ASHESI UNIVERSITY COLLEGE

EVALUATION OF THE BLUETOOTH MESH MANAGED FLOODING ALGORITHM USING SIMULATION

UNDERGRADUATE APPLIED PROJECT

B.Sc. Computer Science

Job Nkuusi Mwesigwa

2018

EVALUATION OF THE BLUETOOTH MESH MANAGED FLOODING ALGORITHM USING SIMULATION

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

Job Nkuusi Mwesigwa

April 2018

DECLARATION

I hereby declare that this dissertation is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature: ……..

Candidate's Name: Job Nkuusi Mwesigwa

Date: ……..

I hereby declare that the preparation and presentation of the thesis were supervised in accordance with the guidelines on supervision of thesis laid down by Ashesi University College.

Supervisor's

Signature……

Supervisor's Name: Nathan Amanquah, Ph.D.

Date:…….

ACKNOWLEDGMENTS.

Special thanks to my supervisor, Dr. Nathan Amanquah for guiding and challenging me to explore more throughout the project timeline. I also appreciate him for suggesting this topic for me to consider and research it.

ABSTRACT

Bluetooth Special Interest Group released a new version of Bluetooth at the end of 2016 which is called Bluetooth 5. The version had superior features such as increased range which is 4 times the range of Bluetooth 4.2 and improved speed. It was expected to revolutionaries the use of IoTs. However, it used a star topology which limited its range within which it could operate. Both routing and flooding solutions from both Bluetooth SIG and academia have been suggested which led to the development of Bluetooth Mesh. Bluetooth Mesh uses a flooding algorithm which uses a large number of nodes just to send a single message which indicates that more energy is used. This project develops a simulator for both routing and flooding algorithms and evaluates their ability to facilitate Bluetooth Mesh functionalities and energy efficiency in terms of relays done to deliver a single message. From this simulation, although managed flooding uses a lot of energy to transmit, it is the most efficient because it takes care of the individual power needs of the nodes in the network.

Keywords: Bluetooth Mesh, Internet of Things, Routing, Managed Flooding, Simulation.

Table of Contents

Table of Figures

List of Tables

Chapter 1: Introduction

1.1 Background

Advancements in technology have led to a smart era where many electrical devices can easily be controlled remotely. This has increased the desire for users to have full control over all their electrical devices and systems without carrying them physically. This connection of devices, systems, and services is technically referred to as the Internet of Things (IoT). The growth of the IoT is on an increase which affirms that "by 2025 Internet nodes may reside in things that we use every day" like furniture. (Sezer, Dogdu, & Ozbayoglu, 2018). This will give power to the user because of the ability to communicate with most of the objects such as furniture, appliances, transport systems, etc. without having any form of physical contact with them. The connection to this internet could be wireless or may involve a physical media. The wireless communication interface media include Zigbee, Bluetooth, and Wi-Fi (Sethi & Sarangi, 2017).

Among others, Bluetooth has consistently shown progress in improving the mode of communication between IoTs as it has progressed from the initial Bluetooth release to the current Bluetooth Mesh. From the initial Bluetooth Classic which had limitations in the coverage range and data rate, Bluetooth Special Interest Group (SIG) introduced the Bluetooth Low Energy which advanced the scope of Bluetooth 4.1 network by increasing the number of nodes that can be connected in a network. Bluetooth Classic was made up of the master-slave connections which form a piconet which is illustrated in figure 1.1. Bluetooth Low Energy (LE) supports these ideas and creates a network of many piconets connected together with a bridge node as the connection between the different piconets. The enhancement of Bluetooth LE had topology limitations which hindered the increase of the link range. There was also limitation in path diversity since only one path, i.e. via the bridge node, was used to send data from one piconet to another (Darroudi &

Gomez, 2017). In addition, failure of a master node or a bridge node suggested a failure of a big portion of the scatternet since all nodes connected to either node would automatically be off. Therefore, the need to rectify the limitations of Bluetooth Low Energy ignited the research by the Bluetooth SIG leading to the release of Bluetooth Mesh in July 2017.

1.2 Bluetooth Mesh

1.2.1 Evolution of Bluetooth Topology

As explained in the previous section, Figure 1.1 below illustrates the network topology of Bluetooth classic which had a master node at the center where all other nodes are connected. In Figure 1.1, node X is the master node and all communications go through it before they reach their destination.

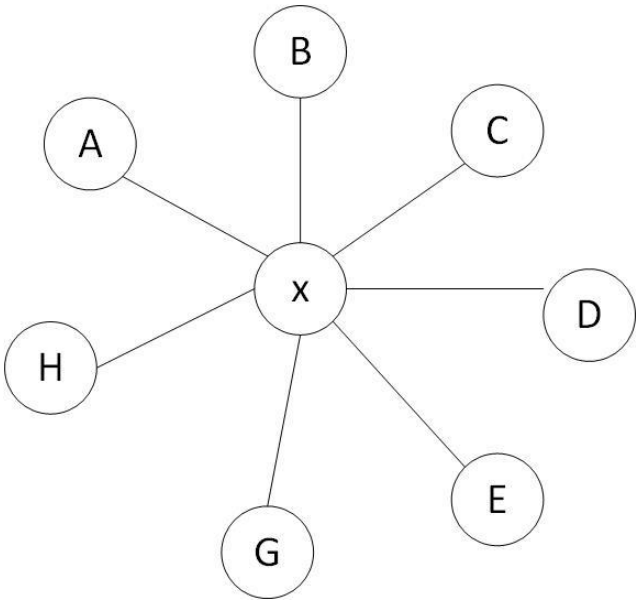

Figure 1.1 Piconet

Bluetooth 4.2 supported the increased scope of the network by using a bridge node to connect to other piconets to form a scatternet. The bridge node can either be a master or a slave node which is connected to more than one piconet. Figure 1.2 illustrates two piconets with node X

and node Y as the master nodes of the individual piconets. Therefore, in Figure 1.2 below, a message can flow from A to L but if node E fails, then messages cannot flow from one piconet to another.

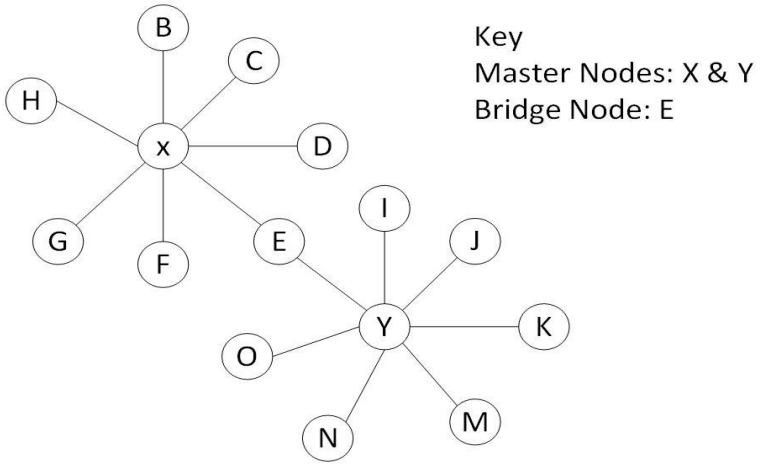

Figure 1.2 Scatternet

The Bluetooth Mesh Topology eliminated the node dependencies of the start topology where the network strongly relies on the master nodes and the bridge nodes. Figure 1.3 illustrates the Mesh network topology which Bluetooth Mesh uses. In the network below, a new node can be added anywhere and there are many paths a message can take from one node to another.

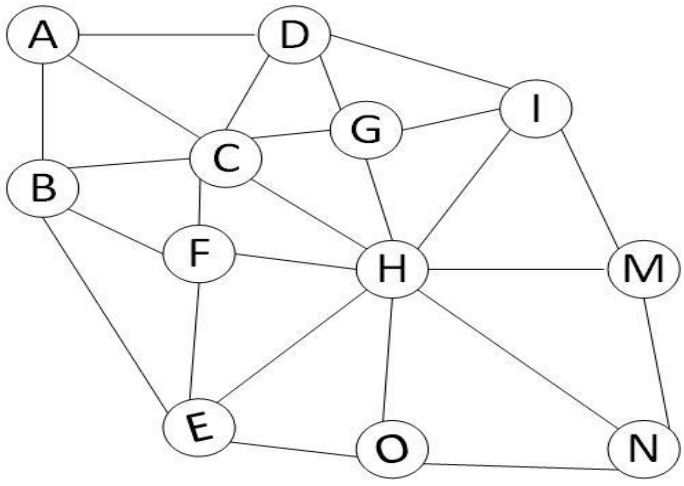

Figure 1.3 Mesh Network Topology

Bluetooth Mesh revolutionized the use of Bluetooth in IoT by switching from a one-to-one communication to a many-to-many communications giving the nodes in the mesh network the ability to communicate with most of the devices in the network. In the mesh network, one node, the provisioner, has the task of adding other nodes to the network using the 5 steps that make up the provisioning process. These steps are beaconing, invitation, exchanging public keys, authentication, and distribution of the provisioning data (Bluetooth SIG, 2017). When nodes are added to the network, they can take up one or more of the following features;

- Relay node which pass on information to other nodes
- Proxy nodes which are Bluetooth LE nodes with the ability to interact with nodes that possess the mesh stack
- Friend and low power nodes. A relationship where the friend nodes receive messages meant for the low power nodes when the low power nodes sleep.

The Bluetooth Mesh Architecture is made of 7 layers with the top 6 layers sitting on the Bluetooth Low Energy stack as the bottom layer. When a message is sent from one node to another, the sending node broadcasts the message to all nodes in its direct radio range which also do the same until the message is delivered to the intended receiver. Therefore, messages are sent by broadcasting them through a process called managed flooding rather than being routed through one path to a specific destination which is the case for most traditional networks that use Open Shortest Path First (OSPF) routing protocol. The report explains more about the managed flooding algorithm in section 2.2.2. The different layers of the Mesh Architecture are responsible for determining whether the message should be relayed or delivered to any node that has received it.

1.3 Research Problem

In flooding algorithms, the process of sending messages from one node to another is done by broadcasting the message to all nodes connected to the sending node, this process is repeated continuously until the message gets to its destination. This is advantageous because the message will always be delivered through the shortest path available and it also provides path diversity since the message can be sent through any of the available paths (Darroudi & Gomez, 2017). However, this suggests that sending one message is expensive in terms of energy used by the entire mesh network because most of the nodes use their energy to transmit signals to all nodes in their direct radio range. Therefore, given a network with nodes sensitive to power needs, continuous transmission of signals may consume most of the battery of these power sensitive nodes which may require them a constant supply of power. On the other hand, routing algorithms that require a fixed path may save energy but may be at risk of path failure in case one of the nodes along the message path malfunctions. In addition, most routing algorithms use more memory because of the need to store paths in form of routing tables.

Bluetooth SIG has set up measures to enhance message broadcasting in the network through the managed flooding process. These include heartbeats to monitor active nodes at a given time, time to live (TTL) which monitors the number of hops to avoid sending one message infinitely, message cache which keeps track of all messages received by a given node, and lastly a friend node that backs up messages for the low power nodes (Bluetooth SIG, 2017). The use of a flooding algorithm threatens the energy efficiency requirements of some IoT nodes, hence there is need to evaluate and choose an energy efficient algorithm for routing messages throughout the mesh network.

1.4 Project Objectives

- Develop a Bluetooth mesh simulator of 26 nodes in the network using some of the Bluetooth Mesh network specification, model specification, and the Bluetooth Mesh device properties. In addition, this work will also simulate the managed flooding process of broadcasting messages through the network
- Simulate the Open Shortest Path First (OSPF) protocol on the same Bluetooth Mesh network
- Use the simulations to evaluate flooding, managed flooding and OSPF on a Bluetooth Mesh network.
- Evaluation will be done with respect to Bluetooth Mesh functionality needs such as sending messages to subnetworks, heartbeats and energy used to send each message in a network for flooding, OSPF routing protocol and managed flooding.

1.5 Report Overview

In chapter one, the report introduces Bluetooth Mesh, its flooding algorithm and the motivation to evaluate the managed flooding with respect to other routing algorithms as far as energy consumption and Bluetooth Mesh functionalities. Chapter two explores the work of other scholars that have researched specific topics related to the objectives of this paper and how their work contributes to this paper. Chapter three lays the methodology used for the simulations, and farther explains the implementation of each concept of Bluetooth Mesh. After that, chapter, 4, discusses the results of the simulations and limitations, recommendations and future works are stated in the final chapter, 5. The report also includes the code of the simulation in the appendix.

