionTime} = \frac{(60 - 10)s}{5s} = 10s
\]
Figure 4.9: Scheduling with round robin algorithm

After the connection time has been determined, the list of BluetoothDevice will be used to arrange the connection to the sensors one by one sequentially. Firstly, the smartphone will send the connect packet to Sensor 1. After some time, the callback packet will be forwarded by Sensor 1, and later after discovering the services, the data of the services and their characteristics will be received by the smartphone. After the connection time has been reached, the disconnect command will be sent via a packet from the smartphone to sensor and sensor will send back the disconnect acknowledgment packet signaling that the connection between the two entities has been terminated.

Afterward, the cursor of the code moves to the next code execution which is in Figure 4.9, the smartphone connects to the Sensor2. Then, the same process of connection will be performed. However, if the disconnect packet is sent by the sensor before the connection time has been reached, as shown in Figure 4.9 that the Sensor2 receive a disconnect packet, the connection between these two entities will also be terminated. After the Cycle Time has been reached, a new cycle of the algorithm will happen, and the same processes will again be performed for the RR algorithm repeatedly.

The advantage of using this algorithm is the fairness of the connections to several BLE devices can be guaranteed by the balance of calculation of connection time as previously mentioned. Furthermore, the real concurrent of connections between BLE devices can be avoided in here so that the energy usage of connection may be slightly reduced. However, this
timing based connection time may also contain a disadvantage, including decreasing some of the throughput numbers because of connecting to a device has not finished until the timer has reached the connection time. This is due to the difference in time needed to connect to different BLE devices.

**Exhaustive Polling**

Exhaustive Polling algorithm or abbreviated as EP algorithm is an intelligent scheduling algorithm which is based on the semaphore scheduling algorithm with some additional of

---

Figure 4.10: Scheduling with exhaustive polling algorithm

Exhaustive Polling algorithm or abbreviated as EP algorithm is an intelligent scheduling algorithm which is based on the semaphore scheduling algorithm with some additional of
pieces of stuff that bolster the energy efficient mechanism. According to [52], an Exhaustive Polling algorithm is a master continuous to poll the addressed slave until the queue is emptied. This means that somehow the result of the scan from the master (the smartphone) must be put in a queue and then this queue should be continuously polled.

In this proposed design system, a node repository component (database) is used to make a queue of data from the nearby IoT devices. Henceforth, this queue will be used to bolster the energy efficient mechanism by reducing the Scanning Time for the cycles after the node repository has been fulfilled with BLE devices data. Figure 4.10 depicts the sequence diagram of the scheduling with the EP algorithm. As aforementioned at the beginning of this subsection that this algorithm is inherited from the semaphore algorithm so that the connection method also uses the concurrent mechanism. Therefore, as depicted in Figure 4.10, the first cycle of this algorithm is performed same as the semaphore algorithm does where the first cycle in here is assumed that the node repository or database is still empty.

Nevertheless, in the second cycle and so forth, there are some slight changes due to the usage of the database as a queue for the information of nearby IoT devices that have been scanned before. Due to the usage of this queue, the scan time can also be reduced in which it is safe to assume that for the devices which have been scanned, the data of scan callbacks from them have been stored in the node repository. Hence, it is not necessary to scan again and again these devices. Instead, the relatively shorter scan can be performed to discover BLE devices that have not been scanned before. Overall, for the connection to BLE devices is still the same with the semaphore algorithm. The term Object.lock() is still used to make sure that the execution of the code is halted until the Object.notify() is triggered during the callback given by the sensor.

The advantage of this algorithm is the continuous of the master to poll the addressed slave from the queue (in this case the master uses database information to find the already known BLE devices) which is needed to save some energy from the scan mode in BLE device by decreasing the time for scanning. Nevertheless, this algorithm according to [52] is still not an optimal one because of this mechanism will benefit slaves for generating packets at a maximum rate. Hence, a parameter for limiting the processing time to some predetermined maximum time is required to be added.

**Fair Exhaustive Polling**

As another intelligent scheduling algorithm, a fair exhaustive polling algorithm or abbreviated as FEP is a scheduling algorithm which combines the last two earlier intelligent algorithms. The basic method of FEP is to poll slaves which has nothing to dispatch as rarely as possible [52]. To build such an algorithm, a mechanism of using the two slave device states based on the ability to connect and obtain data is implemented in FEP. These two states are active state and inactive state. The main difference between these two states is, a slave that has no information nor data to be sent or cannot be connected is considered as inactive slave. In contrast, the slave that performs the other way around is judged as active slave.
4.5 Energy-Aware Mobile Gateway Design

Figure 4.11: Scheduling with fair exhaustive polling algorithm

This algorithm also utilizes the node repository for queuing data from the already scanned BLE devices. Thus, the energy efficient mechanism by reducing Scanning Time is also employed in this algorithm. Figure 4.11 illustrates the sequence diagram of the scheduling using FEP algorithm. As shown in Figure 4.11, the first cycle of this algorithm is the same as the RR algorithm. As what the RR algorithm does at the beginning of the cycle, the FEP algorithm turns on the scan for a certain amount of time for discovering the available nearby IoT devices while also stores the scan callback to the database. The BluetoothDevice objects resulted from the scan callbacks of adjacent BLE devices are then populated on a list of the scan result. After that, the FEP algorithm performs the calculation of connection time before
connecting to several nearby IoT devices. Same as the RR algorithm, the FEP algorithm connects using sequential way.

The different things happened in the FEP algorithm can be seen in the second cycle and so forth. At the beginning of these cycles, the same as the EP algorithm does, the usage of the database as a queue for the information of BLE devices is also performed here. Henceforth, the scan time in these cycles is also reduced into a relatively shorter period. However, there is also a slight difference between the EP algorithm and FEP algorithm in term of the usage of the queue from the database. In the FEP algorithm, there is a mechanism that differentiates the slaves based on their state. In here, only the active devices will be polled to do the connection and data exchange. Furthermore, as mentioned in [52], after a certain amount of time, all of the device states listed in the database will be changed back to active to be polled in the next polling cycle.

The advantage of using this algorithm is the possibility of asymptotic behavior of the algorithm to work as both RR and EP depends on a load of works resulted from the connections to nearby IoT devices. At the low load, FEP works as an exhaustive schedule. Meanwhile, FEP behaves as RR scheduler at high loads [52]. However, although in FEP algorithm the slave devices are divided into some active and inactive states, the ordering of the available BLE devices in communication radio range has not been prioritized by their cost efficiency towards the connection to the smartphone. This might be useful for a further energy aware mechanism that considers a low power consumption. Hence, another mechanism is also added to this proposed design to tackle the priority issue of the nearby IoT devices based on their cost efficiency.

4.5.10 Multi Criteria Decision Making

When connecting to the nearby IoT devices, the calculation of cost-efficiency against multiple criteria consideration from the device’s parameters is done by using a Multi-Criteria Decision Making (MCDM). MCDM is a way of merging decision alternatives performance for some conflicting qualitative and/or quantitative criteria and results in a compromise solution [53]. In this work, MCDM is used to decide the priority of each nearby IoT devices based on a specified algorithm and rank these devices to give priority in an arrangement procedure. This decision making calculation is essential following the energy consumption and throughput parameters so that the proposed mobile gateway may decide which device has the highest and the least priority to be connected. To use the specified algorithm for MCDM, there are several criteria which are underlying the input of the algorithm, including:

- **RSSI** this criterion determines the Received Signal Strength Indication which presents the power strength in a received signal by the smartphone as a master device. This criterion can also be used to define the distance of a slave device from the master.

- **Smartphone’s Residual Energy** defines the value of power consumed by smartphone as a master device to connect to a slave device using BLE wireless connection.
• **Device State** this criterion defines the state of a certain slave device whether the slave can be connected or not and whether the slave has data to send or not.

These criteria are stored in the node repository component during the scheduling algorithm run for the BLE connection mode. Henceforth, many kinds of MCDM can be used to perform a decision making for prioritizing the available nearby IoT devices. The examples of these algorithms are Analytical Hierarchy Process, Analytical Network Process, Weighted Sum Model, Weighted Product Model, Best Worst Method, Nonstructural Fuzzy Decision Support System, and many more. Accordingly, in this work, two MCDM algorithms will be used for prioritizing the nearby IoT devices namely Analytical Hierarchy Process and Weighted Sum Model. These algorithms will be explained in the following subsections.

**Analytical Hierarchy Process**

Firstly, based on the work of [6], the MCDM algorithm of Analytical Hierarchy Process or abbreviated as AHP is used for ordering available BLE devices in communication radio range following their cost efficiency towards a mule considering multiple metrics. Aimed to combine different measures into a single overall point for ranking decision option, the AHP works by defining the optimum alternatives and to categories the others criteria which portrays them. Several steps that are involved in the AHP weighting scale [54, 53], including:

- Build a hierarchical structure of problem that needs a decision with a structure as follow: overall objective, criteria, subcriteria, and decision alternatives.
- Specify the relative priorities of criteria and subcriteria which state their significance in consideration to the connection towards the element at the higher level on a pairwise basis.
- Provide attribute conformity ratings that define the decision alternatives concerning the subcriteria on a pairwise basis.
- Do the calculation of overall valuations from the decision alternatives, weighting the conformity ratings with the relative priorities of criteria and subcriteria.

The output from the AHP weighting scale used in the proposed system model is the ranking from the nearby IoT devices that becomes an underlying of the prioritization for connection using BLE wireless connection technology. Figure 4.12 shows the building of the hierarchical structure of the AHP in the proposed design system.

As shown in Figure 4.12, the objective is set to a device to connect. The criteria are, as previously mentioned at the beginning of this subsection, RSSI, Device State, and Power Consumption. The subcriteria for RSSI are its value of ≤ -80 dBm and > -80 dBm. The subcriteria for device state is active and inactive. And, the subcriteria for power consumption are some predetermined values which become a constraint for the smartphone’s residual energy. These values are considered as follow:

- Constraint1 is > 1000 mW
4 Mobile Gateway Design

Figure 4.12: Hierarchical structure for the AHP

- Constraint 2 is 100 mW until < 1000 mW
- Constraint 3 is < 100 mW

Furthermore, there are four alternatives in this example of the hierarchical structure proposed design. Thus, in this example, the prioritization of four different BLE devices will be explained.

After building the hierarchical structure, the consideration of significance is specified in these criteria. In here, because of power usage is more important than device state and device state is more critical than RSSI. Henceforth, the number of weight on each criterion will be given with the scale from 1 to 9.

Table 4.1: Scaling for the AHP weight [54]

<table>
<thead>
<tr>
<th>Verbal Judgement</th>
<th>Degree of Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equally Preferred</td>
<td>1</td>
</tr>
<tr>
<td>Moderately Preferred</td>
<td>3</td>
</tr>
<tr>
<td>Strongly Preferred</td>
<td>5</td>
</tr>
<tr>
<td>Very Strongly Preferred</td>
<td>7</td>
</tr>
<tr>
<td>Extremely Preferred</td>
<td>9</td>
</tr>
</tbody>
</table>

From the scale in Table 4.1, the term of an intermediate value between these given values can also be used for providing additional levels of judgment. This scale is important to compare all elements pair wise concerning the objective. Thus, the scale value for each of these criteria, such as 1 is for RSSI, 3 is for Device State, and 5 is for Power Consumption, are given. Therefore, the comparison of them follows the following scale in matrix value:
### 4.5 Energy-Aware Mobile Gateway Design

\[
\begin{bmatrix}
\text{RSSI} & \text{DeviceState} & \text{ResEnergy} \\
1 & \frac{1}{3} & \frac{1}{7} \\
3 & 1 & \frac{1}{7} \\
5 & 2 & 1
\end{bmatrix}
\]  
\tag{4.2}

From the matrix comparison of scale for the criteria, the normalize matrix can be obtained using the computation of the eigenvector of matrix principle \[53, 54\]. The usage of the eigenvector matrix is because the AHP is using pairwise comparison data which is in general translated into a matrix to solve the following Equation:

\[a * aW = k * aW\]  
\tag{4.3}

From Equation 4.3, a is the pairwise comparisons matrix, aW is the vector of the absolute values and k is the eigenvalues of the matrix a \[54\]. As a result, the weight values resulted from eigenvector matrix in Equation 4.2 can be obtained as shown in Table 4.2.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSI</td>
<td>0.11</td>
</tr>
<tr>
<td>Device State</td>
<td>0.31</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 4.2: Normalized matrix for all 3 criteria

From the normalized matrix result, the weighting value of each criterion can be obtained, including 0.11 for RSSI, 0.31 for Device State and 0.58 for Power Consumption. Afterward, the consideration of significance is also given to all of the subcriteria based on each criterion. As in RSSI criteria, the subcriteria RSSI \(\leq\) -80 dBm is given scale \(\frac{1}{2}\) and RSSI \(>\) -80 dBm is given scale 1. Meanwhile, in Device State criteria, active is given scale 3 and inactive is given scale 1. Lastly, in power consumption criteria, constraint 1 is given scale 1, constraint 2 is given scale 3, and constraint 3 is given scale 5. Thus, the scale matrix for each subcriteria follows the following scale in matrix value:

\[
\begin{pmatrix}
\text{RSSI} > -80dBm & \text{RSSI} \leq -80dBm \\
\text{RSSI} > -80dBm & 1 \\
\text{RSSI} \leq -80dBm & \frac{1}{2}
\end{pmatrix}
\]  
\tag{4.4}

\[
\begin{pmatrix}
\text{Active} & \text{Inactive} \\
\text{Active} & 1 \\
\text{Inactive} & \frac{1}{3}
\end{pmatrix}
\]  
\tag{4.5}
For the matrix comparison of scale for each subcriteria, we can again obtain the normalize matrix using the eigenvector of matrix principle for each subcriteria. The results of the eigenvector on each subcriteria will be calculated the average value. This average value will then be multiplied by the criteria weight value and divided by the total of the average value resulted in a sub weight value on the given subcriteria. As the result, Table 4.3, 4.4 and 4.5 show the weighting for each subcriteria as the result from the calculation mentioned before.