Chapter 2: Literature Review and Related work

This Chapter explores and discusses the main topics related to Bluetooth and its functionality. Section 2.1 explores the relationship between Bluetooth and the internet of things and also the energy efficiency requirement of IoT nodes. Section 2.2 looks at the versions of Bluetooth and how they have advanced in terms of network topology and scope, energy efficiency and throughput. Lastly, since one of the main objectives of this project is to build a simulator, section 2.4 looks at the research about the available simulators and the basics of setting up a simulator.

2.1 IoT and Bluetooth

The concept of Internet of Things (IoT) has been defined differently by researchers, innovators, and academicians which nullifies the idea of having a standards definition of IoT. However, all the definitions have a core upon which they build. This is the fact that the Internet is no longer a collection of data obtained from people alone but also from physical objects such as refrigerators. This paper uses the definition by Madakam et al., (2015) which defines IoT as;

> "An open and comprehensive network of intelligent objects that have the capacity to auto-organize, share information, data, and resources, reacting and acting in face of situations and changes in the environment"

As the IoT grows and continues to affect most human activities, there is a need for the connectivity technologies of sensor nodes to keep growing at a similar or much higher rate to keep up with the changing requirements of the IoT. "These requirements are total coverage, zero fails (high performance), scalability and sustainability (hardware and software)" (Hortelano, Olivares, Ruiz, Garrido-Hidalgo, & López, 2017). Given the stated requirements, Hortelano et al., (2017)

concluded that Bluetooth Low Energy was the ideal technology for the required improvements. They also recommended that Bluetooth Mesh, which was still under development at the time, would be a better solution and complement to the IoT. Bluetooth Mesh allows devices of lower Bluetooth versions to share the same network and privileges as the devices with the latest version of Bluetooth through its proxy feature. The proxy feature allows Bluetooth Low Energy nodes that have the Generic Attribute Profile interface to interact with nodes that have the Bluetooth Mesh stack in a network (Bluetooth SIG, 2017). Hence, an upgrade in the technology does not affect the connectivity of IoT nodes. Bluetooth Mesh also covers the essential requirement of networks which is security. The Bluetooth Mesh network is secure from nodes that have been removed from the network. Since one can access network information from nodes that were previously in the network, Bluetooth Mesh combats this weakness by assigning new network control data such as the node addresses to all nodes (Bluetooth SIG, 2017). Hence, the information collected from the removed nodes has no links to the network which keeps it secure.

2.1.1 IOT and power usage

The application of IoT has increased in most of the fields which include security, medical, education, and entertainment. In most of these fields, the IoT nodes are left in the field to either relay data to the servers and users or to collect important data that may include temperature, videos, motion alerts and light intensity (Park, Cho, & Lee, 2014). Right from the node that collects the data, through intermediate nodes to the destination where the collected data is needed, a lot of energy is consumed by all nodes that relay this information. This makes the efficient use of energy by nodes an important factor in the use of IoTs since some of them are in remote areas with limited physical supervision (Rani et al., 2015).

The need for efficient use of energy in IoT calls for the need to supply energy constantly to all nodes that relay information or to efficiently relay data to its destination with minimal energy consumption. Considering the latter, two frequently used routing protocols are ad hoc on-demand distance vector (AODV) and dynamic source routing (DSR) which use shortest path algorithms that do not include energy consumption needs in their cost evaluation (Park, Cho, & Lee, 2014). Therefore, continuous use of the same route may workout the nodes along the path used since using them continuously will consume most of their energy as they relay messages. On the other hand, the use of flooding algorithms may guarantee the delivery of the data sent from the field but at the expense of energy since almost every node in the network is actively relaying data. However, Bluetooth Mesh considered this problem and designed a solution called friendship for the low power nodes which is explained in section 1.2.2.

2.1.2 BTIoT-5 and IoT Architecture

IoT has no standard architecture but it has the three main layers on which all other architectures build from. These are perception layer, network layer and application layer (Zhong, Zhu, & Huang, 2017). The perception layer, which is at the base in Figure 2.1, takes over control of the data after its collection, processes and packages it for transmission. In this layer data such as temperature, images and time is collected using sensors, timers, cameras and many others which then process the data for transmission. The network layer, which also serves as the bridge between the perception and the application layer, manages the transfer of the collected data from the sensors to the destination nodes. In the transfer of this data, the network layer uses connectivity technology such as WiFi, satellite communication, Bluetooth and Zigbee. The layer on top, which is the application layer, allows the users to interact with the data collected and also manipulate it. All the data analysis platforms and cloud platforms fall under this layer. (Zhong, Zhu, & Huang, 2017).

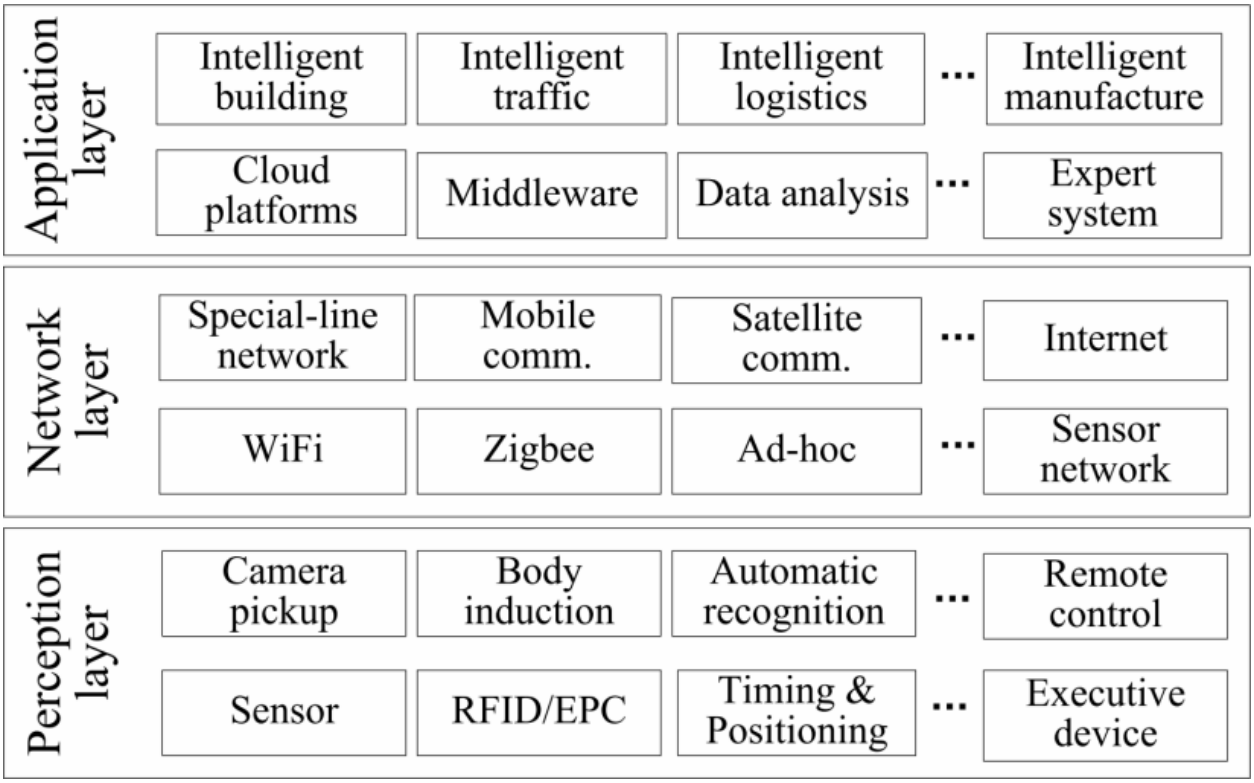

Figure 2.1 Three-layer architecture of IoT

(Source: Zhong, Zhu, & Huang, 2017).

Ray and Agarwal, (2016) developed a more complex architecture that blends both the IoT and Bluetooth 5 layers which is still applicable to Bluetooth mesh because its architecture has Bluetooth Low Energy at the base where other layers sit. Their architecture is made up of 6 layers which are: hardware/layer of Things, Microcontroller layer, the Bluetooth Connectivity layer, IoT Bluetooth Cloud Stack layer and Application layer, see Figure 2.2. In this architecture, Bluetooth plays a major role that is defined in the Bluetooth Connectivity layer. This layer manages all the wireless exchange of data which it does with the two divisions of its stack, the controller, and the host. Lastly, the Bluetooth Host API/Protocol defines all the protocols involved in the transfer of data between IoT nodes and other nodes on the network such as servers and the cloud where data may be stored (Ray & Agarwal, 2016).

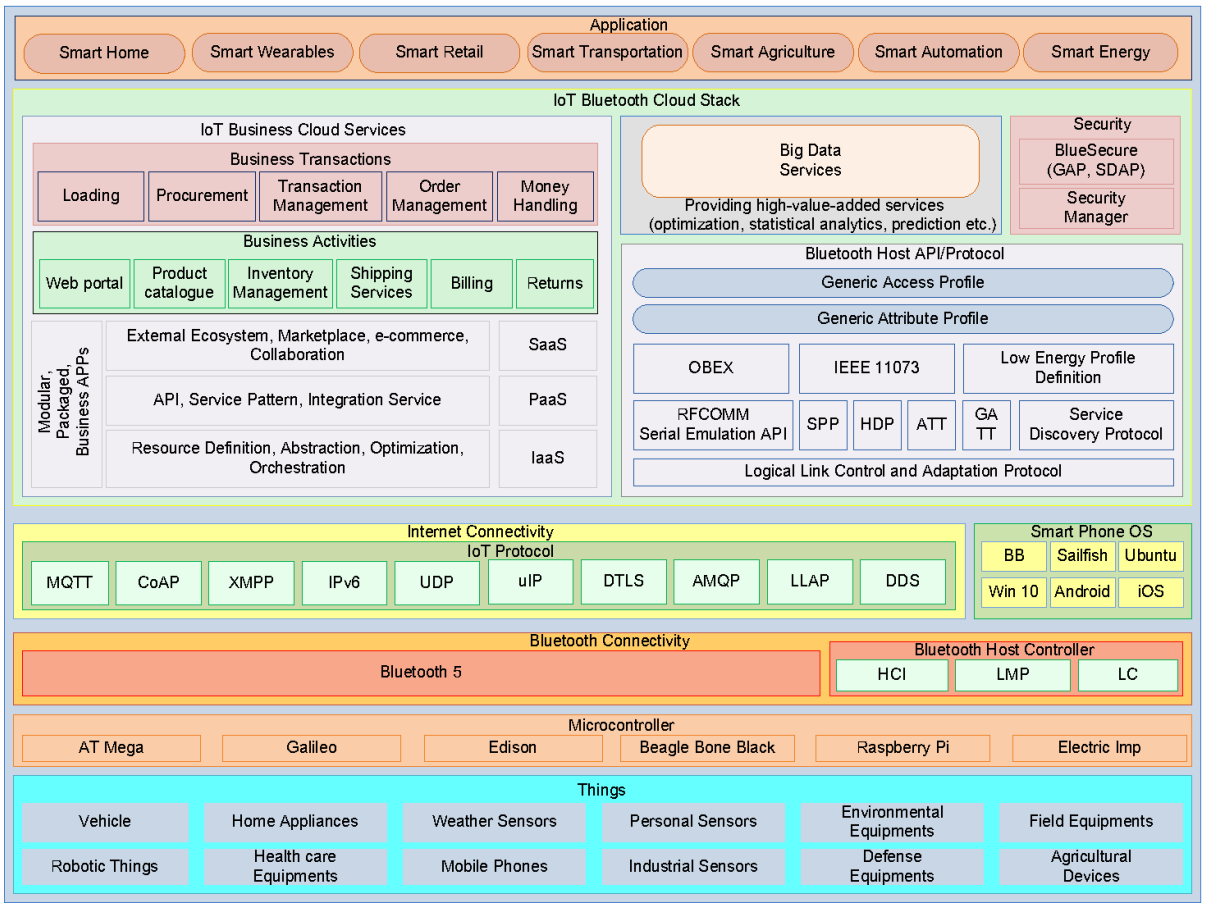

Figure 2.2 Bluetooth-IoT (BTIoT-5) architecture

(Source: Ray & Agarwal, 2016)

2.2 Evolution of Bluetooth

Madhvi Verma et al., (2015) define Bluetooth as "the technology which allows the devices to communicate with each other, synchronize data with each other, and connect to the Internet without the use of cables or wires." Bluetooth has developed to become one of the essential technologies in the wireless transfer of data by increasing its efficiency in term of network coverage range, reliability and speed. Currently, Bluetooth has evolved from the initial Bluetooth Basic Rate to Bluetooth Low Energy which has been revised to create Bluetooth Mesh. The classic Bluetooth / Basic Rate, did not consider nodes with critical power needs although it was used with

low power nodes such as mobile phones and headsets which required constant recharging (DeCuir, 2014).