Table 4.3: Normalized matrix for RSSI subcriteria

<table>
<thead>
<tr>
<th></th>
<th>RSSI &gt; -80 dBm</th>
<th>RSSI ≤ -80 dBm</th>
<th>Average</th>
<th>SubWeight</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSI &gt; -80 dBm</td>
<td>0,6667</td>
<td>0,6667</td>
<td>0,6667</td>
<td>0,0731</td>
</tr>
<tr>
<td>RSSI ≤ -80 dBm</td>
<td>0,33333</td>
<td>0,3333</td>
<td>0,3333</td>
<td>0,0365</td>
</tr>
</tbody>
</table>

Table 4.4: Normalized matrix for device state subcriteria

<table>
<thead>
<tr>
<th></th>
<th>Active</th>
<th>Inactive</th>
<th>Average</th>
<th>SubWeight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>0,75</td>
<td>0,75</td>
<td>0,75</td>
<td>0,2319</td>
</tr>
<tr>
<td>Inactive</td>
<td>0,25</td>
<td>0,25</td>
<td>0,25</td>
<td>0,0773</td>
</tr>
</tbody>
</table>

Table 4.5: Normalized matrix for the smartphone’s residual energy subcriteria

<table>
<thead>
<tr>
<th></th>
<th>Constraint1</th>
<th>Constraint2</th>
<th>Constraint3</th>
<th>Average</th>
<th>Subweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint1</td>
<td>0,11111</td>
<td>0,1</td>
<td>0,11764</td>
<td>0,11</td>
<td>0,0637</td>
</tr>
<tr>
<td>Constraint2</td>
<td>0,33333</td>
<td>0,3</td>
<td>0,29411</td>
<td>0,31</td>
<td>0,1797</td>
</tr>
<tr>
<td>Constraint3</td>
<td>0,55556</td>
<td>0,6</td>
<td>0,58824</td>
<td>0,58</td>
<td>0,3379</td>
</tr>
</tbody>
</table>

After getting weighting values on each subcriteria and also on each criterion, the calculation for total AHP weight of each device alternative can also be obtained by using the Equation 4.7.

\[
AHP = \sum_{j=1}^{n} \frac{a_{ij}}{m} \times w_{ij} = \sum_{j=1}^{n} \sum_{i=1}^{m} Subweight_{ij}
\]  

The Equation 4.7 shows the AHP weight is the summation on each SubWeight of each Subcriterias from each device alternative. For instance, Device1 has RSSI value of -90 dBm, device state is inactive and power used for the connection is 300 mW. Meanwhile, Device2
has RSSI value of -50 dBm, device state is active and smartphone’s residual energy is 90 mW. Thus, the AHP weight of Device1 and Device2 by using Equation 4.7 are:

\[ AHP_{Device1} = (0.0365 + 0.0773 + 0.1797) = 0.2935 = 29.3\% \]

\[ AHP_{Device2} = (0.0731 + 0.2319 + 0.3379) = 0.6429 = 64.29\% \]

by comparing both device AHP weight values, the priority to connect to Device2 is higher than Device1. Thus, the smartphone will connect to Device2 first before connect to Device1.

The MCDM algorithm using AHP has an advantage of having a good accuracy for the decision making because of a more detail weighting to the subcriteria domain, and the usage of pairwise comparison judgments started from the top side criteria section of the hierarchy down to the subcriteria. However, in term of the energy consumption consumed by this MCDM algorithm, it is not an optimal one because of the matrix calculation that may increase the number of calculations done by the system. Hence, it is essential to utilize another MCDM approach which does not use any matrix calculation to measure the device prioritization decision making from multiple criteria.
Weighted Sum Model

Another algorithm that is used in the proposed mobile gateway for decision making from the multiple criteria is Weighted Sum Model or abbreviated as WSM. This algorithm has been chosen because it is the simplest available method in term of calculation for criteria decision making and is usable to a single-dimensional problem because it adheres intuitive process [53]. Compared to the AHP in which the algorithm uses matrix calculation for calculating the decision making weighting values, the WSM uses just simple addition and multiplication to obtain the weighting values needed for decision making. Equation 4.8 shows the calculation of the weighted sum score of WSM in case of n criteria and m alternatives:

$$WSM = \max_{i} \sum_{j=1}^{m} a_{ij} \times w_{j}$$ (4.8)

where \(i = 1, \ldots, m\) and \(a_{ij}\) is the score of the i-th alternative with respect to j-th criterion and \(w_{j}\) is the weight of j-th criterion.

For example, from the previously mentioned criteria —which is utilized in this proposed mobile gateway about RSSI, Device State, and Power Consumption— a calculation of weight score in the WSM value will also then be calculated. Same as the AHP weighting scale, the criteria in the WSM must also be given any weighing scale which is ranging from 1 to 9. In this case, RSSI is given scale 1, Device State is given scale 3, and Power Consumption is given scale 5.

<table>
<thead>
<tr>
<th>Subcriteria</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSI &gt; -50 dBm</td>
<td>5</td>
</tr>
<tr>
<td>RSSI &gt; -60 dBm</td>
<td>4</td>
</tr>
<tr>
<td>RSSI &gt; -70 dBm</td>
<td>3</td>
</tr>
<tr>
<td>RSSI &gt; -80 dBm</td>
<td>2</td>
</tr>
<tr>
<td>RSSI &lt; -80 dBm</td>
<td>1</td>
</tr>
</tbody>
</table>

In the AHP, the decision-making process covers a good depth accuracy due to the calculation utilizes the weighting score in details until the subcriteria section of the hierarchy. Thus, in the WSM, to make the same depth accuracy, the subcriteria in here must also broaden into a more depth subcriteria so that the weight scale calculation can also cover a more detail value. For instance, in device RSSI criteria, the subcriteria are RSSI > -50 dBm, RSSI > -60 dBm, RSSI > -70 dBm, RSSI > -80 dBm, and RSSI < -80 dBm. Furthermore, the given weight scale for this subcriteria can be seen in Table 4.6. Meanwhile, for subcriteria from Device State criteria, the weighing scale 5 is given to active and 2 to inactive. Furthermore, for smartphone’s residual energy criteria, there are several subcriteria which determine the connection between the current battery percentage and the alternatives value. Table 4.7 shows these alternative values.
### 4.5 Energy-Aware Mobile Gateway Design

#### Table 4.7: Weighting scale for subcriteria in smartphone’s residual energy criteria

<table>
<thead>
<tr>
<th>Subcriteria</th>
<th>Battery Percentage</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power &lt; 100 mW</td>
<td>60-100%</td>
<td>5</td>
</tr>
<tr>
<td>Power 100 - 1000 mW</td>
<td>60-100%</td>
<td>3</td>
</tr>
<tr>
<td>Power &gt; 1000 mW</td>
<td>60-100%</td>
<td>1</td>
</tr>
<tr>
<td>Power &lt; 0 mW</td>
<td>20-60%</td>
<td>4</td>
</tr>
<tr>
<td>Power 0 - 100 mW</td>
<td>20-60%</td>
<td>2</td>
</tr>
<tr>
<td>Power &gt; 100 mW</td>
<td>20-60%</td>
<td>1</td>
</tr>
<tr>
<td>Power &lt; 100 µW</td>
<td>0-20%</td>
<td>3</td>
</tr>
<tr>
<td>Power 100 - 1000 µW</td>
<td>0-20%</td>
<td>2</td>
</tr>
<tr>
<td>Power &gt; 1000 µW</td>
<td>0-20%</td>
<td>1</td>
</tr>
</tbody>
</table>

The Equation 4.8 shows the WSM weight is the total summation on each subcriteria weight from each device alternative multiplied by criteria weight. For instance, assume that all devices are in 100% battery state, Device1 has the RSSI value of -90 dBm, the device state is inactive, and the power used by smartphone for the connection is 300 mW. Meanwhile, Device2 has the RSSI value of -50 dBm, the device state is active, and the power used by smartphone for the connection is 90 mW. Thus, the AHP weight of Device1 and Device2 by using Equation 4.8 are:

\[
WSM_{Device1} = ((1\times1) + (3\times2) + (5\times3)) = 22
\]

\[
WSM_{Device2} = ((1\times5) + (3\times5) + (5\times5)) = 45
\]

By comparing both device WSM weight values, the priority to connect to Device2 is higher than Device1. Thus, the smartphone will connect to Device2 first before connecting to Device1.

As aforementioned in the previous subsection that the MCDM using WSM has an advantage of having a low energy consumption because it utilizes only some small calculations compared to the AHP which implements the computation of matrix and eigenvector values while the WSM also can still maintaining the depth accuracy by adding some detail values in the subcriteria section.
Chapter 5

Implementation

In this Chapter, the implementation of the proposed mobile gateway will be explained in detail. Started by the description of the preliminaries, including the BLE implementation in the Android system, proposed system mobile gateway’s class diagram, user interface, and system settings. Subsequently, the discussion will be shifted into the designed system implementation. This section is divided into four subsections, such as the implementation of the gateway controller, gateway service, scheduling and the merge of scheduling and MCDM mechanism.

5.1 Preliminaries

In this Section, the basis of the implementation of this work will mainly be discussed. Four Subsections are underlying the basis of the implementation of this work, including BLE in Android, the system user interface, the system class diagram, and the system settings.

5.1.1 BLE in Android

The implementation of BLE in the Android has been provided since the version 4.3 which is known as KitKat. The API level 18, which is brought at Android version 4.3 and above, supports the ability to the third party application for using BLE in Android device for discovering devices, querying for services, and transmitting a small amount of data [55].

As aforementioned in Chapter 2 that the Android implements the principle of least privilege so that to activate the BLE in the Android. First, the permission of BLUETOOTH must be given in the Android manifest XML file. Furthermore, for enabling the App to make use of Bluetooth settings or commence device discovery, then the permission of BLUETOOTH_ADMIN must also be included in the manifest. Thus, these two permissions are written in the XML manifest file as follows:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```
Another important thing which must be written in the manifest file is the declaration that the App will use the only BLE-capable device only. This can be written in the XML manifest file as follows:

```xml
<uses-feature android:name="android.hardware.bluetooth_le"
    android:required="true"/>
```

After implementing the settings of BLE in the Android manifest file, next the set up of BLE in the Android will be explained in details. According to [55], there are two steps for the BLE set up, including:

- Obtain the `BluetoothAdapter` which is needed for all Bluetooth activity. As a representation of the smartphone’s Bluetooth adapter (Bluetooth radio), this one object interacts with the third party App which is used in the proposed mobile gateway as a mobile gateway for IoT devices. The code below shows how the `BluetoothAdapter` can be obtained.

```java
private BluetoothAdapter mBluetoothAdapter;
...
BluetoothManager bluetoothManager =
    (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
```

- Enabling the Bluetooth in the Android system settings. This is essential to avoid the Application Not Responding (ANR) error message while running the App. A method called `isEnabled()` is used to check whether the Bluetooth is currently enabled or not. Henceforth, the code below shows how to the state as mentioned above.

```java
if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent =
        new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

After setting up the BLE, the discovery of nearby IoT devices is performed on the Android. Firstly, the `startLeScan()` method is executed for turning the BLE device discovery on. Then, the turn off for the BLE device discovery uses `stopLeScan()`. According to [29, 55], the scan is a battery-intensive in term of energy consumption. Thus, it is recommended for not performing a scan in a loop and manage a restrict to the time of the scan. The code below depicts a method to start and stop the scan [29, 55].