2.2.1 Bluetooth Low Energy (LE)

Bluetooth LE had its first release in 2010 which is the Bluetooth 4.0, the major aim of advancing to Bluetooth LE was to cater for the IoT energy efficiency demand (Rani et al., 2015). Bluetooth LE achieves this goal by sending data in short bursts allowing it to sleep in intervals depending on the size and number of the bursts. This makes it perform better than the Bluetooth classic which sends streams of data (Guo, Harris, Tsaur, & Chen, 2015). Bluetooth LE started with a star topology which required a single node, the master node, to be connected to all other nodes referred to as slave nodes, and all slaves could only communicate through the masters. This network of slaves and a master is referred to as a piconet with a minimum of 2 nodes and a maximum of 8 devices for version 4.0 and limitless for all subsequent versions. To farther extend the coverage of Bluetooth devices in earlier versions, Bluetooth SIG introduced scatternets which are a network of different piconets connected by either slaves or masters (Guo et al., 2015).

Initially, Bluetooth 4.0 could only send data in a single hop where a message could only be relayed once from the source which was the center node in most cases. In addition, slave nodes could not operate simultaneously in more than one piconet which limited the flow of data across the scatternets. Later versions of Bluetooth have eliminated most of the restrictions allowing slave nodes to communicate freely (Darroudi & Gomez, 2017). The work of Bluetooth SIG also advanced and led to the release of Bluetooth 5 after its predecessor Bluetooth 4.2. Just like the versions suggest, there was a great improvement in the Bluetooth technology in version 5 with an 800% increase in broadcasting capacity, double speed, increased the range by four times and boosted location services (Bluetooth SIG, 2016). However, at this level of efficiency, the star

topology and the routing algorithms used were no longer able to fully utilize the power and capabilities of Bluetooth 5. The suitable algorithms used for such Ad hoc networks are Destination Sequenced Distance Vector (DSDV), Dynamic Source Routing (DSR), Ad hoc On-demand Distance Vector (AODV) (Soni & Khunteta, 2014).

2.2.2 Bluetooth Mesh

Considering the need for full utilization of Bluetooth 5, Bluetooth SIG set aside a team of 80-member companies who worked towards the release of the latest version, Bluetooth Mesh (Bluetooth SIG, 2017). Given that Bluetooth mesh is relatively new in the industry, there has not been much published research describing the major components of Bluetooth Mesh which are, provisioning, managed flooding and security. However, there is published research on the proposed Bluetooth Mesh network solutions which are, academic, standard and proprietary solutions as illustrated in figure 2.3 (Darroudi & Gomez, 2017)

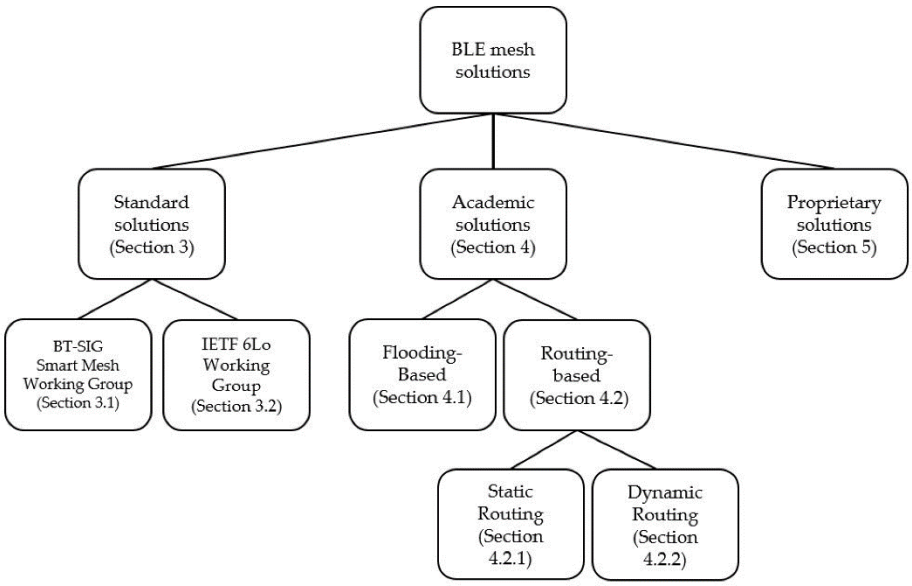

Figure 2.3 A taxonomy of Bluetooth mesh network solutions

(Source: Darroudi & Gomez, 2017)

From Academia there is both the routing-based and the flooding-based solutions. Among the flooding-based algorithms, there is a gossiping-like routing algorithm which is called the trickle algorithm. This algorithm has limited overhead, and it also uses the number of each node's neighboring nodes to determine the re-broadcast probability. As the number of nodes increase, the routing probability reduces which suggests higher packet delay compared to a pure flooding algorithm (Gogic, Mujcic, Ibric, & Suljanovic, 2016). Additionally, Kim, Lee, & Jang, (2015) suggest another flooding algorithm called the BLEmesh which is a bounded flooding mechanism. In this mechanism, the sending node specifies a priority list of intermediate nodes which reduces the number of nodes a packet may have to visit to reach its destination. Most of these suggested algorithms modify the pure flooding algorithm which uses all the nodes in the network and limits the number of relay nodes so that limited energy is used but the common features of flooding such as path diversity are preserved.

On the other hand, the routing solutions are divided into solutions which are the static and the dynamic routing solutions. The main difference between the two forms of routing is that the dynamic routing solution finds the shortest path according to the real-time logical network layout (Darroudi & Gomez, 2017). Both are based on the routing principal of finding the shortest path depending on the distances and traffic in the network (Gogic et al., 2016). These, however, do not take power into consideration. Therefore, if a low power node is in the shortest path, its power may get used up first leading to a dead node in the desired path which means messages cannot be delivered. In dynamic routing, algorithms based on Routing Information Protocol (RIP) and Open Shortest Path First (OSPF) are used. This paper will also implement the OSPF routing protocol which uses the Dijkstra's algorithm to efficiently evaluate the flooding algorithms.

For the standard solutions, the Bluetooth SIG developed the Bluetooth mesh which uses a flooding algorithm called managed flooding. The mesh network is built by a five-step process called provisioning where a device, provisioner, adds other devices on to the network. These processes are; (1) beaconing in which the unprovisioned device reaches out to the provisioner, (2) invitation in which the provisioner then invites the device; (3) then the two devices exchange public keys; (4) authentication of the keys is done and (5) then the provisioner shares the provisioning data with the new device which has just joined the network (Bluetooth SIG, 2017). It is also during this process that the main features of the node are defined. These features are relay nodes which can relay messages to other nodes, proxy nodes which allows Bluetooth LE devices to be added on to the mesh and lastly the low power nodes and friend nodes which enable low power nodes to function efficiently. It is these features that define the role of the nodes on the mesh network which is illustrated in figure 2.4.

In the Figure 2.4, all the nodes that are not specified in the key have no special features. Considering a scenario of a message from node A to K, the message from A will be sent to B, C and D of which only C will relay the message to its neighbors because it is a relay node. In this case, A, B and D will discard the message since it already exists in their cache. The message will then be sent to all neighbors of H which include I, M, N, O and F. F being a friend node to node E, node F will check whether E is the intended destination and then discard the message. At node I, authentication is done to ensure that the destination address of the message matches any of the low power nodes connected to node I. If K is awake and active, the message is delivered instantly, otherwise the message is momentarily buffered at node I and later delivered to K when it is awake and active.

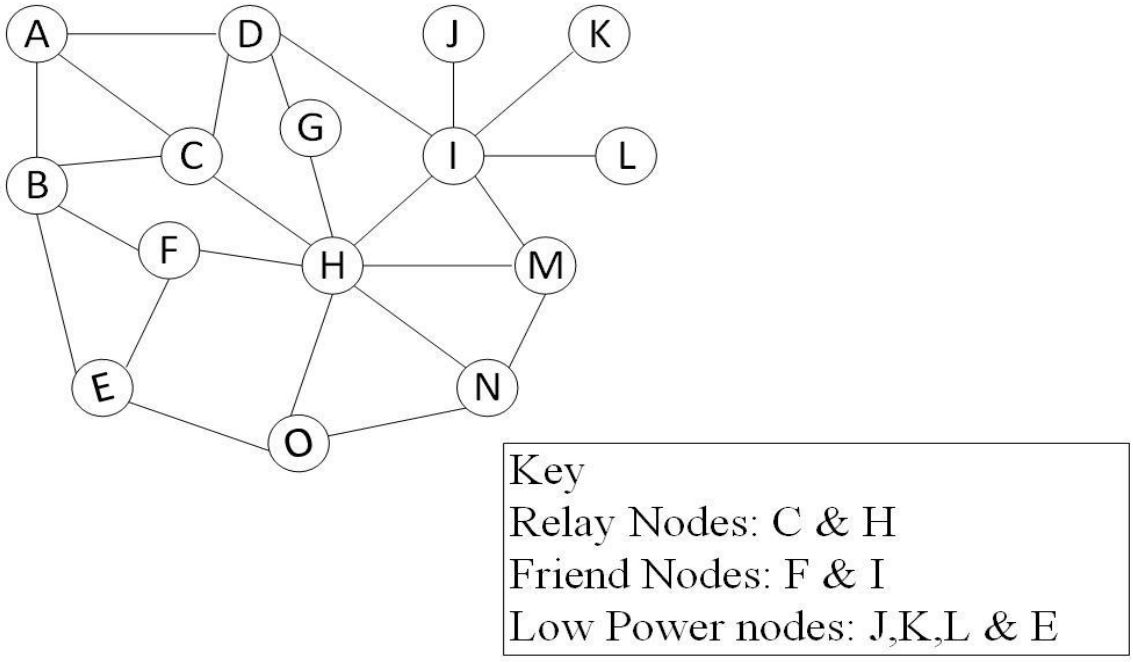

Figure 2.4 Bluetooth Mesh

For a message to be sent using managed flooding, Bluetooth SIG implemented measures that limit the number of relays made in a network and these are; heartbeats, TTL, friendship and message cache (Bluetooth SIG, 2017). The heartbeat messages are sent by each node in the network periodically to inform other nodes in the network that it is still available. Failure to send these messages indicates the node is no longer active which implies no message will be sent to or through it. The time-to-live measure indicates the maximum number of hops each message to make in the network, this helps to eliminate the sending of a single message infinitely if its destination is not in the network (Bluetooth SIG, 2017). The message cache keeps track of all messages that have been sent to a node so that when that same message is resent to it, the message is just discarded instead of resending to other nodes. Lastly is the friendship measure which allows friend nodes to receive messages meant for low power nodes and deliver them when the low power nodes are awake. Friend nodes must have a constant power supply of durable batteries to keep them awake always (Bluetooth SIG, 2017).

2.2.2.1 Bluetooth Mesh Architecture

Managed flooding is also controlled by the bearer, network and transport layers of the Bluetooth mesh architecture which examine the address of the given node to determine whether it is the defined destination of the message, processes the messages received and enforces the roles of the nodes as defined by the 3 features explained earlier in section 1.2.1 (Bluetooth SIG, 2017). The Bluetooth mesh Architecture is made up of 7 layers as illustrated in figure 2.5. The bottom-most layer is the Bluetooth LE stack which provides the fundamental wireless communications capabilities (Bluetooth SIG, 2017). The rest of the layers sit on the Bluetooth LE stack which are a pile of layers on its own.