```java
private void scanLeDevice(final boolean enable) {
    if (enable) {
```
// Stops scanning after a pre-defined scan period.
mHandler.postDelayed(new Runnable() {
    @Override
    public void run() {
        mScanning = false;
        mBluetoothAdapter.stopLeScan(mLeScanCallback);
    }
}, SCAN_PERIOD);

mScanning = true;
mBluetoothAdapter.startLeScan(mLeScanCallback);
} else {
    mScanning = false;
    mBluetoothAdapter.stopLeScan(mLeScanCallback);
}
...
...

As shown in the above code, the SCAN_PERIOD is a parameter of time given for discovering BLE devices. After that, the result of the BLE scan is a callback received by the smartphone which will be obtained from the interface `BluetoothAdapter.LeScanCallback`. The code below shows an interface for receiving the callback from the BLE scan.

```java
private LeDeviceListAdapter mLeDeviceListAdapter;
...
private BluetoothAdapter.LeScanCallback mLeScanCallback =
    new BluetoothAdapter.LeScanCallback() {
        @Override
        public void onLeScan(final BluetoothDevice device, int rssi,
            byte[] scanRecord) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // Code for receiving callback is put here
                }
            });
        }
    };
```

From the code written above, the scanRecord parameter which contains a byte array parameter is the representation of the received data from the advertise mode from the slave devices. The data received here usually contains device manufacturer ID, the available services from
the device and many more. Thus, this data is stored in the database as a Binary Linkage Object (BLOB) which can be used for further processing.

After receiving the callback, then the BluetoothDevice object can be obtained. Furthermore, for interacting with a specific BLE device, the connection mode can be performed by method connectGatt() from the BluetoothDevice object obtained from the scan callback. This method links the GATT server hosted by the specific BLE device and returns a BluetoothGatt instance. The code below illustrates the connection to a specific BLE device GATT server [29][55].

```java
private BluetoothGatt mBluetoothGatt;
...
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

The BluetoothGatt instance can be used to perform GATT client operations, and the result from these operations are delivered back to as a BluetoothGattCallback interface. The code below shows an interface for receiving the callback from the BluetoothGatt operations [55].

```java
private final BluetoothGattCallback mGattCallback =
new BluetoothGattCallback() {

@Override
public void onConnectionStateChange(BluetoothGatt gatt, int status,
int newState) {
    String intentAction;
    if (newState == BluetoothProfile.STATE_CONNECTED) {
        // GATT connection is successful
    } (newState == BluetoothProfile.STATE_DISCONNECTED) {
        // GATT connection is unsuccessful
    }
}

@Override
public void onServicesDiscovered(BluetoothGatt gatt, int status) {
    if (status == BluetoothGatt.GATT_SUCCESS) {
        // New services discovered
    } else {
        // New services is not discovered
    }
}

@Override
public void onCharacteristicRead(BluetoothGatt gatt,
    BluetoothGattCharacteristic characteristic, int status) {
```
As mentioned in the above code, there are some callback methods which are automatically generated from the BluetoothGattCallback interface, including onConnectionStateChange(), onServicesDiscovered(), onCharacteristicRead(), onCharacteristicWrite(), onCharacteristicChanged() and so forth. The five methods mentioned above are the important methods which underlying the callback from the GATT client to the GATT server.

Firstly, the onConnectionStateChange() method will be triggered when the state of the Bluetooth the GATT client is changed. Then, the onServicesDiscovered() method is triggered when a service/services are found in the GATT client. Afterward, the trigger of onCharacteristicRead() happens when a characteristic/characteristics are requested to be read by the GATT server. Then, the onCharacteristicWrite() will be triggered when a characteristic/characteristics are requested to be written to the GATT client by the GATT server. Lastly, if there is a characteristic value change in the GATT client by the activation of notify or indicate from the GATT server, the trigger of onCharacteristicChanged() happen.

Subsequently, to receive the automatic GATT notification from the GATT client for a specific characteristic, the method of setCharacteristicNotification() is used. Once a notification for the specific characteristic is activated, the method of onCharacteristicChanged() will be triggered. Using the code below, the activation of a specific characteristic notification can be performed by the Apps [55].

```java
private BluetoothGatt mBluetoothGatt;
BluetoothGattCharacteristic characteristic;
...
    mBluetoothGatt.setCharacteristicNotification(characteristic, true);
    ...
```
5 Implementation

```
BluetoothGattDescriptor descriptor = characteristic.getDescriptor(
    UUID.fromString(SampleGattAttributes.CLIENT_CHARACTERISTIC_CONFIG));
descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
mBluetoothGatt.writeDescriptor(descriptor);
```

5.1.2 Class Diagram

For determining the structure of the proposed mobile gateway by displaying the system’s classes, their methods, their attributes and the relationships between the objects, a class diagram is used in the software engineering area as a section of Unified Modelling Language (UML).

As depicted in Figure 5.1, there are several packages which pack and organize the classes and interfaces alongside the Java code for the Android. All of these classes and interfaces are mainly started or entered first by the user from the `MainActivity.class`. The packages included in the proposed mobile gateway consist of Bluetooth, database, gateway, service, thread, and helper.

The Bluetooth package organizes all classes which are used to control the Bluetooth and BLE operations. Furthermore, the database package packs all classes that are implemented for the database related operations. Henceforth, the gateway package compiles all classes and interfaces regarding the primary operations of the proposed mobile gateway system. Then, the service package manages the classes concerning the service interface mechanism in the system. Finally, the last two packages namely thread and helper organize all classes used by the system that help to achieve results, including the UI related helper, some data converters, network related utilities and the controller for the usage of the multi-threading system in the Android.
Figure 5.1: Class diagram from the proposed mobile gateway
5 Implementation

5.1.3 User Interface

As a part where the interaction between users and machine happens, a user interface (UI) is also an integral part of the proposed mobile gateway. A simple yet useful and follows human-centered design is needed for an App to make it more exciting and enjoyable to be used. However, because of the aimed of the proposed system is to build an energy-aware system that can run for an extended period without any user’s interaction. Thus, the designated system UI should be made lean and clean from the unnecessary graphics or component.

Three main screens are underlying the proposed design system UI. These screens are divided each other based on the Android tabs with Fragments. As depicted in Figure 5.1, three main fragment classes connect to the Android display layout XML file. These fragments are the modular section of the main activity class that holds the whole parts of the UI from the proposed mobile gateway. The starting point for the user to enter the App is in the MainActivity.class Which complies with the entry point for the user to enter the App as described in [27]. These fragment classes are:

- **GatewayFragment.class** is a portion of the MainActivity.class That is opened at the beginning when the user opens the App for the first time. This fragment shows the current status of the mobile gateway which is written in a command line based non-changeable and non-clickable UI as shown in Figure 5.2a.

- **ScannerFragment.class** is a piece of the MainActivity.class Which is opened when the user clicks on the scanner tab as shown in Figure 5.2b. This fragment exhibits the scanning screen for the available BLE devices so that the user can see which devices are available in the nearby Bluetooth communication range.

![Gateway fragment](image1)
![Scanner fragment](image2)
![Service interface](image3)

Figure 5.2: Screenshot of the system UI
5.1 Preliminaries

- **ServiceInterface.class** is a part of the **MainActivity.class** To which the user will be pointed after a nearby IoT device has connected with the smartphone, and the data has been gathered from it and stored in the smartphone’s database. These connected devices are then listed in the list as shown in Figure 5.2c that the device’s name and the device’s MAC Address become an identifier of the device information. If the user clicks this list, then another Fragment will be opened which contains the screen of the device’s services and characteristics information or the screen for an uploader of the gathered data to the cloud.

5.1.4 System Settings

As aforementioned in Chapter 4 Subsection System Settings, there are four settings which will be used as the system parameter divided into two XML files namely **Settings.xml** and **Manufacturer.xml**. Firstly, in the **Settings.xml**, the first setting is the default scheduling algorithm. The implementation of this setting can be seen in the code below:

```xml
<DataAlgorithm>
  <data>
    <currentAlgorithm>fep</currentAlgorithm>
  </data>
</DataAlgorithm>
```

As written in the above code, the the element `<currentAlgorithm>` describes the default algorithm used by the proposed system is Fair Exhaustive Polling (FEP). Furthermore, the second setting written in the **Settings.xml** file is the power measurement constraint. The code below shows the implementation from this setting:

```xml
<DataPowerConstraint>
  <data>
    <id>case1</id>
    <batteryLevel>60</batteryLevel>
    <batteryLevelUpper>100</batteryLevelUpper>
    <threshold1>10^{-14}</threshold1>
    <threshold2>10^{-15}</threshold2>
    <threshold3>10^{-15}</threshold3>
  </data>
  <data>
    <id>case2</id>
    <batteryLevel>20</batteryLevel>
    <batteryLevelUpper>60</batteryLevelUpper>
    <threshold1>10^{-13}</threshold1>
    <threshold2>10^{-14}</threshold2>
    <threshold3>10^{-14}</threshold3>
  </data>
</DataPowerConstraint>
```
As written in the code above, the power consumption constraint consists of three kinds as previously mentioned in Table 4.7 for weighting scale of power consumption constraint. The element <batteryLevel> describes the lower level of the battery constraint while the element <batteryLevelUpper> determines the upper level. Furthermore, there are three threshold states which determine the each kind of power consumption constraint.

Afterward, the time based constant is the third setting written in the Settings.xml file. The snippet of the code for this setting is shown below:

As written in the code above, there are six different parameters that become the system settings, including processing time \(^1\), scanning time \(^2\), and scanning time \(^3\), unit, mape time start and mape time. As mentioned in Chapter 4 Section System Settings, these settings become the basis of all timers used in the proposed mobile gateway.

---

1 A timer setting for one cycle of a program
2 A timer setting for discovering nearby IoT devices
3 A shorter timer setting for discovering nearby IoT devices
Lastly, the last setting is written in Settings.xml file is the self-adaptation system based on MAPE setting. The code below depicts the snippet of the code:

```
<DataMape>
    <data>
        <mapeAction>yes</mapeAction>
        <mapeDataUpload>yes</mapeDataUpload>
    </data>
</DataMape>
```

These settings are used to control the state of the MAPE algorithm and upload data actions which in the above code, these two parameters are set to turn on. Nevertheless, the state of the data upload depends on the state of the MAPE algorithm. Hence, if the user wants to run the data upload, it requires also the MAPE algorithm to run. In contrast, the MAPE algorithm does not depend on the data upload.

Another implementation of the setting is the Manufacturer.xml file. For determining the data of the manufacturer from the nearby IoT devices. The code below shows the snippet of the code for this setting:

```
<manufacturers>
    <manufacturer>
        <id>0x0157</id>
        <name>Anhui Huami Information Technology</name>
        <service>0000fee0-0000-1000-8000-00805f9b34fb</service>
    </manufacturer>
    ... // other manufacturer settings
</manufacturers>
```

As written in the code above, the manufacturer data is written in a list of settings which can be used for many manufacturers all at once. Henceforth, there are three parameters included into the setting. The first element <id> defines the manufacturer id parameter from a BLE device. The second element <name> defines the manufacturer name parameter and the last element <service> determines a particular available service in the BLE device.