When the message arrives at a node, the protocol data units (PDUs) have information on what the receiving node should do. The message received is sent up the stack to the network layer which verifies the NetKey (network key) to determine if the message belongs to the current node's sub-network (Bluetooth SIG, 2017). If it does not, then it is discarded or forwarded to other nodes if the current node has a relay feature. The network layer also does the network message integrity check (MIC) and if it fails, then it is discarded. If the message is meant for the receiving node, it continues up the stack to the Upper transport layer where the AppKey is verified and the transport message integrity check (TransMIC) is done to decide if the message should either be relayed, discarded or continue to the upper layers (Bluetooth SIG, 2017). The entire process of traversing the stack upwards is to ensure that the message gets to the right destination

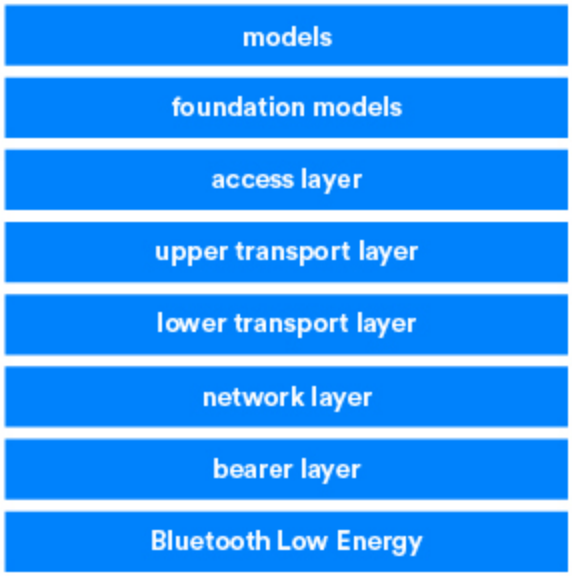

Figure 2.5 Bluetooth Mesh architecture

(Source: Bluetooth SIG, 2017)

2.3 Network Simulation

One of the major objectives of this work involves simulation of the Bluetooth Mesh which will then be used for the evaluation process. Before simulation starts, there is need to build a model which represents the aspects of the actual physical entity that is to be simulated. In computer networks, simulation enables the creation, development, and testing of new protocols in an isolated, yet flexible realistic environment before deploying them in the field which protects the physical network from any malfunctions that could have been caused by the new protocols (Khana, Bilalb, & Othmana, 2013). There are many network simulators. This suggests an important task for researchers during simulation which is selecting the best simulator for the network model to be simulated. All the different simulators have different strengths; therefore, it is important that a researcher selects one whose strengths match the most vital network aspects to be simulated (Khana et al., 2013). The most commonly used simulators are, Qualnet, Omnet++, Network

Simulator 2 (NS-2), Network Simulator 3 (NS-3) and Optimised Network Engineering Tool (OPNet) (Zarrad & Alsmadi, 2017).

NS-2 is an event-driven network simulator that was developed specifically for the simulation of small networks, that is, less than 500 nodes. It allows the simulation of very complex models and it has an energy efficiency model that provides an easy pattern of traffic and movement (Zarrad & Alsmadi, 2017). However, it has poor GUI and its documentation is outdated (Mohammed Humayun Kabir et al., 2014). NS-3 provides real network environments and has characteristics that allow the incorporation of software-defined networks which is a new area of interest for most researchers. It is similar to NS-2 but with more improved characteristics to cater for big network simulation (Zarrad & Alsmadi, 2017).

Omnet++ is a component-based open source simulator that uses C++ to create the components. The programmed components can then be compiled together to create a bigger component of a model giving it the ability to support large-scale networks with the help of high-level languages (Mohammed Humayun Kabir et al., 2014). Its GUI is advanced with intelligence support and it has a vast library that is made up of the simulation kernel and utility classes. It is this Library that enables the creation and assembling of components during simulation. On the other hand, the throughput of Omnet++ reduces as the number of active nodes in the simulation increase and its computation time is directly proportional to the size of the network (Zarrad & Alsmadi, 2017).

OPNet is a commercial discrete event network simulator which works with all wired and wireless networks. Despite its user-friendly GUI and a vast complete model library, it is not commonly used in academic research because of the high costs involved in using it. Hence, it is

mainly used in industrial research and network development. It is one of the most powerful simulators because of its peculiar features which include; grid computing support, customizable wireless modeling, Integrated, GUI-based debugging and analysis and the Hierarchical modeling environment (Kabir et al., 2014).

Among the listed simulators above, NS-2 and Omnet+ are the most popular ones in academia despite their limitations such as the complicated architecture for NS-2. Omnet++ stands out of the two because of advanced GUI, flexible model structure and the well- designed simulation engine (Zarrad & Alsmadi, 2017). However, Bluetooth Mesh is a new technology whose features are not supported by these simulators. Therefore, this project will build its own simulator in java programing language using the basics of simulation as defined by White & Ingalls, (2017). These are inputs, outputs, and state, entities and attributes, resources, activities and events, and statistics collectors.

2.4 Literature Review Summary

The literature review section has explored the communication technologies of IoT, its energy efficiency needs and the evolution of Bluetooth from Bluetooth Basic Rate, to Low Energy and finally to Bluetooth Mesh. Since there is limited research published about Bluetooth Mesh, this work used the Bluetooth Mesh specifications document. This chapter also discusses the most popular simulation tools for both academic research and industrial research.

Chapter 3: Methodology

This chapter explores the methodology used to implement and evaluate the managed flooding algorithm for Bluetooth Mesh. This project uses simulation because it is expensive to buy the physical nodes required and it also consumes physical storage space. Therefore, simulation will be the best option since the entire Bluetooth Mesh can be modeled and built virtually saving the physical space and monetary costs. Since Bluetooth Mesh is new, the available simulators are not able to efficiently simulate its behavior, therefore, a simulator was built for this project and the main features of managed flooding evaluated. To evaluate the energy efficiency of managed flooding, a routing algorithm that uses the shortest path is simulated and its energy efficiency also evaluated in comparison to the results of the managed flooding and pure flooding.

3.1 Basics of Simulation

This project will develop a simulator using some of the basics of simulation and use it to set up a simulator for the Bluetooth Mesh. These are inputs, outputs, and state, entities and attributes, resources, activities and events, and statistics collectors (White & Ingalls, 2017). The simulation starts with the input values. The input of the simulation is the configuration file that defines the topology, the number edges and nodes connected to each node and whether the node relays messages or not. After getting the data, the simulator chooses one of the three states of the network to be simulated and these are flooding, OSPF and managed flooding. In each state, the main activity is sending the message whose path is determined by the algorithm used by each of the three states. The nature of resources shared in the simulation changes depending on the state being used, however, the main resources are, the paths used and Time To Live(TTL) which is also known as the hop count. With the OSPF protocol, there is only one definite path from source to destination while the flooding protocols have a variety of the path resources available for them

which is one of the main advantages of flooding. Finally, at the end of the simulation, there is need to collect data which is then used for evaluating each state (White & Ingalls, 2017). The statistics collected here include the total power used in sending one message, the total number of hops, number of nodes in the network and the number of edges in the network. Below is the flow chart of the entire simulation process.

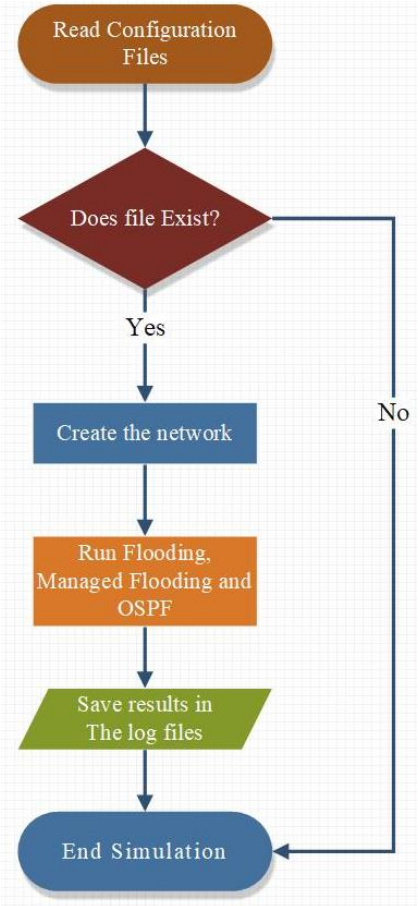

Figure 3.1 The flow of the simulation

3.2 Routing and Broadcasting

The simulation uses both broadcasting and routing to convey messages from the sending node to the receiving node. In the former, the sending node does not have any information on the right path to take. However, a message is broadcasted to all nodes attached to the sending node, the adjacent nodes also forwards the messages to their neighbors until the message finally gets to its destination. This will be used by both pure flooding and the managed flooding algorithms. The OSPF protocol uses the Dijkstra's algorithm in which every node has information on how to get to all the other nodes in the network. Hence, the message follows a defined path from source to destination. The message broadcasted or routed has the information about its destination, source, frequency and the payload to enable the intermediate nodes to relay it to its right destination.

3.3 Project structure

The simulation is defined by three main stages which are, reading network data from configuration files, creating the network and finally sending the message using the chosen protocols and results saved in log files. This is achieved by using six Java classes and one main class that runs them. These classes are illustrated in figure 3.2.

The device class lays the attributes and actions of each node in the network. The main attributes of the device used by the OSPF protocol are the distance which is the cost to that node and the shortest path which is a list of nodes through which you can reach the node at the least cost. The flooding protocols have the relay Boolean feature added to the nodes which when true allows the node to forward messages received to all nodes connected to it. The path attribute defines the path taken to reach the desired node. In addition to the above attributes, all nodes have an ID attribute that distinguishes them from others, a list that keeps track of the immediate neighbors, another list that keeps track of the messages that have been relayed by the node so that

they get discarded in case resent to the same node. Lastly, the text attribute captures the payload in the message sent to the node and sent to other layers for interpretation. All these variables are private and can be set and retrieved using the constructor, mutator and accessor methods.

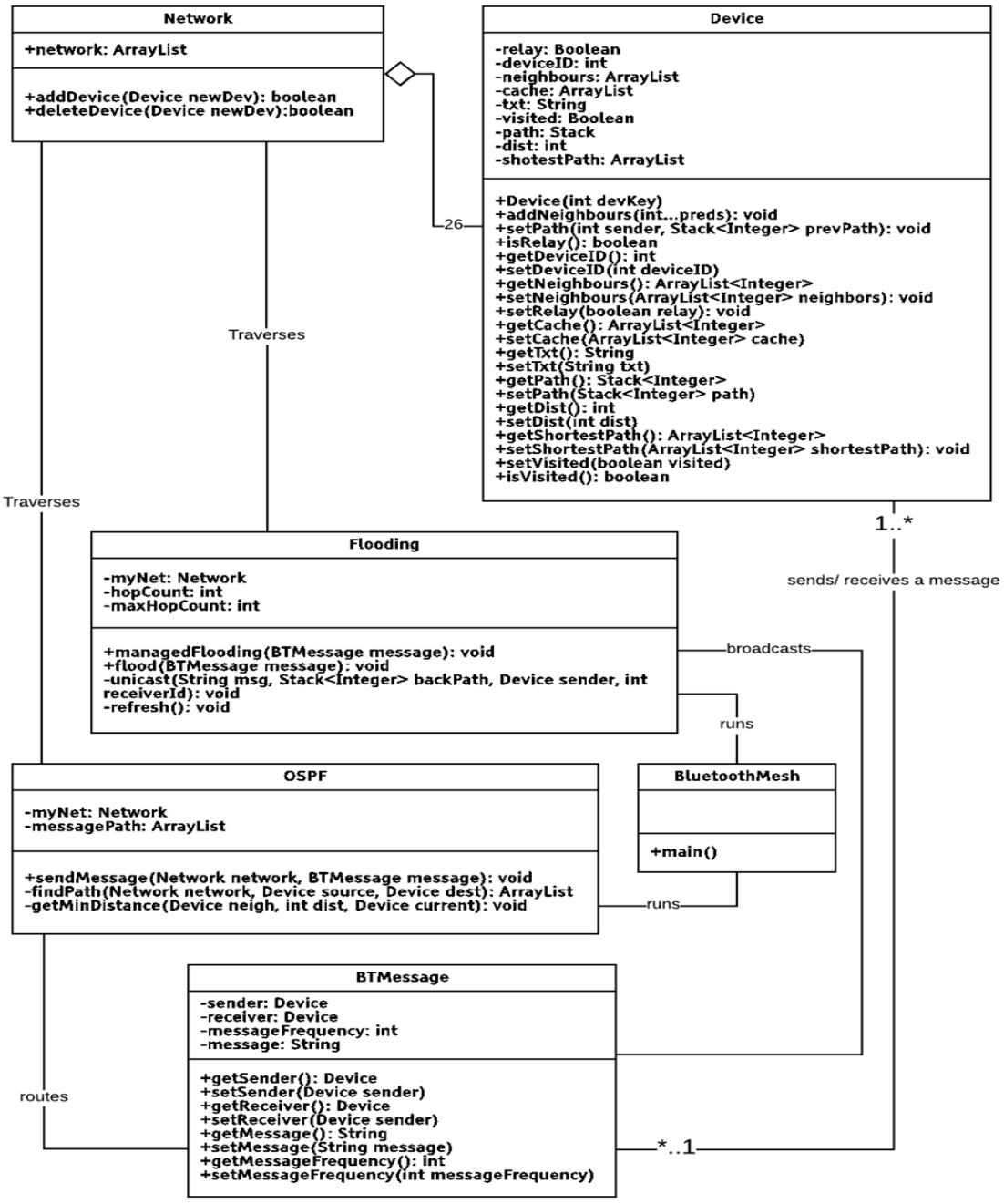

Figure 3.2 UML Class Diagrams

The message class defines the composition of the message which makes up each PDU. The main content of the message is the payload which is the main information that is transmitted and the rest is added information required for routing. The added information includes the source address, destination address, and the frequency for identifying the message uniquely. The network class gets information from the configuration files, creates the network and keeps track of the connections in the network to ensure the specifications from the configuration file are maintained.

The flooding class implements both the pure flooding and the managed flooding algorithms and operates depending on the function called. It also has a unicast function that keeps track of the path that is finally used to send the message and uses it to send acknowledgments along the same path to limit flooding. The main attributes include the network which the two algorithms traverse to send the message, hop count and maximum hop count (TTL). The OSPF protocol is implemented in the OSPF class which also has the network as an attribute and a list of nodes which make up the desired path for the message. It is made up of 3 functions which describe the underlying OSPF routing algorithm, the Dijkstra's algorithm.

3.4 Implementing flooding

Bluetooth Mesh uses managed flooding as a means of sending messages from one node to another in the network. It was adopted from the pure flooding algorithms (illustrated in Figure 3.3) from which a few limitations where introduced to improve its energy efficiency. In this project, a pure flooding algorithm will be implemented and 3 features of managed flooding nodes will be introduced. Flooding is implemented using the breadth-first search (BFS) algorithm which uses a queue to keep track of the nodes visited. The BFS algorithm ensures that a message is sent to all immediate neighbors of the source node before it proceeds to other levels. At each node, the algorithm also keeps track of the nodes visited from the source to destination in order to establish

a single path that would be used to send an acknowledgment to the source node. In order to improve the energy efficiency of flooding, managed flooding has some characteristics that reduce the number of nodes broadcasting the message. These include friendship, time to live, message cache and the relay nodes. This project will add the last three characterizes to the pure flooding algorithms to reduce the total number of messages sent which in the end will reduce the energy used in delivering a single message. The diagram below illustrates flooding with thick arrows showing the direction the message is being sent to, doted arrows represent discarded messages that are sent to nodes that have received it, and the S to Z (line with a dot) arrow points to the destination node.

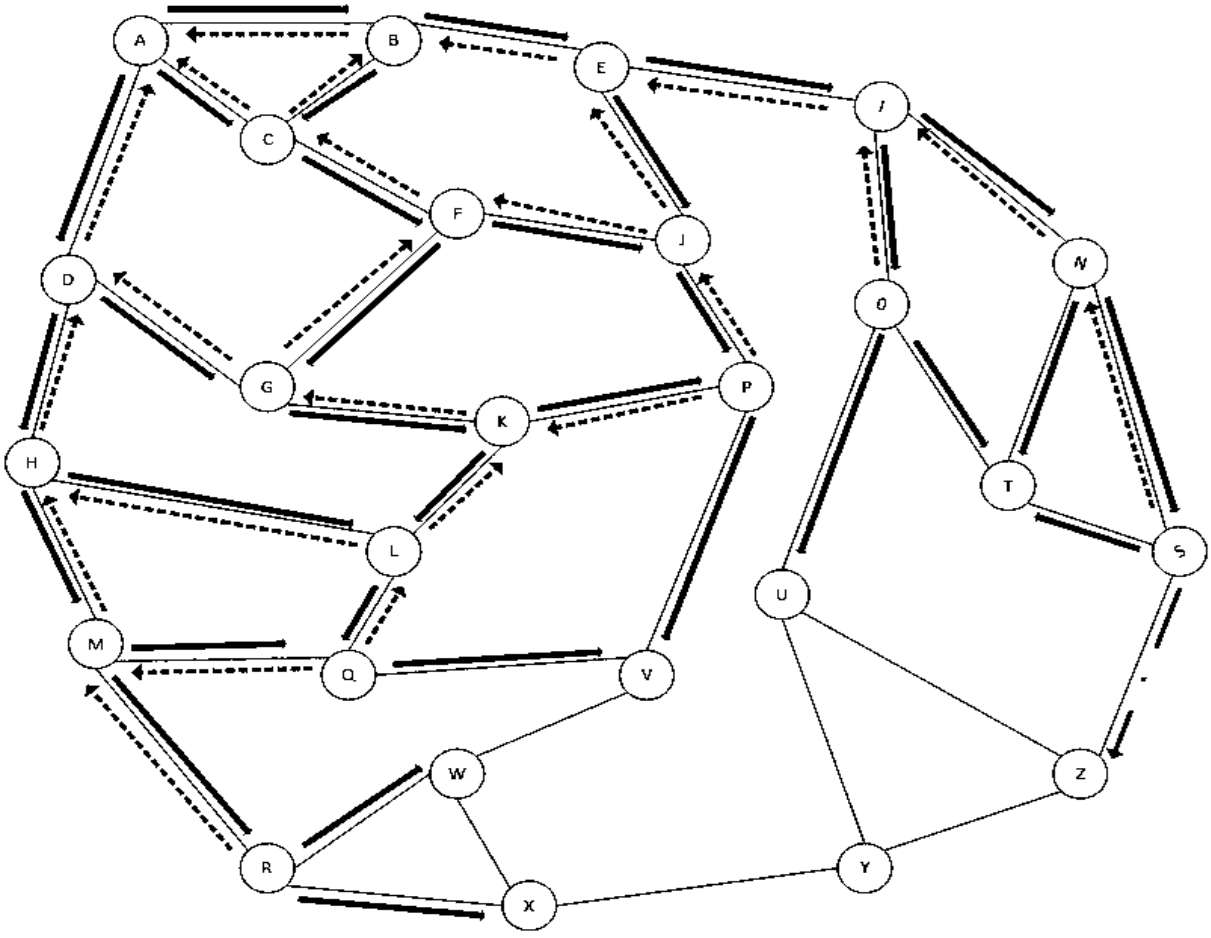

Figure 3.3 How Flooding works (sending from node A to node Z)

3.5 Implementing Open Shortest Path First (OSPF)

OSPF is a routing protocol that finds the shortest path from the source to destination and uses it to deliver the message. It may be energy efficient in terms of the number of nodes used but if a single path is used continuously, some nodes may die out which will block the path in cases of fixed routing. When the algorithm is started, a method is called to find the path to the destination. This message goes through all the nodes in the network setting the minimum distance of that node from the source node. After traversing the enter network of nodes, a list of nodes along the shortest path from the source to the destination is returned and used to send the message. Dijkstra's shortest path algorithm is used to implement the OSPF protocol.

3.6 Network Setup

For proper evaluation of the flooding algorithm, three networks of 26 nodes were developed and simulation was done with varying number of edges connected to each node. This model was chosen because the number of edges per node determines the possible routes that can be accessed from a node by the flooding algorithm. For the case of two edges on each intermediate node and one edge on each of the source and the destination nodes, this case results in having a single path even when using flooding. This set up is can also be referred to as the linear layout. Hence, the decision to start with 3 edges per node. As the number of edges per node increases, the number of paths increases, as well as the energy used to send one message from source to destination. Since relay nodes are required in the managed flooding algorithm, the fourth column in the configuration file specifies the relay nodes in the network using 0 for non-relay nodes and 1 for relay nodes. These are mainly nodes that improve the accessibility to other nodes.

The simulation started with 26 nodes and 3 edges for each node as shown in the figure 3.4 and table 1 below. The simulation ran for all the 3 routing protocols, pure flooding, managed

flooding and OSPF. The message was sent from node 0 (node A in figure 3.4) to node 25 (node Z figure 3.4).

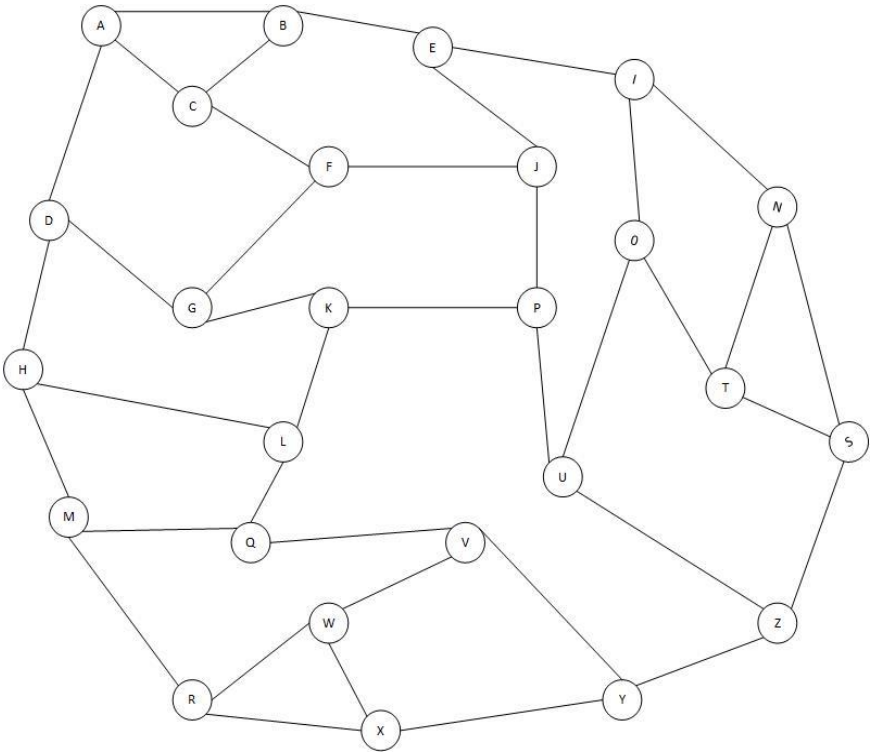

Figure 3.4 Mesh network with 26 nodes and 3 edges each

Name	ID	Neighbors	Relay
A	0	1,2,3	
B	1	0,2,4	
C	2	0,1,5	√
D	3	0,6,7	
E	4	1,8,9	
F	5	2,6,9	√
G	6	3,5,10	√
H	7	3,11,12	
I	8	4,13,14	
J	9	4,5,15	√
K	10	6,11,15	√
L	11	7,10,16	√
M	12	7,16,17	
N	13	8,18,19	
O	14	8,19,20	√

P	15	9,10,21	√
Q	16	10,11,21	√
R	17	12,22,23	
S	18	13,19,25	
T	19	13,14,18	√
U	20	14,24,25	√
V	21	15,16,22	√
W	22	17,21,23	√
X	23	17,22,24	
Y	24	20,23,24	
Z	25	18,20,24	

Table 1 Mesh network with 3 edges per node

The same process was repeated with the sane number of nodes but increasing the number of edges to 4 per nodes in the network. Both Figure 3.6 and Table 2 illustrate the setup of this network. Similarly, the message was sent from node 0 (node A in figure 3.5) to node 25 (node Z figure 3.5) and all the 3 protocols are considered.