5.2 The Designed System

After explaining the preliminaries, in this Section, we introduce the implementation of the proposed mobile gateway. The main parts that are discussed include the gateway service, the gateway controller, scheduling algorithms and MCDM mechanisms.
5 Implementation

5.2.1 Gateway Controller

Algorithm 1 Gateway Controller

Require: $A_L \leftarrow A_{Settings}$ or $A_L \leftarrow A_{MAPE}$
Ensure: $A_L \neq \text{Null}$

// Code for Scheduling_Thread
while $A_L \neq \text{Null}$ do
    if $A_L$ is "EP" then
        doScheduleEP()
    else \{$A_L$ is 'FEP'}
        doScheduleFEP()
    ....
    .... // Do other scheduling algorithms
    ....
end if
end while

// Code for MAPE_Thread
while true do
    if time schedule has been reached then
        getParameters() // Obtain some Mape input parameters
        $X \leftarrow \text{doMape}()$
        interruptThread1()
        $A_L \leftarrow X$ // Put new algorithm from temporary memory
        startThread1()
    end if
end while

startSchedulingThread()
startMAPEThread()

A gateway controller for the proposed mobile gateway can be considered as one of an integral part due to the controlling of the central mechanism for the whole mobile gateway system happens in this component. The primary mechanism that is performed in this component is the selection of the scheduling algorithm based on the output from the self-adaptive system using MAPE. Another consideration that makes this component as an integral part is that the start & stop of the scheduling mechanism is also performed in here. To perform such mechanisms, there is a critical point for the usage of multi-thread processing between the self-adaptive system and the scheduling mechanism. For the clear illustration of the previously mentioned scenario, the algorithm for the gateway controller is depicted using pseudocode at the Algorithm 1.

As shown in Algorithm 1 a multi-threading programming is used to perform both MAPE
algorithm and the scheduling algorithm concurrently. This algorithm is executed under the Java class of GatewayController.class. To do the previously mentioned mechanism, the scheduling is wrapped into a runnable and put into Thread1. Meanwhile, the MAPE algorithm is also wrapped into a runnable and performed in another separate Thread using ScheduledThreadPoolExecutor. While the scheduling algorithm is running in Scheduling_Thread, the MAPE algorithm is waiting for a specific schedule to run in MAPE_Thread. When the time has been reached, the doMape() method is executed using a fuzzy set algorithm which produces a new algorithm that will be selected and executed after interrupting the previous scheduling algorithm thread. Thus, the keyword synchronized is used in this case to make sure that the atomic region is created and the thread safe mechanism, as mentioned in Chapter 4, is performed in this critical section.

5.2.2 Gateway Service

As mentioned in Chapter 4 Section Gateway Service, the primary usage of this component is for providing a programming interface in the proposed system. This interface is used to give an abstraction for some other components to access the peripheral system on the Android, including database access, power estimator, wireless connections, etc. Furthermore, this abstraction should also run the system in the background process for an extended period to which the user does not notice that the system is running it. For fulfilling these requirements, the background service using AIDL is used in this particular component.

The implementation of AIDL requires the system to have an interface with a .aidl file written on the same package as the service class is made. In this designed system, the .aidl file namely IGatewayService.aidl is made alongside the GatewayService.class in the same package. The example of the interface written in the .aidl file can be seen in the code below.

```java
import BluetoothDevice;
import List;
import ParcelUuid;
...

interface IGatewayService {
    int getPid();
    void setProcessing(boolean mProcessing);
    ....
    ....
}
```

As shown in the above code, firstly, import of all required objects, which are not included in the some Java primitive data types, are performed. After that, the interface codes begin. After giving the interface codes, then the implementation of these interface codes are automatically generated in the GatewayService.class. This class can be depicted using pseudocode at the Algorithm 2.
5 Implementation

Algorithm 2 Gateway Service

Require: \texttt{iBinder} ⇒ \texttt{Null}
Ensure: xmlSettings \neq \texttt{Null}

doInitialize()
\texttt{iBinder} \leftarrow \texttt{new iGatewayService.Stub()}
\textbf{while} true \textbf{do}
\hspace{1em} \textbf{if} \texttt{iBinder} \neq \texttt{Null} \textbf{then}
\hspace{2em} public int getPid()
\hspace{2em} public void setProcessing(boolean mProcessing)
\hspace{2em} ....
\hspace{2em} ....
\hspace{1em} \textbf{end if}
\textbf{end while}

From the algorithm 2, the gateway service at first does the initialization of some variables. Then, \texttt{iGatewayService.Stub()} is assigned to the binder framework \texttt{mBinder}. The \texttt{iGatewayService.Stub()} is resulted from the AIDL server process which indicates the server process is ready for the operation of marshaling and unmarshalling data, and the Java code has been created from the list of interfaces written in the .aidl file. Then, the methods inside the \texttt{mBinder} are ready to be used and are wrapped into the Java code that can be executed by the client process as mentioned in the AIDL section earlier in this chapter. Therefore, if a client binds to it, the client can then execute some methods or services that are available in the gateway service.

To bind to the gateway service, the client must first call the method \texttt{bindService()} and give the parameter of this method an intent of the \texttt{GatewayService.class}. Furthermore, a define for callbacks for service binding, passed to the method \texttt{bindService()} must also be given so that the binding status can be determined from the callbacks given on \texttt{onServiceConnected()} and \texttt{onServiceDisconnected()} methods.

After the service is bounded, the execution can be done in the method which is written inside the Gateway Service component. The example of the RPC using AIDL is, the scheduling \& MCDM component has started to do the scanning at the beginning of the cycle, and it stores data to the database in the node repository component. Then the scheduling \& MCDM component should perform the binding as mentioned above and looks for the methods available in the gateway service for storing data into the BLE Device Data in the database section, i.e., the method of \texttt{iGatewayService.insertDeviceData(int param1, String param2, ...)} is called. Then, the method should be called from the client process — in this case, the scheduling \& MCDM component is the client process, and the Gateway Service component is the server process — with the predetermined parameters which are also sent to the method execution in the server process. Then, the process of AIDL remote procedure call as depicted in Figure 4.4 will be performed, and the result of the return is sent back to the client process.
5.2 The Designed System

5.2.3 Scheduling

The scheduling part implementation is based on the designed system in which four basic scheduling algorithms are underlying the proposed mobile gateway. These algorithms consist of Semaphore, Round Robin, Exhaustive Polling and Fair Exhaustive Polling. In the following Subsection, the implementation of these algorithms is explained using the pseudocode for the explanation of the algorithm.

Implementation of the SEM Algorithm

As previously mentioned in Chapter 4, synchronous communication is utilized for this algorithm scheduling of connection method. The implementation of such type of connection scheduling can be seen in the pseudocode of Algorithm 3.

Algorithm 3 Semaphore algorithm

Input: Set of IoT nodes \( L_s \)
Output: Set of collected data \( L_{data} \)

Require: \( T_{scan} \leq 10s \)

1: while true do
2: \( set T \) into \( T_{scan} \)
3: \( \text{while } T \) has not been reached do
4: \( \text{switch into scanning state} \)
5: \( \text{detect nearby IoT devices from } L_s \)
6: \( \text{if } IoT \text{ devices satisfies a certain condition then} \)
7: \( \text{add into list } L_d \)
8: \( \text{end if} \)
9: \( \text{end while} \)
10: \( \text{switch into stop scanning state} \)
11: \( \text{call semaphore connection procedure} \)
12: \( \text{return } L_{data} \)
13: \( \text{end while} \)

As depicted from the Algorithm 3 about the SEM scheduling, at first, the setting of the time used for the period of scanning \( T_{scan} \), which is used to discover the nearby IoT devices, is taken from the parameter in the System Settings component. After that, the program switches the scan to turn on and do the scanning while also adding the scan result in the list \( L_d \). After the timer \( T \) has reached a certain amount of time, the system switches the scanning to turn off.

Afterward, the system calls the semaphore connection procedure, as depicted in Algorithm 4. Based on the list of devices \( L_d \), the system iterate for each device. After that, the system comes to the connection to a specific BLE device which acts as a slave device based on
the BluetoothDevice object resulted from the ForEach iteration. In here, the keyword of synchronized is used to make sure that only one thread is accessing the critical section of the connection to a slave device. After the connection to the slave device is established, the cursor moves to the next section of lock() where the system waits for the callback connected or disconnected resulted by the connection via BLE. After the slave device triggers the callback, the exchange of data is then performed, and the data is saved in the list $L_{data}$.

Algorithm 4 Semaphore connection procedure

Ensure: $L_d \neq Null$
1: for each device $d \in L_d$ do
2:     synchronized device $d$
3:     connect device $d$
4:     $d$.lock() // wait until connected / disconnected
5:     if data is available then
6:         $L_{data} \leftarrow$ obtain data from device $d$
7:     end if
8: end for

Implementation of the RR Algorithm

Algorithm 5 Round Robin algorithm

Input: Set of IoT nodes $L_s$
Output: Set of collected data $L_{data}$
Require: $T_{processing} \leftarrow 60s, T_{scan} \leftarrow 10s$
1: while $T_{processing}$ has not been reached do
2:     set $T$ into $T_{scan}$
3:     while $T$ has not reached do
4:         switch into scanning state
5:         detect nearby IoT devices from $L_s$
6:         if IoT devices satisfies a certain condition then
7:             add into list $L_d$
8:         end if
9:     end while
10:     switch into stop scanning state
11:     set $N$ into number of devices
12:     $T_{remaining} \leftarrow T_{processing} - T$
13:     set $T$ into $T_{remaining} / N$
14:     call round robin connection procedure
15: return $L_{data}$
16: end while
The basic of the scheduling algorithm using the RR algorithm is to obtain the fairness of the bandwidth in between several slave devices. Thus, to get such fairness, a timing model based on the cyclic operation is implemented. The implementation of such an algorithm is written in the pseudocode on Algorithm 5.

As written in the Algorithm 5 about the RR scheduling, the time of each cycle is determined by processing time $T_{\text{processing}}$. Furthermore, the time of the connection $T$ in this algorithm is determined by the available number of devices within the Bluetooth scanning range. In this case, after getting the list of devices in $L_d$, the number of nearby IoT devices can be obtained and stored in $N$ temporarily. After that, the time used to connect to a specific BLE device, which acts as a slave device, can be obtained by getting the remaining time $T_{\text{remaining}}$ (the overall processing time $T_{\text{processing}}$ of one cycle subtracted by the scanning time $T_{\text{scan}}$) divided by the $N$.

**Algorithm 6 Round Robin connection procedure**

Ensure: $L_d \neq \text{Null}$
1: for each device $d \in L_d$ do
2: connect device $d$
3: if data is available then
4: $L_{\text{data}} \leftarrow$ obtain data from device $d$
5: end if
6: wait($T$) // wait for timer if reached disconnected
7: end for

Afterward, the system calls the round robin connection procedure, as shown in Algorithm 6. Based on the list of devices $L_d$, the system iterate for each device. After that, the system comes to the connection to a specific slave device based on the BluetoothDevice object resulted from the ForEach iteration. Henceforth, there comes a difference between this scheduling algorithm with the semaphore scheduling algorithm to which in round robin, after the connection to a slave device has been established, the data will be obtained and a certain amount of time $T$, which is obtained from the result of division between remaining time $T_{\text{remaining}}$ and $N$, will be given in order to give a timeout for the connection to the slave device. This ensures each device within the Bluetooth range may have the fairness of connection opportunity which is required to obtain the data.

**Implementation of the EP Algorithm**

An Exhaustive Polling algorithm is, as aforementioned in Chapter 4 Section Exhaustive Polling Scheduling Algorithm, performed in a master device to poll the addressed slaves continuously until the queue, which is a node repository component in the proposed mobile gateway, is emptied. The implementation of such an algorithm is written in the pseudocode on Algorithm 7. The main difference between this algorithm and the semaphore algorithm is on the scanning part. In semaphore algorithm, the scan is performed without considering the data in the database $L_{\text{DB}}$. Thus, the timer $T$ uses a reasonable scan time $T_{\text{scan}}$. In contrast, the EP
algorithm uses the data in the database $L_{DB}$ which is used as a queue and store data of the previously scanned devices on it. It is resulting in the timer $T$, when the database $L_{DB}$ does not empty, uses a shorter scan time $T_{other}$.