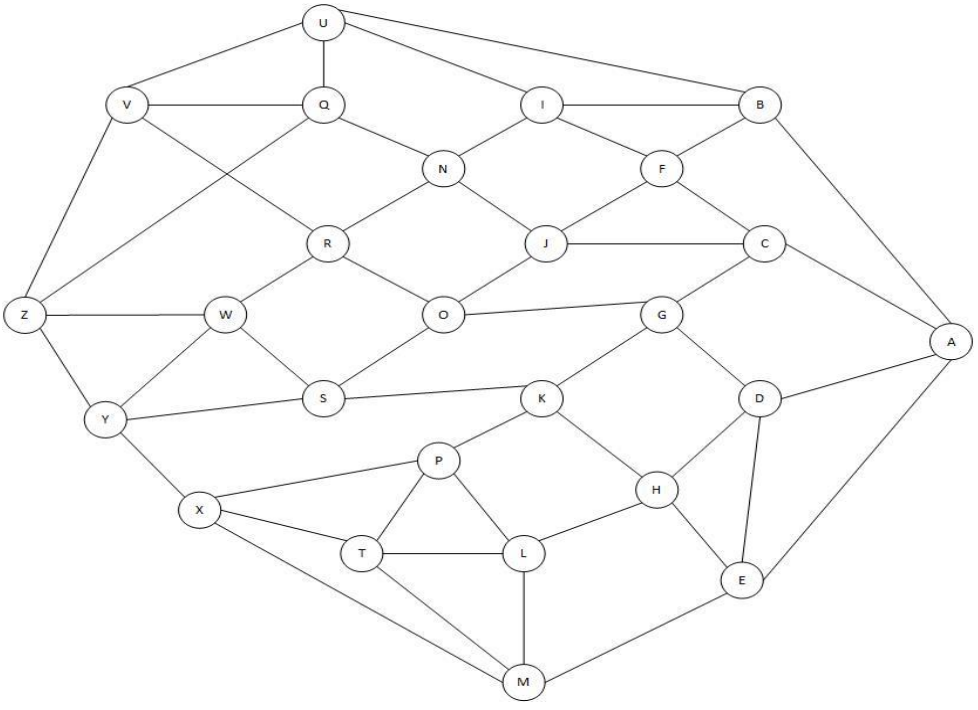

Figure 3.5 Mesh network with 26 nodes and 4 edges each

Name	ID	Neighbors	Relay
A	0	1,2,3,5	
B	1	0,5,8,20	√
C	2	0,5,6,9	√
D	3	0,4,6,7	
E	4	0,3,7,12	√
F	5	1,2,8,9	
G	6	2,3,10,14	√
H	7	3,4,10,11	
I	8	1,5,13,20	
J	9	2,5,13,14	√
K	10	6,7,15,18	√
L	11	7,12,15,19	
M	12	4,11,19,23	√
N	13	8,9,16,17	
O	14	6,9,17,18	√
P	15	10,11,19,23	
Q	16	13,20,21,25	√
R	17	13,14,21,22	
S	18	10,14,22,24	√
T	19	11,12,15,23	
U	20	1,8,16,21	√
V	21	16,17,20,25	
W	22	17,18,24,26	√
X	23	11,12,15,24	
Y	24	18,22,23,25	
Z	25	16,21,22,24	√

Table 2 Mesh network with 4 edges per node

Lastly, the setup of a denser graph of 26 nodes and 5 edges per node which increases the number of paths available for any desired source to destination route. The network is illustrated in figure 3.7 and table 3. The message flows from node 0 (node A in figure 3.6) to node 25 (node Z figure 3.6) and all the 3 protocols are considered.

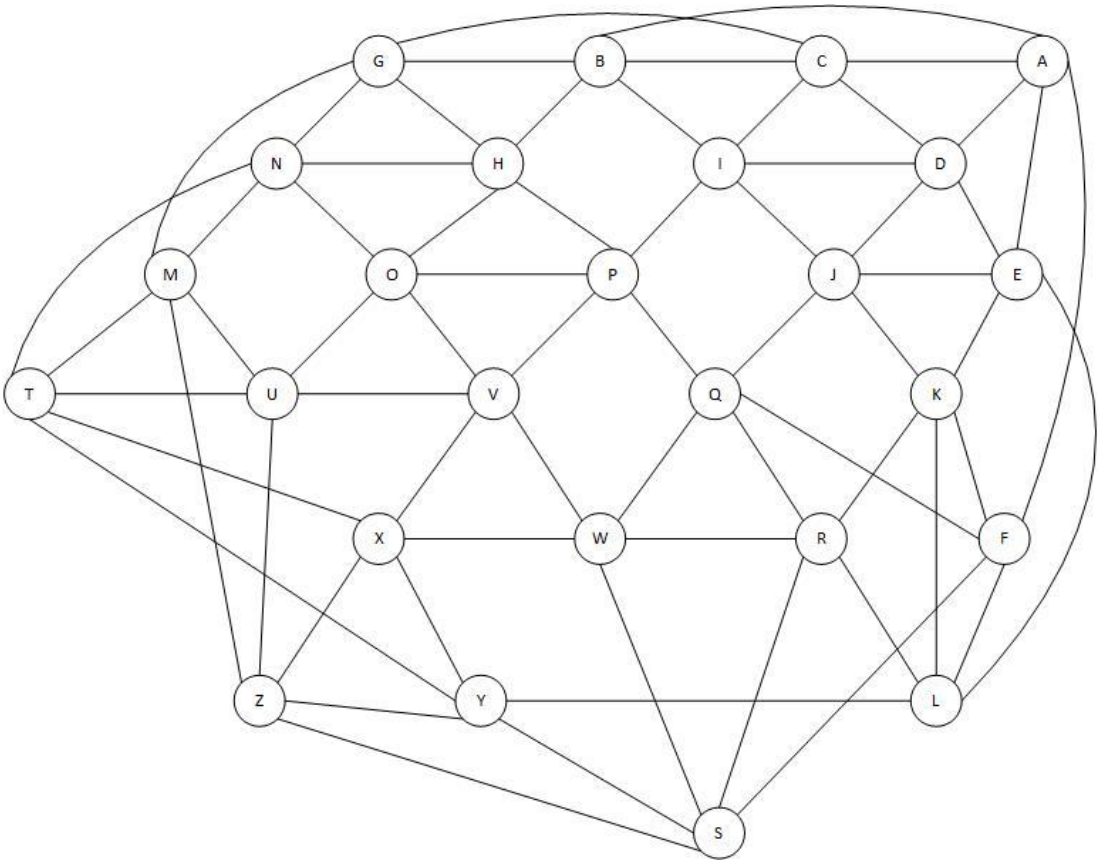

Figure 3.6 Mesh network with 26 nodes and 5 edges each

Name	ID	Neighbors	Relay
A	0	1,2,3,4,5	√
B	1	0,2,6,7,8	√
C	2	0,1,3,6,8	
D	3	0,2,4,8,9	√
E	4	0,3,9,10,11	
F	5	0,10,11,16,18	√
G	6	1,2,7,12,13	√
H	7	1,6,13,15,16	
I	8	1,2,3,9,15	
J	9	3,4,8,10,16	
K	10	4,5,9,11,17	
L	11	4,5,10,17,24	√
M	12	6,13,19,20,25	√
N	13	6,7,12,14,19	√
O	14	7,13,15,20,21	√
P	15	7,8,14,16,21	
Q	16	5,9,15,17,22	√

R	17	10,11,16,18,22	√
S	18	5,17,22,24,25	√
T	19	12,13,20,23,24	√
U	20	12,14,19,21,25	√
V	21	14,15,20,22,23	
W	22	16,17,18,21,23	√
X	23	19,21,22,24,25	
Y	24	11,18,19,23,25	
Z	25	12,18,20,23,24	√

Table 3 Mesh network with 5 edges per node

3.8 Data Collected

As part of the basics of simulation, data collection is very important in the process of evaluating the simulations made. In this simulation, two forms of data were collected. First, the total hop count of sending a single message from source to destination. For flooding algorithms, it comprises every single message sent even if it gets discarded. Second, data on the actual number of hops required to deliver a message. This includes all hops on nodes along the path the delivered message took from the source to destination. This information is collected for all the three protocols being analyzed and used to generate the energy used sending the message. Knowing the hop rate and the energy transmission rate provided room to use the generated hop count to produce the energy used in mA (milliamps). According to Mostafa & Islam, (2016), Bluetooth has a hop rate of 1600 hops per second, hence, 1 hop takes 0.000625 seconds. Nilsson & Lindman, (2017) also stated that sending Bluetooth messages consumes 52 mAh every 3.8 seconds, hence, in one second, 13.68mAh are consumed while sending. This value was abstained by applying a voltage of 3.3V to the nodes. Narrowing it down to just one hop which takes 0.000625 seconds, each hop, therefore, consumes 0.0086mAh. The simulator uses the energy consumed per hop to then generate the total energy consumed by the total hop count and the hop count along the delivery path. These results

are then used to measure and analyze the energy consumption efficiency of each protocol in the simulation.

3.9 Assumptions

In the process of developing models, some assumptions are made in order to represent the actual complex system. In this simulation, 3 main assumptions are made. These include a unit cost for all edges in the network, no low power nodes in the system and successful delivery of all messages. The main concern being the number of messages sent out, the simulation assumes all edges are homogeneous and have a unit cost. In addition, all nodes are static therefore no node changes location to alter the cost involves in reaching it

The simulation also assumes that there are no low power nodes in the simulation of managed flooding. Given the four main attributes of managed flooding stated in section 3.4 above, the simulation only considers leaving out friendship because, in a friendship pair, all the work of sending and receiving messages is done by the friend node. Therefore the simulation also assumes that in case of any low power node in the network, it should be always awake to receive messages on its own.

Lasts, the simulation assumes that all messages sent from the source node to the destination node are successfully delivered without any interference. This means all messages arrive correctly, in their right order and shape without any duplications and none of them is expected to get lost during transmission. In addition, all nodes are assumed to be on and fully functioning at all times to receive and send messages and none of them gets lost in the network. Therefore the simulator does not do message error control in the process of transmitting the message.

3.10 Methodology Summary

This chapter discusses the basics involved in the process of developing the simulator and the setup of the simulation process. The chapter farther describes the setup of all the three networks and how the three protocols are used to transfer messages from one point of the networks to another. The data collected from the simulation is used in the next chapter to analyze and discuss the efficiency of each of the protocols in terms of energy consumption and Bluetooth Mesh functionalities.

Chapter 4: Results and Analysis

The results analyzed in this chapter are collected after running the simulation three times on the three routing protocols being used to evaluate managed flooding which is used by Bluetooth Mesh. At each simulation, the number of edges in the network is varied in an increasing manner from 3 to 5 which changes the number of routes and number of hops involved in accessing the destination node.

The main results collected and to be analyzed are the total number of hops, number of hops along the message delivery path and the energy used in delivering the message to its destination. This information then informs the recommendations and choice of protocol that is feasible and more valuable to the Bluetooth Mesh.

4.1 Results

4.1.1 Flooding

Considering flooding in the first simulation of the 26 node network with 3 edges on each node. Running the simulation for the pure flooding algorithm yielded a total of 89 hops at a cost of 0.7654mAh. However, the message path only had 7 hops at an energy cost of 0.0602 mAh which is 7.9% of the total number of hop and energy used.

In the second simulation of 26 nodes in the network and 4 edges on each node, the same algorithm had a total of 83 hops at a cost of 0.7138mAh. The path along which the message was delivered had 5 hops at an energy cost of 0.043mAh which yields a 6% of the total number of hops and energy used

The last simulation increased the number of edges to 5 keeping the number of nodes constant and produced 114 hops in total with 0.9804mAh. However, the delivery path had only 4

hops and used 0.0344mAh to deliver the message to the destination. This results in 3.5% of the total number of hops and energy used.

4.1.2 Managed flooding

The second algorithm tested is an optimization of flooding, referred to as managed flooding. In the first simulation with 3 edges on each node in a network of 26 nodes, managed flooding yielded a total of 47 hops with 7 hops on the delivery path and a total of 0.4042mAh. The message delivery path had an energy cost of 0.0602 mAh which is 14.9% of the total number of hops and energy used

In the second simulation of a 26 node network with 4 edges on each node, a total hop count of 27 hops at an energy cost of 0.2322mAh. The number of hops on the path along which the message was delivered was 5 with an energy cost of 0.043mAh. This is just 18.5% of the total number of hops and energy used.

In the last simulation, the number of edges was increased to 5 per node in the 26 node network the total hop count was 39 hops at an energy cost of 0.3354mAh. Of the 39 hops, only 4 hops were used to deliver the message to the destination at a cost of 0.0344mAh. Hence, only 10.2% of the hop count and energy was used along the actual message delivery path.

4.1.3 OSPF

This protocol uses Dijkstra's algorithm with only considers the shortest path and sends the message to the destination along that path. Therefore, the total hop count and energy used in this protocol is equivalent to the energy and hop count used along the delivery path. In the first simulation, OSPF used 7 hops and 0.0602mAh, followed by 5 hops and 0.043mAh in the second simulation and finally 4 hops and 0.0344mAh.