Algorithm 7 Exhaustive Polling algorithm

Input: Set of IoT nodes $L_s$
Output: Set of collected data $L_{data}$

Require: $T_{scan} \leftarrow 10s$, $T_{other} \leftarrow 5s$

1: while true do
2:    $L_{DB} \leftarrow$ get BLE device database
3:    if $L_{DB}$ is not Null then
4:        set $T$ into $T_{other}$
5:    end if
6:    $L_d \leftarrow$ get list of device from $L_{DB}$
7: else
8:    set $T$ into $T_{scan}$
9: end if
10: while $T$ has not been reached do
11:    switch into scanning state
12:    detect nearby IoT devices from $L_s$
13:    if IoT device satisfies a certain condition then
14:        add into list $L_d$
15:        if IoT device not listed in $L_{DB}$ then
16:            insert device into database
17:        end if
18:    end if
19: end while
20: switch to stop scanning
21: call procedure semaphore connection
22: return $L_{data}$
23: end while

After determining the timer $T$, the system moves to the scan loop in which the scan is performed for a certain amount of time from the timer $T$ and detecting the nearby IoT devices. If the BLE device has not been written into the database, then it will be inserted as a new data in BLE Device Data database. Afterward, the scan is stopped, and the system moves to call the procedure of connection with semaphore algorithm. This procedure is the same as what written in Algorithm 4 about Semaphore Connection Procedure.
5.2 The Designed System

Implementation of the FEP Algorithm

According to [52], as an intelligent scheduling algorithm, the FEP algorithm is performed by merging both the RR and EP algorithms so that the advantages of both algorithms can be achieved. However, another important thing that is also utilized in this algorithm is the use of a mechanism of two slave device states based on the ability to connect and obtain data. These two states are active state and inactive state. The implementation of such algorithm is shown in the pseudocode on Algorithm 8.

**Algorithm 8** Fair Exhaustive Polling algorithm

**Input:** Set of IoT nodes \( L_s \)

**Output:** Set of collected data \( L_{data} \)

**Require:** \( T_{processing} \leftarrow 60s \), \( T_{scan} \leftarrow 10s \), \( T_{other} \leftarrow 5s \)

1: while ProcessingTime has not been reached do
2: \( L_{DB} \leftarrow \) get BLE device database
3: if \( L_{DB} \) is not Null then
4: set \( T \) into \( T_{other} \)
5: \( L_d \leftarrow \) get list of active device from \( L_{DB} \)
6: else
7: set \( T \) into \( T_{scan} \)
8: end if
9: while \( T \) has not been reached do
10: switch into scanning state
11: detect nearby IoT devices from \( L_s \)
12: if IoT device satisfies a certain condition then
13: set device inactive
14: add into list \( L_d \)
15: if IoT device not listed in \( L_{DB} \) then
16: insert device into database
17: end if
18: end if
19: end if
20: end while
21: switch into stop scanning state
22: set \( N \) into number of devices
23: \( T_{remaining} \leftarrow T_{processing} - T \)
24: set \( T \) into \( \frac{T_{remaining}}{N} \)
25: call procedure round robin connection
26: return \( L_{data} \)
27: end while
5 Implementation

As shown in the Algorithm 8, the discovering of the nearby IoT devices is the same as the EP algorithm does in which its scan process considers the data in the database $L_{DB}$ which is used as a queue and store data of the previously scanned devices on it. The previously mentioned mechanism resulting in the timer $T$, when the database $L_{DB}$ does not empty, uses a shorter scan time $T_{other}$.

Afterward, the system shifts the process to the scan loop in which the scan is performed for a certain amount of time from the timer $T$ and detecting the nearby IoT devices. Same as in the EP algorithm, if the BLE device has not been written into the database, then it will be inserted as a new data in BLE Device Data database.

After that, when the scan state is moved to stop, the calculation of timer for each connection as in the round robin scheduling algorithm is calculated. First, the time of each cycle is determined by processing time $T_{processing}$. Next, the time of the connection $T$ in this algorithm is determined by the available number of devices within the Bluetooth scanning range. In this case, after getting the list of devices in $L_d$, the number of nearby IoT devices can be obtained and stored in $N$ temporarily. After that, the time used to connect to a specific BLE device, which acts as a slave device, can be obtained by getting the remaining time $T_{remaining}$ (the overall processing time $T_{processing}$ of one cycle subtracted by the scanning time $T_{scan}$) divided by the $N$.

After getting the timer $T$, the system calls the procedure of round robin connection, the same procedure that is written in the Algorithm 6. Hence, the round robin part of the FEP algorithm is performed in this section. Furthermore, in during the connection procedure, the active and inactive state of the IoT device is determined by waiting the BLE callback from it. When the callback returns connected and the service discovery is successful, then the IoT device state is set to active. However, if the callback returns disconnected, then the IoT device state is set to inactive.
5.2 The Designed System

5.2.4 Scheduling Combined with MCDM

The mechanism of MCDM, as previously mentioned in Chapter 4 Section Multi-Criteria Decision Making, ensures that the cost-efficient calculation during the connection to a particular slave device against multiple criteria consideration from the device’s parameters is considered. This mechanism is implemented during the scheduling algorithm execution. In the following Subsection, the implementation of the kinds of MCDM combined with the scheduling algorithms is explained using the pseudocode for the detail explanation of their implementations.

Implementation of WSM as a Decision Making

The WSM is one of a kind of the MCDM mechanism which is used in this thesis work for performing a decision making from multiple criteria consideration.

Algorithm 9 Weighted Sum Model

Input: Set of Bluetooth devices $L_d$

Output: Set of sorted Bluetooth devices $L_{sorted}$

1: while true do
2:     for each device $d \in L_d$ do
3:         get device parameters
4:         set criteria scales
5:         set subcriteria scales
6:         $D_{percent}$ set to default
7:         $D_{percent} \leftarrow D_{percent} + (\text{subcriteria scale} \times \text{criteria scale})$ in RSSI
8:         $D_{percent} \leftarrow D_{percent} + (\text{subcriteria scale} \times \text{criteria scale})$ in Device State
9:         $D_{percent} \leftarrow D_{percent} + (\text{subcriteria scale} \times \text{criteria scale})$ in Power Consumption
10:     $L_{unsorted} \leftarrow \text{map device } d \text{ and } D_{percent}$
11: end for
12: $L_{sorted} \leftarrow \text{sort } L_{unsorted} \text{ based on percentage}$
13: return $L_{sorted}$
14: end while

An Algorithm which is written in the Algorithm 9 shows the implementation of the WSM as a kind of MCDM mechanism in pseudocode. The WSM mechanism starts by running the ForEach loop on the list of devices $L_d$ to get device $d$. After that, the system obtains the device parameters based on the device $d$ identity. Next, the criteria scales are set to volatile memories. The same as the criteria that on each criterion on a certain scale is given, subcriteria are also set on each of the subcriterion a certain scale number. Afterward, the calculation for finding the WSM weight percent $D_{percent}$ is performed in the next execution. Starting with the setting of the value $D_{percent}$ to default, each criterion multiplies the criteria
5 Implementation

and the subcriteria scales. The result of this calculation is then added with the previous value of the WSM weight percent $D_{\text{percent}}$ and stored in the temporary memory of the WSM weight percent $D_{\text{percent}}$.

**Implementation of AHP as a Decision Making**

As one of the kinds of the MCDM, the AHP provides a decision making that combines different measures into a single overall point for ranking decision option. An Algorithm which is written in the Algorithm 10 shows the implementation of the AHP as a kind of MCDM in pseudocode.

**Algorithm 10 Analytical Hierarchy Process**

**Input:** Set of Bluetooth devices $L_d$

**Output:** Set of sorted Bluetooth devices $L_{\text{sorted}}$

1: while true do
2:   set criteria scale into matrix
3:   compute normalized matrix
4:   $M_x \Leftarrow$ compute criteria weights
5:   set subcriteria from RSSI scale into matrix
6:   compute normalized matrix with $M_x$
7:   $M_1 \Leftarrow$ compute subcriteria weights
8:   set subcriteria from Device State scale into matrix
9:   compute normalized matrix with $M_x$
10:  $M_2 \Leftarrow$ compute subcriteria weights
11:  set subcriteria from Power Usage scale into matrix
12:  compute normalized matrix with $M_x$
13:  $M_3 \Leftarrow$ compute subcriteria weights
14:  for each device $d \in L_d$ do
15:    get device parameters
16:    $D_{\text{percent}} \Leftarrow$ get $M_1$ based on subcriteria
17:    $D_{\text{percent}} \Leftarrow D_{\text{percent}} +$ get $M_2$ based on subcriteria
18:    $D_{\text{percent}} \Leftarrow D_{\text{percent}} +$ get $M_3$ based on subcriteria
19:    $L_{\text{unsorted}} \Leftarrow$ map device $d$ and $D_{\text{percent}}$
20:  end for
21:  $L_{\text{sorted}} \Leftarrow$ sort $L_{\text{unsorted}}$ based on $D_{\text{percent}}$
22: return $L_{\text{sorted}}$
23: end while

As shown in the Algorithm 10, the AHP mechanism starts by setting the criteria scale in a matrix comparison between each criterion with the others. After that, the normalize process
5.2 The Designed System

using Eigenvector is performed, and the result is stored in a temporary memory \( M_x \). After that, the same operations are also performed on each of the subcriteria, and the results are stored in temporary memory \( M_1, M_2, \) and \( M_3 \). Next, on each device listed on the list \( L_d \), a calculation of the percent of AHP weight \( D_{\text{percent}} \) from the subcriteria scale is performed. Then, the result of the calculation on each device is stored in an unsorted list of hashmap \( L_{\text{unsorted}} \). Lastly, based on the results stored in \( L_{\text{unsorted}} \), a sorting mechanism is done for the list of hashmap between BluetoothDevice objects and \( D_{\text{percent}} \) to sort in ascending the best AHP weight percent \( D_{\text{percent}} \) to the worst. The result is then stored again in a list of sorted devices \( L_{\text{sorted}} \).

Implementation of the EP with MCDM

The implementation of a combination between the EP scheduling algorithm and MCDM mechanism is shown in the Algorithm 11.

**Algorithm 11 Exhaustive Polling algorithm with MCDM**

<table>
<thead>
<tr>
<th>Input: Set of IoT nodes ( L_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Set of collected data ( L_{\text{data}} )</td>
</tr>
<tr>
<td>Require: ( T_{\text{scan}} \leftarrow 10s, T_{\text{other}} \leftarrow 5s )</td>
</tr>
</tbody>
</table>

1: while true do
2: perform EP Scanning Process
3: switch into stop scanning state
4: perform MCDM into \( L_d \)
5: call procedure semaphore connection
6: return \( L_{\text{data}} \)
7: end while

According to Algorithm 11, at first, the scheduling of the EP algorithm is performed. This may include the check of the existing data in the database as a queue as shown in Algorithm 7. Afterward, the scan is conducted for a period in a timer as mentioned in the EP scheduling algorithm. After the timer has reached the particular period, the scan state is changed to stop. Afterward, the MCDM mechanism as a cost-efficient calculation during the connection to a particular slave device against multiple criteria consideration from the device’s parameters is executed. The result of this mechanism is a sorted list of devices based on their cost-efficiency against multiple criteria \( L_d \). After that, the connection to the nearby IoT devices is performed using semaphore as shown in Algorithm 4, and the data gathered from these connections are stored in the database.
5 Implementation

Implementation of the FEP with MCDM

The implementation of a combination between the EP scheduling algorithm and MCDM mechanism is shown in the Algorithm 12.

Algorithm 12 Fair Exhaustive Polling algorithm with MCDM

**Input:** Set of IoT nodes \( L_s \)

**Output:** Set of collected data \( L_{\text{data}} \)

**Require:** \( T_{\text{processing}} \leftarrow 60\text{s}, T_{\text{scan}} \leftarrow 10\text{s}, T_{\text{other}} \leftarrow 5\text{s} \)

1: while ProcessingTime has not been reached do
2: \hspace{1em} perform FEP Scanning Process
3: \hspace{1em} switch into stop scanning state
4: \hspace{1em} perform MCDM into \( L_d \)
5: \hspace{1em} set \( N \) into number of devices
6: \hspace{1em} \( T_{\text{remaining}} \leftarrow T_{\text{processing}} - T \)
7: \hspace{1em} set \( T \) into \( T_{\text{remaining}} \)
8: \hspace{1em} call procedure semaphore connection
9: \hspace{1em} return \( L_{\text{data}} \)
10: end while

According to Algorithm 12, at first, the scheduling of the FEP algorithm is performed. This may include the check of the existing data in the database as a queue as shown in Algorithm 8. Afterward, the scan is conducted for a period in a timer as mentioned in the FEP scheduling algorithm. After the timer has reached the specified period, the scan state is changed to stop. In this step, the calculation of the timer for each connection is calculated based on the remaining time \( T_{\text{remaining}} \) and the number of devices \( N \). Afterward, the MCDM mechanism as a cost-efficient calculation during the connection to an individual slave device against multiple criteria consideration from the device’s parameters is executed. The result of this mechanism is a sorted list of devices based on their cost-efficiency against multiple criteria \( L_d \). After that, the connection to the nearby IoT devices is performed using round robin as shown in Algorithm 6 and the data gathered from these connections are stored in the database.
Chapter 6

Performance Evaluation

In this Chapter, the explanation of experimental works for evaluating the performance of the proposed mobile gateway which is conducted during the thesis work will be discussed. First, the experimental tools used in the experiment will be explained. Then, there are five Sections explaining the experiments that show the abilities and some metrics of the App usage concerning QoS of a system that is being examined, including application overhead and performance experiments, memory consumption, specific vs. generic gateway, the experiment about the self-adaptive system and influence of the device mobility. In the last Section of this Chapter, the discussion of all the results which are obtained from the experiments will be explained in detail.

6.1 Experimental Tools

As mentioned in Chapter 3, the entities in the proposed mobile gateway consist of BLE sensors, a smartphone as a mobile gateway based on the Android and Google Firebase as a cloud system. The main intention of building such a system is to harness the smartphone to provide an energy-aware system which works as a mobile gateway to forward data from the sensors to the cloud and provide a generic gateway which can be used for any kinds of the already known BLE devices. As mentioned in Chapter 4 and 5, the connection mode of the BLE device is used in the proposed mobile gateway. Furthermore, by using this mode in the BLE, the smartphone connects to several BLE devices while during this connection, the data from each sensor is requested and gathered in a node repository component in the smartphone. After the implementation, the several experimental works have been conducted to measure parameters based on two QoS metrics, such as throughput and power consumption. To measure such parameters, experimental works with several scenario testing have been conducted to observe behavior and changes happened after the proposed mobile gateway is implemented in different experiment scenarios.

To do such experimental works for measuring the two QoS metrics, there are several tools which are used to obtain results in such a way that the resulted data can be provided conveniently. The resulted data will be shown in graphical charts which represent the two QoS metrics according to the experiment’s scenarios. These tools consist of:
6 Performance Evaluation

- **Trepn Profiler** is a third party application built by Qualcomm [18] which is used as a tool to profile the performance and on-target power on a mobile device. In this work, this App is used to measure the power consumption consumed during the proposed mobile gateway system is running on the Android system based smartphone. Figure 6.1a and 6.1b show the user interface of the Trepn and Trepn used for the App profiler.

---

Figure 6.1: Screenshot of the Android Apps used for experimental tools

(a) Trepn Profiler   (b) Trepn Profiling the Apps   (c) Bluetooth HCI snoop log

Figure 6.2: Wireshark network protocol analyzer tool
6.2 Application Overhead & Performance

• **Android Bluetooth Host Controller Interface (HCI) Snoop Log** is a protocol implemented in the Android system to provide a log of all Bluetooth HCI packets captured by the Android system. This log is stored in the internal memory of the Android-based smartphone with directory `/sdcard/btsnoop_hci.log`. This work uses this tool implemented in the Android system which is also available on the smartphone used for testing this experimental works. Figure 6.1c shows the settings for activating this protocol in the Android Developer option settings.

• **Wireshark** is a worldwide network protocol analyzer built by Riverbed [20] which is used to observe network activities at a microscopic level. This tool allows the developer to see the packets which are sent and receive by the system via a network. In this thesis work, Wireshark is used to analyze the log data output from the Bluetooth HCI Snoop Log in the Android system and see the number of packets from the BLE which are sent and received by the smartphone to be used as a mobile gateway. Figure 6.2 shows the user interface for Wireshark as a network protocol analyzer.

6.2 Application Overhead & Performance

This experiment is aimed to figure out the overhead resulted when the proposed mobile gateway App runs on the Android-based smartphone. Furthermore, the performance of the proposed mobile gateway App is also inspected in this experiment. The four scheduling algorithms and the other four combinations between the EP algorithm, FEP algorithm and the kinds of MCDM mechanism are observed in accordance to the changes the number of the nearby IoT devices. Setup of the experiment and the results will be discussed in the following Subsections.

6.2.1 Setup

**Power Consumption Experiment Setup**

For the setup in this experiment, several sensors are used to advertise BLE beacon data and to provide connection mode BLE with some data embedded into some services and characteristics. Furthermore, the smartphone in here is used to run the proposed design mobile gateway for IoT devices installed on the Android. Table 6.1 shows the devices used for this experiments.

<table>
<thead>
<tr>
<th>No</th>
<th>Device Name</th>
<th>Usage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Faros Board &amp; nRF 8001</td>
<td>IoT Sensors</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>LG Nexus 5X</td>
<td>nRF Connect for The Sensors</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Sony Xperia Z5</td>
<td>Device Test</td>
<td>1</td>
</tr>
</tbody>
</table>
In this experiment scenario, the algorithm setting in the system setting component is changed within the eight scheduling algorithms and some of the combination between scheduling algorithms and MCDM mechanisms, including semaphore (SEM), round robin (RR), exhaustive polling (EP), fair exhaustive polling (FEP), Exhaustive Polling with Analytical Hierarchy Process (EPAHP), Fair Exhaustive Polling with Analytical Hierarchy Process (FEPAHP), Exhaustive Polling with Weighted Sum Model (EPWSM) and Fair Exhaustive Polling with Weighted Sum Model (FEPWSM). Afterward, the number of IoT devices is also changed for each of the algorithms. For this experiment, the number of IoT devices is set to 2 devices, 4 devices, 6 devices, 8 devices, and 10 devices. Furthermore, for each experiment scenario, the QoS metric of power consumption is measured where the measurement of the experiment is done five times. Each of the measurement is done for 10 minutes.

The measurement of power consumption is done using Trepn Profiler in which the App is installed first on the smartphone, and the profiling of the proposed mobile gateway is done using this App. The result of this App is a .csv file which captures the power consumption in RAW data. The sampling, which is used to read the power spark as a result of the App activity, is done in 100 milliseconds. Subsequently, the results of the power consumption reading must be made an average value into 1 second before these results are made to be accumulated on each minute.

**Throughput Experiment Setup**

The setup for this experiment is almost the same as the power consumption experiment. The list of devices used in this experiment is also using the same devices as mentioned in Table 6.1. However, the QoS metric that will be measured in this experiment is the throughput.

The measurement of throughput is done using Bluetooth HCI Snoop Log in which this protocol has already been implemented in the Android-based smartphone. The result of this protocol is a .log file which can be opened with Wireshark as a network protocol analyzer. Subsequently, the result of the number of packet reading is accumulated on each minute, and the average throughput can be obtained by using Equation (6.1) as follows:

\[
\text{Average Throughput} = \frac{\text{Average Number of Packets}}{\text{Interval}}
\]

**6.2.2 Results**

**Power Consumption Results**

The power consumption (PC) defines the application overhead. The average number of the 5 data readings obtained from the measurements is calculated and plotted into graphs of each scenario as mentioned in the setup. Figure 6.3, 6.4, 6.5, 6.6 and 6.7 show the power consumption in cumulative compared with the time.
6.2 Application Overhead & Performance

Figure 6.3: Power consumption with number of devices = 2

Figure 6.4: Power consumption with number of devices = 4
Figure 6.5: Power consumption with number of devices = 6

Figure 6.6: Power consumption with number of devices = 8
From the results shown in Figure 6.3, 6.4, 6.5, 6.6 and 6.7, it is evidence that the semaphore based scheduling algorithms consume more power than the round robin based scheduling algorithms. Moreover, it can be seen that the more number of device available in the Bluetooth range, the power consumption consumed by the App increases.

Furthermore, the changes happened in between the algorithms which have the same scheduling algorithm basis are showing a slight difference each other. Thus, by using a look from the eyes in a glance at the graphs, it is hard to know the real differences happened because of the changes in the number of nearby IoT devices to connect with. Thus, the method of finding the average values on each algorithm from all number of IoT devices to connect with is done. From the average values, the ranking for the power consumption from the worst to the best value can be seen in Table 6.2.

Table 6.2: Ranking of scheduling algorithm based on power consumption

<table>
<thead>
<tr>
<th>No</th>
<th>Scheduling Name</th>
<th>Average PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Exhaustive Polling with Analytical Hierarchy Process</td>
<td>49.14 mW</td>
</tr>
<tr>
<td>2</td>
<td>Semaphore</td>
<td>48.39 mW</td>
</tr>
<tr>
<td>3</td>
<td>Exhaustive Polling</td>
<td>41.49 mW</td>
</tr>
<tr>
<td>4</td>
<td>Exhaustive Polling with Weighted Sum Module</td>
<td>33.35 mW</td>
</tr>
<tr>
<td>5</td>
<td>Fair Exhaustive Polling with Analytical Hierarchy Process</td>
<td>21.09 mW</td>
</tr>
<tr>
<td>6</td>
<td>Round Robin</td>
<td>15.75 mW</td>
</tr>
<tr>
<td>7</td>
<td>Fair Exhaustive Polling with Weighted Sum Module</td>
<td>14.47 mW</td>
</tr>
<tr>
<td>8</td>
<td>Fair Exhaustive Polling</td>
<td>12.79 mW</td>
</tr>
</tbody>
</table>
6 Performance Evaluation

Throughput Results

Figure 6.8: Throughput with number of devices = 2

Figure 6.9: Throughput with number of devices = 4
6.2 Application Overhead & Performance

Figure 6.10: Throughput with number of devices = 6

Figure 6.11: Throughput with number of devices = 8
The throughput determines the application performance. Figures 6.8, 6.9, 6.10, 6.11, 6.12 show the throughput in cumulative compared with the time. As illustrated from the results given above, it is evidence that the semaphore based scheduling algorithms provide a better throughput value than the round robin based scheduling algorithms. Moreover, it can be seen that the more number of devices available in the Bluetooth range, the more throughput is obtained by the App.

Furthermore, the changes happened in between the algorithms which have the same scheduling algorithm basis shows a slight difference each other. Hence, the method of finding the average value on each algorithm from all number of IoT devices to connect with is done. From the average value of the throughput on each scheduling algorithm, the ranking for throughput from the best to the worst value can be seen in Table 6.3.

<table>
<thead>
<tr>
<th>No</th>
<th>Scheduling Name</th>
<th>Average throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Exhaustive Polling with Weighted Sum Module</td>
<td>101.18 packet/s</td>
</tr>
<tr>
<td>2</td>
<td>Exhaustive Polling with Analytical Hierarchy Process</td>
<td>97.78 packet/s</td>
</tr>
<tr>
<td>3</td>
<td>Exhaustive Polling</td>
<td>84.03 packet/s</td>
</tr>
<tr>
<td>4</td>
<td>Semaphore</td>
<td>71.51 packet/s</td>
</tr>
<tr>
<td>5</td>
<td>Round Robin</td>
<td>57.38 packet/s</td>
</tr>
<tr>
<td>6</td>
<td>Fair Exhaustive Polling with Analytical Hierarchy Process</td>
<td>54.83 packet/s</td>
</tr>
<tr>
<td>7</td>
<td>Fair Exhaustive Polling with Weighted Sum Module</td>
<td>54.18 packet/s</td>
</tr>
<tr>
<td>8</td>
<td>Fair Exhaustive Polling</td>
<td>50.98 packet/s</td>
</tr>
</tbody>
</table>
6.3 Memory Consumption

This experiment is intended to find out the consumption of the memory when the proposed mobile gateway App operates on the Android-based smartphone. This is also an essential consideration for the energy-aware parameter because of the more memory used by an App, the more energy consumed by the App. In this experiment, the comparison of the memory consumption when the system uses the self-adaptive system without uploading data to the cloud and vice versa. The setup of the experiment and the results will be discussed in the following Subsections.

6.3.1 Setup

Setup for this experiment uses ten sensors for advertising BLE packets and providing connection mode BLE which embeds data into services and characteristics. These sensors consist of six Faros Board & nRF 8001 module and four nRF Connect simulators. The smartphone here installs the proposed mobile gateway system and is set to use the self-adaptive system by MAPE to select a suitable algorithm depending on several criteria. Table 6.4 shows the devices used for this experiment.