Table of results

Hop count of the algorithms						
Number of Edges	**OSPF**		**Flooding**		**Managed Flooding**	
	Energy used (mAh)	**hops**	**Energy used (mAh)**	**hops**	**Energy used (mAh)**	**hops**
3	0.0602	7	0.7654	89	0.4042	47
4	0.043	5	0.7138	83	0.2322	27
5	0.0344	4	0.9804	114	0.3354	39

Table 4 Simulation results

Graphical representation of the number of hops for each protocol

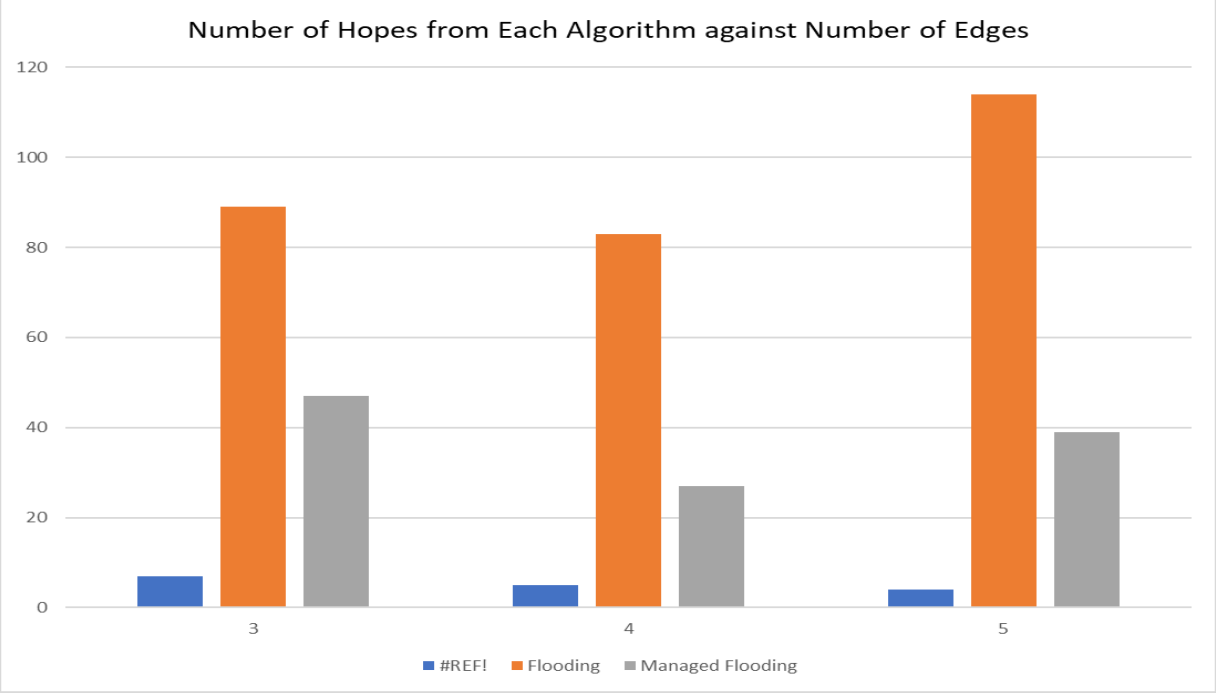

Figure 4.1 Hop counts of each protocol

Energy used by each protocol

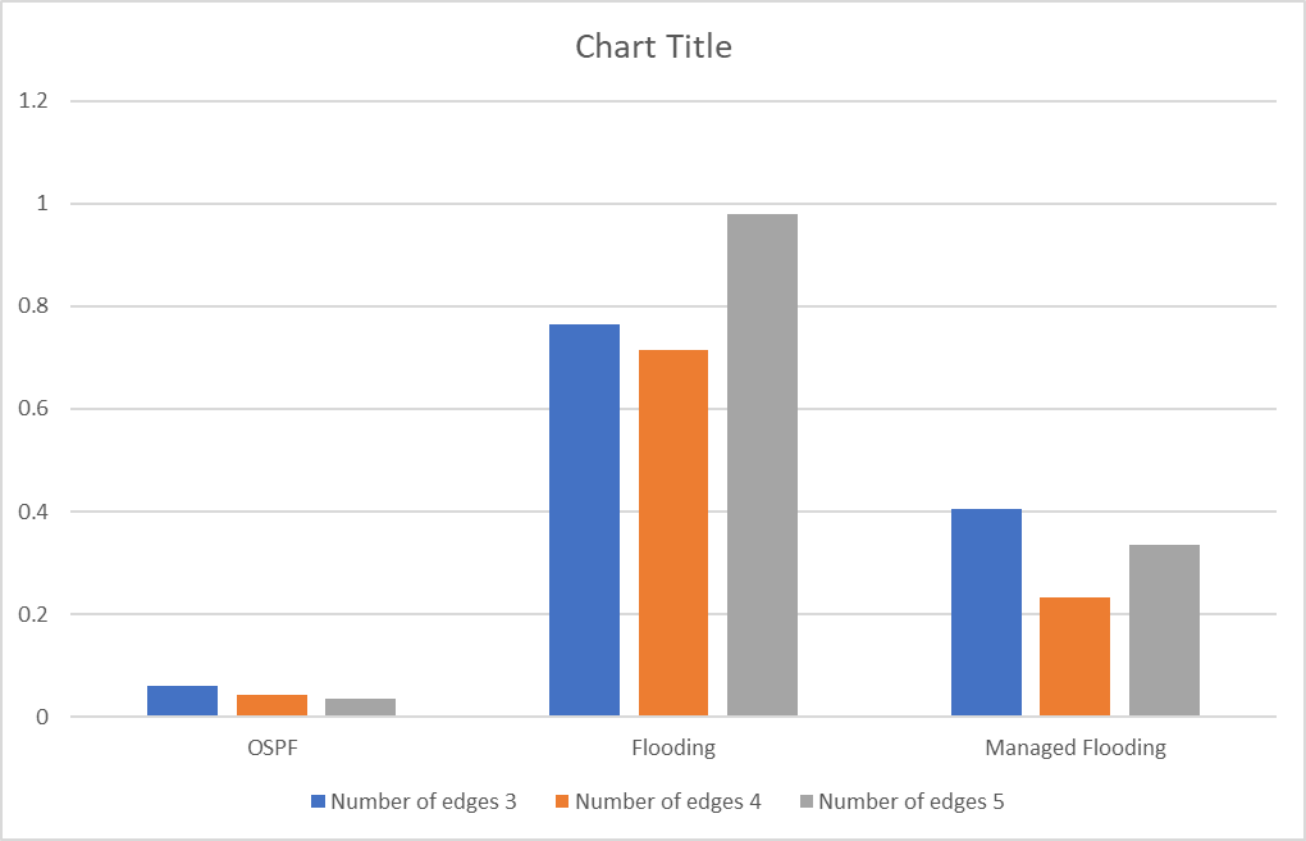

Figure 4.2 Illustration of energy consumption for each protocol

4.2 Discussion of Results

4.2.1 Energy consumption

One of the main objectives of this project is to evaluate the energy efficiency of each of the 3 protocols during the processes of message transmission. In flooding, as the number of edges increases, the total number of hops increases because all edges in the network are utilized. However, on delivering the message to the destination, the message delivered takes the shortest path and discards the rest of the messages in the network. Therefore, only a small portion of the

total energy used, that is, 7.9%, 6% and 3, respectively for the 3 simulations goes into delivering the message. This makes this algorithm inefficient in terms of energy used in delivering messages.

Managed flooding which is an improved version of flooding reduces the number of hops made which reduces the amount of energy used compared to the pure flooding. In this protocol, the number of hops and energy consumed vary arbitrarily with the number of edges per node because it is the number of relay nodes that determines the reduction and increase in the number of hops. The improved algorithm however shows an improvement in the energy efficiency with 14.9% in the first simulation, followed by 18.5% and 10.2% in the last simulation

OSPF is the most efficient of all because the total number of hops used is the same as the hop count along the delivery path. Therefore, if other factors are constant, the network used 100% of the total hop count to deliver the message. However, the good performance of OSPF is also because the cost of routing has been excluded yet flooding factors in both routing and forwarding. The path used by OSPF is also equivalent to the delivery path of both flooding and managed flooding which means the two algorithms will always find the shortest path but at a higher energy consumption test.

Lastly, OSPF will always choose the shortest path to the destination no matter the type of nodes along that path. If we consider the last simulation, the message took the following path, 0 - 5 – 18 – 25. If a message is to be sent continuously from Node 0 to 25, it will use the same path at each message delivery. If node 5 is as low power node that transmits messages to node 25 for 2 minutes (120 seconds) per hour;

In each second, node 5 sends 1600 hopes at a cost of (0.0086*1600)mAh

In 120 seconds node 5 consumes (0.0086*1600*120)mAh

Therefore, in a day node five consumes 0.0086*1600*120*24 = 39.63Ah

Which many low power nodes which use coin cells of about 240 mAh or less may not afford.

Hence, managed flooding can take care of nodes such as 5 by not granting them the privilege to relay messages. This saves them from using their energy in transmitting messages and use it to receive and read messages sent to them. Managed flooding also allocates a friend node to such nodes to help them participate in the network while they are sleeping. Therefore, managed flooding becomes more energy efficient when it comes to catering for low power nodes.

4.2.2 Bluetooth Mesh Functionalities

Although OSPF gives great results in terms of energy efficiency, there are other functionalities and needs of Bluetooth Mesh required for it to operate within a long range and to also serve as the backbone communication technology of IoT. One of these is the periodic heartbeat messages sent by nodes in the network to all other nodes within the same network. These messages are broadcasted to inform other nodes that the node sending the heartbeat is still awake and the number of hops within which it can be reached. This functionality is efficiently done by flooding because it has a higher reach of all nodes in the network within a single transmission of a message. These messages are sent to all nodes using a network key (NetKey) as the destination address since all node in the network have it.

The other functionality is the use of subnetworks which is a group of nodes within a Bluetooth Mesh network with a similar identification key known as the subnet key. The best and efficient way to reach a subnet is using a flooding algorithm because a single message will be flooded to all members of the subnetwork at once using the subnet key. On the other hand, OSPF

would require individual device Keys of all nodes in the subnetwork and send the messages individually to all of them.

Lastly, Bluetooth Mesh was also designed to increase the probability of successful message delivery. Given that some of the IoT nodes in a mesh network could be in remote areas, there are chances of losing the messages and connectivity before messages get to their destination. Therefore, there is need to use alternative paths to the destination to increase the success rate of messages delivered even when some nodes are inaccessible. For the case of OSPF, if an inactive node is encountered, the path is cut off and therefore the message won't be delivered until the source node is prompted to resend because of no acknowledgment was received. For example, in the first simulation having 3 edges per node, the managed flooding has 3 paths to take at each node which increases the chances of delivering the message. In the case of sending the message from node 0 to 25, node 25 can receive the message either through 18, 20 and 24 making it a better option for increasing the probability of delivering a message. Yet, OSPF will only deliver the message through node 20

4.3 Chapter Summary

This chapter states the results and analyses them after running the simulation three times for all the 3 protocols. The results are analyzed in accordance with the efficient energy consumption need of the nodes and the functional limitations in the previous version that Bluetooth Mesh addresses. Chapter 5, lays out the future works, limitations and the conclusion of the entire project.

Chapter 5: Conclusion

The project sought to simulate and analyze the efficiency of the routing protocol used by the new Bluetooth Mesh that was released in the year 2017. Being a new technology, the available simulators could not be used to simulate Bluetooth Mesh which brought in a new requirement of designing a simulator first before doing the evaluation and analysis. The simulation was done using Java programing language and all the routing algorithms used in this project were also built in Java programing language.

Managed flooding, an optimized approach of a flooding algorithm is evaluated alongside pure flooding and OSPF routing protocol that uses Dijkstra's algorithm mainly in terms of energy efficiency and the functional requirements of Bluetooth Mesh. OSPF is efficient when it comes to using the least energy while transmitting messages but managed flooding is a better solution because of its ability to efficiently cater for low power nodes, high probability of message delivery, group message delivery, and heartbeat message delivery. Hence, although managed flooding uses a lot of resources while sending a single message, it meets the needs of low power nodes and the main functionalities of Bluetooth Mesh which make it a superior version compared to is predecessors.

5.1 Limitations

The simulation tackled the energy need problem generally. Although the nodes in the network have different power needs. Low power nodes have friend nodes that receive and send messages on their behalf while they are asleep and delivery is done later when they are awake which enables them to save energy. The project did not implement this feature because in a case where the destination node is a low power node that is not awake, message delivery may be delayed to an unknown time.

5.2 Future work

Future work may start by implementing the friendship and heartbeat requirement of Bluetooth Mesh which will involve analyzing results from nodes with varying energy requirements. Future work may also focus on analyzing the efficiency of routing protocols in broadcasting group messages that are sent to subnets. Evaluation and research on other optimized flooding algorithms such as BLEmesh and Dynamic probabilistic flooding algorithm versus the managed flooding algorithm can also be done because both of them are optimized versions of the flooding algorithm. In addition to this project, future work may also consider revaluation of the energy efficiency of both flooding and OSPF by also considering the OSPF overhead cost of finding the cost of all routes which matches flooding that does both routing and flooding at the same time

References

Bluetooth SIG. (2017a). Bluetooth Mesh Networking. Bluetooth SIG, Inc.

Bluetooth SIG. (2017b). Core Specifications | Bluetooth Technology Website. Retrieved September 18, 2017, from https://www.bluetooth.com/specifications/bluetooth-core-specification

Bluetooth SIG. (2016). Bluetooth® 5 Quadruples Range, Doubles Speed, Increases Data Broadcasting Capacity by 800% | Bluetooth Technology Website. Retrieved April 14, 2018, from https://www.bluetooth.com/news/pressreleases/2016/06/16/-bluetooth-5-quadruples-rangedoubles-speedincreases-data-broadcasting-capacity-by-800

Darroudi, S. M., & Gomez, C. (2017). Bluetooth Low Energy Mesh Networks: A Survey. *Sensors*, *17*(7), 1467. https://doi.org/10.3390/s17071467

DeCuir, J. (2014). Introducing Bluetooth Smart: Part 1: A look at both classic and new technologies. *IEEE Consumer Electronics Magazine*, *3*(1), 12–18. https://doi.org/10.1109/MCE.2013.2284932

Gogic, A., Mujcic, A., Ibric, S., & Suljanovic, N. (2016). Performance Analysis of Bluetooth Low Energy Mesh Routing Algorithm in Case of Disaster Prediction, *10*(6).

Guo, Z., Harris, I. G., Tsaur, L. f, & Chen, X. (2015). An on-demand scatternet formation and multi-hop routing protocol for BLE-based wireless sensor networks. In *2015 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 1590–1595). https://doi.org/10.1109/WCNC.2015.7127705

Hortelano, D., Olivares, T., Ruiz, M. C., Garrido-Hidalgo, C., & López, V. (2017). From Sensor Networks to Internet of Things. Bluetooth Low Energy, a Standard for This Evolution. *Sensors*, *17*(2), 372. https://doi.org/10.3390/s17020372

Kim, H. S., Lee, J., & Jang, J. W. (2015). BLEmesh: A Wireless Mesh Network Protocol for Bluetooth Low Energy Devices. In *2015 3rd International Conference on Future Internet of Things and Cloud* (pp. 558–563). https://doi.org/10.1109/FiCloud.2015.21

Madakam, S., Ramaswamy, R., & Tripathi, S. (2015). Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*, *03*(05), 164. https://doi.org/10.4236/jcc.2015.35021

Madhvi Verma, Satbir Singh, & Baljit Kaur. (2015). An Overview of Bluetooth Technology and its Communication Applications, *5*(3), 2347–5161.

Mohammed Humayun Kabir, Syful Islam, Md. Javed Hossain, & Sazzad Hossain. (2014). Detailed Comparison of Network Simulators, *5*(10), 203–218.

Mostafa, I., & Islam, G. (2016). Improved SDR Frequency Tuning Algorithm for Frequency Hopping Systems. *ETRI Journal*, *38*(3), 455–462. https://doi.org/10.4218/etrij.16.0115.0565

Nilsson, E., & Lindman, T. (2017). *Implementation and evaluation of Bluetooth Low Energy as a communication technology for wireless sensor networks*. Linköping: Linköping University Electronic Press. Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138672

Park, S.-H., Cho, S., & Lee, J.-R. (2014). Energy-Efficient Probabilistic Routing Algorithm for Internet of Things [Research article]. Retrieved March 21, 2018, from https://www.hindawi.com/journals/jam/2014/213106/

Rani, S., Talwar, R., Malhotra, J., Ahmed, S. H., Sarkar, M., & Song, H. (2015). A Novel Scheme for an Energy Efficient Internet of Things Based on Wireless Sensor Networks. *Sensors (Basel, Switzerland)*, *15*(11), 28603–28626. https://doi.org/10.3390/s151128603

Ray, P. P., & Agarwal, S. (2016). Bluetooth 5 and Internet of Things: Potential and architecture. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)* (pp. 1461–1465). https://doi.org/10.1109/SCOPES.2016.7955682

Sethi, P., & Sarangi, S. R. (2017). Internet of Things: Architectures, Protocols, and Applications [Research article]. Retrieved February 23, 2018, from https://www.hindawi.com/journals/jece/2017/9324035/

Sezer, O. B., Dogdu, E., & Ozbayoglu, A. M. (2018). Context-Aware Computing, Learning, and Big Data in Internet of Things: A Survey. *IEEE Internet of Things Journal*, *5*(1), 1–27. https://doi.org/10.1109/JIOT.2017.2773600

Soni, H., & Khunteta, A. (2014). Performance analysis of routing protocols in Bluetooth networks. In *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)* (pp. 1–9). https://doi.org/10.1109/ICRAIE.2014.6909231

White, K. P., & Ingalls, R. G. (2017). The basics of simulation. In *2017 Winter Simulation Conference (WSC)* (pp. 505–519). https://doi.org/10.1109/WSC.2017.8247811

Zarrad, A., & Alsmadi, I. (2017). Evaluating network test scenarios for network simulators systems. *International Journal of Distributed Sensor Networks*, *13*(10), 1550147717738216. https://doi.org/10.1177/1550147717738216

Zhong, C. l, Zhu, Z., & Huang, R. G. (2017). Study on the IOT Architecture and Access Technology. In *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)* (pp. 113–116). https://doi.org/10.1109/DCABES.2017.32

Appendix A (Java Code)

Main Class: BluetoothMesh

```
package bluetoothmesh;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.util.Scanner;

/**
 *
 * @author jobmwesigwa
 */
public class BluetoothMesh {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws
FileNotFoundException {

        Network myNetwork = new Network();
        final int NUM_OF_DEVICES =26;
        Device[] devices = new Device[NUM_OF_DEVICES];
        String file="";
        int edges;
        Scanner in = new Scanner(System.in);


        /**
         * user selects the number of edges and the desired
configuration file
         * and log files set for both the input and output
         */
        do{
        System.out.println("Enter the number of edges (3,4 or 5): ");
        edges = in.nextInt();

        if (edges == 3){
            file = "config/Configmesh3.txt";
            PrintStream out = new PrintStream(new
FileOutputStream("logs/Mesh3Output.log"));
            System.setOut(out);
        }
        else if (edges == 4){
            file = "config/Configmesh4.txt";
```

```
            PrintStream out = new PrintStream(new
FileOutputStream("logs/Mesh4Output.log"));
            System.setOut(out);
        }
        else if (edges == 5){
            file = "config/Configmesh5.txt";
            PrintStream out = new PrintStream(new
FileOutputStream("logs/Mesh5Output.log"));
            System.setOut(out);
        }
        else
            System.out.println("Invalid entry: enter 3,4 or 5: ");
        }while (edges<3 || edges>5);


        /**
         * set the instances of the each node and run all the 3
algorithms
         */
        try (BufferedReader myString = new BufferedReader(new
InputStreamReader(new FileInputStream(file))))
        {
            String currentLine=null;
            int counter =0;
            int counter1 =-1;
            while ((currentLine = myString.readLine())!=null)
            {
                if (counter ==0){
                }
                else{
                    String [] myNode = currentLine.split("\t");
                    devices[counter1] = new
Device(Integer.parseInt(myNode[1]));

                    if (Integer.parseInt(myNode[3])==1)
                        devices[counter1].setRelay(true);

                    String [] nieg = myNode[2].split(",");
                    int [] neighs = new int[nieg.length];

                    for (int i =0; i<nieg.length; i++)
                        neighs[i] = Integer.parseInt(nieg[i]);
                    devices[counter1].addNeighbours(neighs);

                    myNetwork.addDevice(devices[counter1]);

                }
                counter++;
                ++counter1;
            }
        }
            //catching any excemtions
```

```
        catch (Exception e){
              System.out.println("Error -Other Exception
"+e.toString());
            }

        //flooding
        System.out.println();
        System.out.println("\n\n \t\t ********Flooding********\n\n");
        System.out.println();
        Flooding flooding = new Flooding(myNetwork);
        BTMessage message = new
BTMessage(devices[0],devices[25],1,"message");
        flooding.flood(message);

        //Managed flooding
        System.out.println();
        System.out.println();
        System.out.println("\n\n \t\t ********Managed
Flooding********\n\n");
        System.out.println();
        message.setMessageFrequency(2);
        flooding.floodManaged(message);

        //OSPF
        System.out.println();
        System.out.println();
        System.out.println("\n\n \t\t ********OSPF********\n\n");
        System.out.println();
        OSPF ospf = new OSPF();
        ospf.sendMessage(myNetwork,message);
    }
}
```

Node class: Device

```
package bluetoothmesh;

import java.util.ArrayList;
import java.util.Stack;
/**
 *
 * @author jobmwesigwa
 */
public class Device {


    private boolean relay = false;
    private int deviceID = 0;
    private ArrayList<Integer> neighbors, cache = new
ArrayList<Integer>();
    public String txt = "";
    private Boolean visited = false;
```

```
public Stack<Integer> path = new Stack<Integer>();

//instances required for OSPF
private int dist = Integer.MAX_VALUE;
public ArrayList<Integer> shortestPath = new ArrayList<Integer>();
/**
 *
 * @param devKy the device address/key
 */
public Device (int devKy)
{
    deviceID = devKy;
}

/**
 * Adds neighbors to the device neighbors list
 * @param preds an array of neighbors
 */
public void addNeighbours(int...preds){
    ArrayList adds = new ArrayList();
    if (preds.length>0){
        for (int address : preds) {
            adds.add(address);
        }
    }
    setNeighbors(adds);
}

/**
 * sets the path of the current node
 * @param sender the address of the sending node
 * @param prevPath the original path to the current node
 */
public void setPath(int sender, Stack<Integer> prevPath)
{
    if (prevPath.isEmpty() || prevPath.peek()!=sender)
        prevPath.push(sender);
    path = (Stack)prevPath.clone();
}
/**
 *
 * @return Boolean whether the node is relay or not
 */
public boolean isRelay() {
    return relay;
}
/**
 *
 * @return the address of the device
 */
public int getDeviceID() {
    return deviceID;
```

```
    }
    /**
     *
     * @param deviceID the address if the node
     */
    public void setDeviceID(int deviceID) {
        this.deviceID = deviceID;
    }
    /**
     *
     * @return the list of neighbors
     */
    public ArrayList<Integer> getNeighbors() {
        return neighbors;
    }
    /**
     *
     * @param neighbors the list of neighbors
     */
    public void setNeighbors(ArrayList<Integer> neighbors) {
        this.neighbors = neighbors;
    }
    /**
     *
     * @param relay a boolean state of the node if it is relay or not
     */
    public void setRelay(Boolean relay) {
        this.relay = relay;
    }
    /**
     * @return the cache the list of frequencies that have been
received
     */
    public ArrayList<Integer> getCache() {
        return cache;
    }
    /**
     * @param cache the cache to set
     */
    public void setCache(ArrayList<Integer> cache) {
        this.cache = cache;
    }
    /**
     *
     * @return text the message sent
     */
    public String getTxt() {
        return txt;
    }
    /**
     *
     * @param txt the message sent
```

```
 */
public void setTxt(String txt) {
    this.txt = txt;
}
/**
 *
 * @return path to the node
 */
public Stack<Integer> getPath() {
    return path;
}
/**
 *
 * @param path the path to the node
 */
public void setPath(Stack<Integer> path) {
    this.path = path;
}
/**
 *
 * @return distance to the node
 */
public int getDist() {
    return dist;
}
/**
 *
 * @param dist distance to the node
 */
public void setDist(int dist) {
    this.dist = dist;
}
/**
 *
 * @return the shortest path to the node
 */
public ArrayList<Integer> getShortestPath() {
    return shortestPath;
}
/**
 *
 * @param shortestPath the shortest path to the node
 */
public void setShortestPath(ArrayList<Integer> shortestPath) {
    this.shortestPath = shortestPath;
}
/**
 * sets the node to a new state of visited
 * @param visited the visited to set
 */
public void setVisited(Boolean visited) {
    this.visited = visited;
```

```
    }
    /**
     *
     * @return the state of the node whether visited
     */
    public boolean isVisited() {
        return visited;
    }
}
```

Message Class: BTMessage

```
package bluetoothmesh;

/**
 *
 * @author jobmwesigwa
 */
public class BTMessage {
    private Device sender;
    private Device receiver;
    private int messageFrequency = 0;
    private String message ="";
    /**
     *
     * @param sender source node
     * @param receiver destination node
     * @param freq frequency of the message
     * @param txt the payload
     */
    public BTMessage(Device sender, Device receiver, int freq, String
txt) {
        this.sender = sender;
        this.receiver = receiver;
        this.messageFrequency = freq;
        this.message = txt;
    }
    /**
     *
     * @return sender the source node
     */
    public Device getSender() {
        return sender;
    }
    /**
     *
     * @param sender the source node
     */
    public void setSender(Device sender) {
        this.sender = sender;
    }
```

```
    /