<table>
<thead>
<tr>
<th>No</th>
<th>Device Name</th>
<th>Usage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Faros Board &amp; nRF 8001</td>
<td>IoT Sensors</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>LG Nexus 5X</td>
<td>nRF Connect for IoT Sensors</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Sony Xperia Z5</td>
<td>Device Test</td>
<td>1</td>
</tr>
</tbody>
</table>

For the memory consumption experiment, a scenario is set for first to run the self-adaptive system with MAPE but without using the cloud service to upload the data. Another scenario is set to run on the 2nd run which is also using the self-adaptive system with MAPE using the cloud service to upload data. Both scenarios are performed for experiment with an interval of ten minutes. In the first two minutes, the number of IoT devices is set to only two IoT devices. Whereas within these ten minutes, every two minutes the memory consumption data is taken and then the number of IoT devices is added by two IoT devices. Thus, there are in total ten IoT devices for this experiment. While the experiment is being conducted, the smartphone is connected to the computer because of the monitoring of memory usage is done using Android Profiler which is embedded into the Android Studio for programming the Android App. The Android Profiler gives the ability for the developer to trace the real-time data for the third party App’s memory, CPU and network activity [56].

6.3.2 Results

The memory consumption also defines the energy consumed by the App on the device. The reading from the memory consumption parameter using the Android Profiler is done one time.
6 Performance Evaluation

per scenario. After that, different from the previous experiments, the data of the memory consumption is plotted on a graph whereas each scenario is plotted into the same graph to compare the result each other. The data of each scenario’s memory consumption is plotted based on the time measured every two minutes.

Figure 6.13: Memory consumption comparison

Figure 6.13 shows the memory consumption of the system using MAPE without uploading to the cloud is slightly lower than the same system that employs the uploading mechanism. For the average memory consumption within 10 minutes of time, the without cloud scenario resulted in the memory consumption of 64.06 MB. Meanwhile, the other scenario shows the memory consumption of 64.9 MB. However, after some time, the figure for memory consumption between these two scenarios shows almost the same value.

6.4 Specific Vs Generic Gateway

In this experiment, the comparison between a specific App which is used by a specific BLE device and the proposed design App which is used to obtain data from several BLE devices will be examined. This experiment is important to find out what precisely the predominance from the proposed design App compared to the specific App built by the BLE device’s manufacturer. The setup of the experiment and the results will be discussed in the following Subsections.
6.4 Specific Vs Generic Gateway

6.4.1 Setup

For this experiment, setup is done using two different smart devices and one wearable device that works as the sensors, e.g., Revogi Smartmeter, Coospo Cycling & Cadence Sensor and Xiaomi MiBand 2. These devices are currently available in the marketplace and provide on each device a specific gateway App installed on the Android. Furthermore, the smartphone of Sony Xperia Z5 works as a device test which runs the proposed design of mobile gateway for IoT devices. Table 6.5 shows devices used for this experiments.

Table 6.5: Some utilities used for specific vs. generic gateway experiment

<table>
<thead>
<tr>
<th>No</th>
<th>Device Name</th>
<th>Usage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Revogi Smartmeter</td>
<td>IoT Sensor</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Xiaomi MiBand 2</td>
<td>IoT Sensor</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Coospo Cycling Sensor</td>
<td>IoT Sensor</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Sony Xperia Z5</td>
<td>Device Test</td>
<td>1</td>
</tr>
</tbody>
</table>

To achieve the goal of getting the comparison between a specific gateway for a specific IoT device and a generic gateway for multiple IoT devices, there are two different experiment scenarios which implement the devices as mentioned in Table 6.5. Firstly, the scenario of experimenting the specific gateway using the available sensors. On each sensor, the measurement of two QoS metrics, such as power consumption and throughput, is done two times. On each of the measurement, the timer is set to 10 minutes of measurement. For Xiaomi MiBand 2, the App of Xiaomi MiFit is installed on the device test. Furthermore, for Coospo Cycling and Cadence Sensor, the App of Waaho Fitness is used and installed on the device test. Subsequently, for Revogi Smartmeter, the App of Revogi Smart meter is installed on the device test. After that, on each of these devices, the measurement of two QoS metrics are done respectively.

Secondly, the scenario of experimenting the generic gateway using the available sensors. The main difference between this scenario with the previous scenario is all the available sensors as mentioned in Table 6.5 are used against the proposed mobile gateway for IoT devices. Hence, the smartphone which has been installed with the mobile gateway will try to connect to the sensors using a middle scheduling algorithm—a scheduling algorithm that has a middle value between the best power consumption and the best throughput—. As agreed upon, the proposed design mobile gateway uses the Fair Exhaustive Polling with Analytical Hierarchy Process (FEPAHP) as the scheduling algorithm. Furthermore, same as the previously mentioned experiment scenario, the measurement of two QoS metrics is also done with the same settings.

Nevertheless, in this experiment, the plot of the result is different from the previous experiments. After obtaining the data from all measurements for each experiment scenario, the precise amount of data is agreed upon to a certain number of bytes. In this experiment, the amount of data is set to maximum 10 kB to be taken into consideration for the power consumption usage and the amount of time used for the system to reach this amount of data.
6 Performance Evaluation

(Latency). After that, these data are plotted into two different graphs. One graph is used to compare the power consumption consumed by the App to reach a certain amount of data per kilobytes, and the other graph is used to compare the latency for the App to reach a certain amount of data per kilobytes.

6.4.2 Results

The specific and generic gateway experiment defines the term of how the proposed mobile gateway working parameters compared to the App designed by an IoT device’s manufacturer. All of the specific Apps designated for a specific device are available on the Google, and these Apps can be downloaded from it and installed on the Android smartphone.

Figure 6.14: Energy vs. data comparison

Figure 6.14 shows the comparison between power consumption and the specific amount of data. From this graph, it is evident that for most of the specific gateway Apps the number of power consumption consumed by each App does not significantly increase with the changes in the size of data. However, a slight increase happened on the Xiaomi MiBand 2 data after reaching 8 kB of data amount. Henceforth, this graph shows also the proposed design generic gateway increases the number of power consumption significantly after 5 kB of data amount.
Meanwhile, Figure 6.15 shows the comparison between the latency and the precise amount of data. From this graph, it is evident that for the specific gateway cases, a significant increase of latency time for reaching a certain amount of data happens differently in between the different Apps. For the Revogi Smartmeter, after reaching 5 kB of data amount, the latency increase significantly for reaching the further amount of data and at the end of 10 kB data amount, it has the highest latency of 7.5 minutes for obtaining such amount of data. The same case also happens for Coospo Cycling Sensor, although it has a slightly better figure compared to the Revogi Smart meter that it has almost 6 minutes of latency for reaching 10 kB data amount. Furthermore, Xiaomi MiBand has a better figure compared to the previous two sensors that it reaches 10 kB data amount with approximately 3.5 minutes. Subsequently, from Figure 6.15, it is evident that for the proposed design of generic gateway, there is a significant increase in latency after 3 kB to 4 kB. Nevertheless, the latency does not increase significantly after that, and at the end of 10 kB data amount, it has the best latency by obtaining the data amount with only approximately 1.2 minutes.

6.5 Self Adaptive System with MAPE

The objective of this experiment is to figure out the influence of the self-adaptive system with MAPE towards the proposed mobile gateway. In this particular experiment, the system will be observed by using the MAPE algorithm compared with a particular algorithm which has an adequate value in the throughput performance and energy efficiency. The setup of the experiment and the results will be discussed in the following Subsections.
6.5.1 Setup

For this experiment, setup is done using several IoT devices which act as the sensors for advertising BLE packets and providing connection mode BLE that embeds data into services and characteristics. Another setup is set for the device test. The device used for the experiment is a smartphone of Sony Experia Z5 which installs the proposed mobile gateway. The setup of devices used in this experiment can be seen in Table 6.6.

<table>
<thead>
<tr>
<th>No</th>
<th>Device Name</th>
<th>Usage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Faros Board &amp; nRF 8001</td>
<td>IoT Sensors</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>LG Nexus 5X</td>
<td>nRF Connect for IoT Sensors</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Sony Xperia Z5</td>
<td>Device Test</td>
<td>1</td>
</tr>
</tbody>
</table>

There are several experiment’s scenarios which underlying this experiment. Firstly, the scenario of the experiment without using the self-adaptive system with MAPE. This scenario is divided into two conditions which relate to which the QoS metric is chosen based on the calculation of the WSM mechanism for the decision making. When the QoS metric of power consumption is chosen (dominant power consumption), the calculation of the best algorithm to use in this scenario is done with the weight scale for power consumption is given more than the throughput. In contrast, if the QoS metric of throughput is chosen (dominant throughput), more weight scale for throughput is given than the power consumption. From the calculation using the WSM mechanism, there is a rank of scheduling algorithms resulted from it. On each scenario, the middle value from the rank of scheduling algorithm using WSM mechanism is taken. Furthermore, in each scenario, the two QoS metrics are also observed. Thus, there are four measurements for this particular scenario. As agreed upon, for the first measurement about dominant power consumption, the scheduling algorithm used for running the mobile gateway is Fair Exhaustive Polling with Analytical Hierarchy Process (FEPAHP). On the other hand, for the dominant throughput as the second measurement, the scheduling algorithm is Exhaustive Polling with Analytical Hierarchy Process (EPAHP).

Secondly, the scenario of the experiment with the usage of the self-adaptive system with MAPE. In this scenario, the self-adaptive system selects the most suitable algorithm based on the situations happened during the measurement process. Afterward, the same mechanisms of the measurement are performed in this experiment scenario.

Both scenarios are performed for experiment with an interval of ten minutes. In the first two minutes, the number of IoT devices is set to only two IoT devices. Whereas within these ten minutes, every two minutes then the number of IoT devices is added by two IoT devices. Thus, there are in total ten IoT devices for this experiment. After obtaining the data from all measurements for each experiment scenario, these data are plotted into graphs. However, because of two conditions available on both scenarios, the graphs are also plotted based on these two conditions to which one QoS metric is dominant from the other metric.
6.5.2 Results

Dominant Power Consumption Weight

The experiment of the self-adaptive system using defines how the self-adaptive system influences the overall proposed mobile gateway compared to the utilization of one specific scheduling algorithm. As previously mentioned in the setup section, the results are divided based on the conditions of which QoS metric is more dominant than the other. The results of power consumption and throughput metrics for condition of dominant power consumption can be seen in Figure 6.16 and 6.17 respectively.

In Figure 6.16, it is evidence that in the condition of dominant power consumption, the cumulative power consumption for the scheduling algorithm of Fair Exhaustive Polling with Analytical Hierarchy Process (FEPAHP) is higher than the self-adaptive system using MAPE. Meanwhile, the throughput resulted from this experiment scenario, as shown in Figure 6.17 shows the adaptive system using MAPE has a lower throughput compared to the FEPAHP scheduling algorithm. Thus, it is safe to say that in the condition of power consumption is more dominant than the throughput, the proposed mobile gateway with the usage of the self-adaptive system consumes less power consumption and produces lower throughput than the system that uses only a particular scheduling algorithm.

![Figure 6.16: Comparison of power consumption with MAPE vs. without MAPE (dominant power consumption)](image)
Figure 6.17: Comparison of throughput with MAPE vs. without MAPE (dominant power consumption)

**Dominant throughput Weight**

Figure 6.18: Comparison of power consumption with MAPE vs. without MAPE (dominant throughput)
6.6 Mobility of The Smartphone

The results of power consumption and throughput metrics for condition of dominant throughput can be seen in Figure 6.18 and 6.19 respectively. In Figure 6.18, it is evidence that in the condition of dominant throughput, the cumulative power consumption metric for the scheduling algorithm of Exhaustive Polling with Analytical Hierarchy Process (FEPAHP) is lower than the self-adaptive system using MAPE.

Meanwhile, the throughput metric resulted from this experiment scenario, as shown in Figure 6.19 shows the adaptive system using MAPE has higher throughput compared to the EPAHP scheduling algorithm. Thus, it is safe to say that in the condition of throughput is more dominant than the power consumption, the proposed mobile gateway with the usage of the self-adaptive system consumes more power consumption and produces higher throughput than the system that uses only a particular scheduling algorithm.

6.6 Mobility of The Smartphone

The primary purpose of this experiment is to understand the influence of the mobility for the proposed mobile gateway. Using the self-adaptive system with MAPE, the system is observed by moving the smartphone with several amounts of speeds which are the case of the smartphone use in daily life. The setup of the experiment and the results will be discussed in the following Subsections.
6 Performance Evaluation

6.6.1 Setup

Table 6.7: Some utilities used for mobility of smartphone experiment

<table>
<thead>
<tr>
<th>No</th>
<th>Device Name</th>
<th>Usage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Faros Board &amp; nRF 8001</td>
<td>IoT Sensors</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>LG Nexus 5X</td>
<td>nRF Connect for IoT Sensors</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Sony Xperia Z3</td>
<td>nRF Connect for IoT Sensors</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Xiaomi MiBand 2</td>
<td>IoT Sensor</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Revogi Smartmeter</td>
<td>IoT Sensor</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Sony Xperia Z5</td>
<td>Device Test</td>
<td>1</td>
</tr>
</tbody>
</table>

In the setup for this experiment, the total of 10 IoT devices is used as the sensors to provide BLE advertising packets and connection mode BLE with the sensor data wrapped into the services and characteristics of the BLE GATT profile. From these 10 IoT devices, there are 6 devices of Faros Board combined with nRF 8001 Module, one smart device, one Smart meter and two smartphones that simulate the BLE devices using the App from Nordic Semiconductor of nRF Connect [42]. The setup of devices used in this experiment is shown in Table 6.7.

In this experiment, the device test is made to move alongside the user’s movement. There are three scenarios of the experiment that have been set for this experiment based on the speed of the user’s walking: 0.5 m/s, 1 m/s and 2 m/s. To make such a movement, the step
of the tester is measured using a ruler for an experiment in advance. Thus, the user should have the same movement while doing the measurement for a particular speed parameter.

Furthermore, this experiment is conducted in an indoor place alongside the 2nd floor of the Informatik building the University of Stuttgart. Alongside this indoor place, the sensors are located randomly to simulate the real case of the movement of a user and the behavior of the proposed mobile gateway when it is used for daily life. Subsequently, for each measurement, the time used for conducting the measurement is set to 5 minutes. The indoor floor plan used for the measurement of this experiment can be seen in Figure 6.20.

As depicted in Figure 6.20, the placement of the sensors are done randomly. Also, the movement of the user along each measurement is illustrated with the arrow line. The movement itself is done to make a rectangle line alongside the building’s indoor floor plan. While the user is conducting the measurement, the smartphone runs the proposed system App with the self-adaptive system of MAPE. The self-adaptive system selects the suitable algorithm for every available situation, including in this experiment, the possibility of number of devices changes is quite often to happen because of the smartphone’s movement itself may let the device to pass several of the nearby IoT devices which have not been scanned before or the connection is established but the discovery of the available services in the BLE has not been accomplished. Thus, the loss of communication is possible to happen, and this might have an impact on the proposed design App of the mobile gateway.

### 6.6.2 Results

![](image.png)

Figure 6.21: The effect of mobility towards throughput
6 Performance Evaluation

The mobility of the smartphone experiment defines the influence of the movement on the proposed design App towards the BLE wireless communication used by it. In this experiment, only one QoS metric that is observed against three experiment scenarios. Figure 6.21 shows the result of the measurement of this experiment in which the cumulative throughput for each scenario is plotted against the time used for conducting the measurement.

As shown in Figure 6.21, it is evidence that mobility of the smartphone gives an influence to the proposed mobile gateway for which the faster the speed of the movement of the smartphone, there is a slight change for the throughput towards the decreasing of its value. In the first 2 minutes of measurement, all three scenarios of speed have almost the same throughput value. Nevertheless, after 3 minutes, the speed of 2 m/s starts to decrease its throughput value. Also, the change is shown continuously until, in 5 minutes, this scenario of speed becomes among the lowest value. Thus, it is evidence that the faster the speed of the smartphone’s movement, the figure of throughput decreases slightly.

6.7 Discussion

The implemented system design for an energy-aware mobile gateway for IoT devices is evaluated in six experiments: application overhead, application performance, memory consumption, specific vs. generic gateway, the influence of self-adaptive system and the effect of mobility of the smartphone. Almost all experiments have shown that the designated system has performed as expected.

In the power consumption experiment, as mentioned in the ranking of the scheduling algorithm with power consumption consideration, the FEP algorithm has the best power consumption compared to the other algorithms. It is because of this algorithm employs intelligent scheduling with the device states of active and inactive. Furthermore, the time for scan process in the 2nd cycle and so forth for this algorithm has also been suppressed to reduce the power consumption. Meanwhile, the EPAHP algorithm becomes the worst power consumption because of the usage of AHP mechanism for MCDM which utilizes matrix calculation for the eigenvector. Moreover, the connection method of this algorithm is based on the SEM algorithm. Because of these two factors, the power consumption for this particular scheduling algorithm becomes very high.

In the meantime, for the throughput experiment, the EPWSM algorithm obtains the highest value compared to other scheduling algorithms. The reason behind this result is because of the basis of EP algorithm is using the SEM algorithm which implies more often of connecting and obtaining data to several IoT devices compared to the RR based algorithm. Furthermore, the usage of MCDM for decision making with WSM, which is the simplest version of MCDM compared to the others, has also contributed to the increase of the throughput parameter. This is because the MCDM algorithm calculates the cost-efficient of connecting to a specific IoT device. Thus, by prioritizing the connection to a device, the more efficient of connection can be performed resulted in the increasing throughput value. On the other hand, the FEP algorithm shows the lowest value of throughput. This is because of the basis of RR scheduling
algorithm that makes the FEP connects and disconnects to several IoT devices slowly. Hence, the throughput of this algorithm drops to the worst number.

From the experiment of the memory consumption, it can be implied that the impact of the memory consumption for the uploading data to the cloud is not that significant compared to the not uploading scenario. The reason behind this is the usage of a garbage collector in java which cleans some unnecessary objects after some amount of time.

Henceforth, from the previous explanations regarding the experiment of the specific vs. generic gateway, the proposed mobile gateway consumes more power for connecting to all of the three IoT devices using the predetermined scheduling algorithm than the specific App for the specific IoT device. This is not a result which is expected of a designated system about an energy-aware with generic mobile gateway that can be used for connecting to many IoT devices concurrently due the overhead happens during the phase of connecting and disconnecting to several IoT devices which happens simultaneously while some of the Apps made by the manufacturers have even just done the scanning one time connect to the device after a long period. However, another interesting fact which is shown in Figure 6.15 is the proposed mobile gateway has the least latency compared to the specific App for the specific IoT device. This is because the more devices that the system can connect with, the more data can be obtained from these IoT devices simultaneously.

Afterward, in the self-adaptive system with MAPE, there are two different results regarding which QoS metric has a dominant result. In the power consumption dominant, the result from the self-adaptive system usage has shown both in power consumption and throughput lower value than the chosen scheduling algorithm for power consumption dominant which is the FEP AHP algorithm. Meanwhile, in the throughput dominant, the self-adaptive system has shown in both QoS metrics higher value than the chosen scheduling algorithm for throughput dominant which is the EPAHP algorithm. Thus, it is safe to say that in both cases, the self-adaptive system with MAPE has outperformed the standard scheduling algorithms that in both dominant cases, the best scheduling algorithm has been chosen.

Lastly, by the result which is shown in Figure 6.21, it is safely to say that the proposed mobile gateway with a self-adaptive system by MAPE, the effect of the mobility from the smartphone happens on the QoS metric of throughput result which is affecting a slight drop of throughput value with the increasing of the speed movement from the smartphone. It is logical because of increasing the speed movement, the possibility of the loss of connection or the inability to scan the device happens. However, it is also evidence that the self-adaptive system can still handle the case with only a slight change after some time.
Chapter 7

Conclusion

7.1 Conclusion

The term of connecting things from all around the world has driven the Internet for further innovation and has opened a gate towards many opportunities lied on the Internet of Things (IoT) which may bring smart living on a smart environment [4]. Harvesting data from many applications in many fields —by embedding computational capabilities to some objects that have valuable data— ends up as a trend that becomes more and more inevitable in the future especially when the internet infrastructures become more establish and the price of the devices used for the sensors become cheaper. Henceforth, there exists a question about how the collected data which has valuable meanings can be presented to the users? The answer is by the use of cloud computing, By distributing data to the cloud, the further process and analyze the collected data can be done without thinking that the processing and analyzing the collected data can be a burden for such a sensor which has limited resources.

Furthermore, when talking about IoT, data acquisition from the environment by the sensors, and cloud computing, there is still a question between these terms on how the data should be sent to the cloud using the internet? This question can be answered by the usage of the direct connection to the internet right from the sensors and the usage of an intermediary device which will forward the data from the sensors to the internet or so-called a gateway. The first approach may have some problems regarding the limited resources hinder the sensor itself to have a direct connection to the internet due to having this ability requires some massive energy consumption. On the other hand, some of the cheap sensors are designed to work independently with a lifetime of a couple of months or a year without having to replace or recharge their batteries. Thus, the latter approach may become a better option. However, having an intermediary device approach comes with the problems underlying the system which is known as gateway problems, such as the siloed versions, cross-platform device connectivity, and energy consumption consumed by the smartphone for exchanging data using a wireless connection.

To tackle the aforementioned issues, in this work, a system for an intermediary device called a mobile gateway with an energy-aware paradigm that providing a generic gateway has been proposed. The focus of this work will be on the design of such a system with the implementation of an application layer which runs on the Android. Aiming for a goal of an energy-aware
system with the performance consideration, there are some design considerations which become the basic system design of the designated system. Some of the design considerations which have been done in this work, including a *Lazy First* design approach, some platform feature implementations to manage the consumption of the battery itself, Thread Design Strategies, service interface approach, the scheduling for BLE wireless connection with some MCDM mechanisms and lastly the usage of self-adaptive system with MAPE for selecting the suitable scheduling algorithm.

Notably, for the energy-aware design consideration, the scheduling algorithm for BLE connection and data gathering with the usage of MCDM mechanisms become the main focus of this work. There are four scheduling algorithms which are used in this work, such as semaphore (SEM), round robin (RR), exhaustive polling (EP) and fair exhaustive polling (FEP). The main difference between these four basic algorithms is in the scanning and connection to some IoT devices. Furthermore, there are two MCDM mechanisms for calculating the cost-efficiency for a connection to be made to a specific IoT device, including Analytical Hierarchy Process (AHP) and Weighted Sum Model (WSM). By combining these two methods, eight possible algorithms are applied to this work. Furthermore, for creating a more efficient system which also considers the current situation of the device and deciding which suitable scheduling algorithm performed, a self-adaptive system with MAPE is also implemented.

Henceforth, to verify the aforementioned energy-aware design consideration in proposed mobile gateway and implementation, several experiments are conducted. Firstly, the experiment of application overhead resulted in the FEP algorithm, which consumes 12.79 mW on average power, becomes the lowest power consumption compared to other scheduling algorithms. Secondly, the experiment of application performance resulted in the EPWSM algorithm that provides the highest throughput, which obtains 101.18 packets/s on average throughput, compared to other algorithms. Meanwhile, the third experiment talks more about the memory consumption consumed by the proposed design system with the MAPE algorithm. The results show the of the system to run the MAPE algorithm with uploading consumes 64.9 MB of memory, while without uploading to the cloud, the system consumes 64.04 MB of memory. Thus, there is only a slight difference in memory consumption consumed by the system.

Then, the fourth experiment discusses more the specific gateway compared to the generic gateway. These experimental results have informed that the proposed mobile gateway consumes more power for connecting to all of the three IoT devices using the predetermined scheduling algorithm than the specific App for the specific IoT device. Nevertheless, the proposed mobile gateway has the least latency compared to the specific App for the specific IoT device. Next, the experiment about the influence of the usage of the self-adaptive system with MAPE shows results that support one to another scenario. In the power consumption dominant, the result from the self-adaptive system usage has shown both in power consumption and throughput lower value than the chosen scheduling algorithm for power consumption dominant. In contrast, for the throughput dominant, the self-adaptive system has shown in both QoS metrics higher value than the chosen scheduling algorithm for throughput dominant.

Lastly, from the experiment of mobility of the smartphone, by the proposed mobile gateway implemented with the self-adaptive system by MAPE, the effect of the mobility from the smartphone with the increasing of the speed of movement shows a slight change of the
throughput value after some time of the measurement in the experiment. Thus, from some experiment conducted in this work, it is safe to say that the proposed mobile gateway of a mobile gateway is exceptionally efficient for handling data forwarding from multiple BLE sensors to the cloud services with energy awareness.

7.2 Future Work

The implementation of this work has only considered forwarding the data to the cloud without further implementations on the cloud which uses the available data by the users. Hence, the implementation of web service interfaces is highly desired.

Furthermore, while working with the real IoT devices, there is some security implemented on them which consider the encryption of the critical information. This security decryption on the smartphone used for the implementation of the designated system must also be implemented which in this case the manufacturer of these IoT devices must be contacted to agree how to decrypt this information to become visible to be used for the service interface component.
References


References


References


Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature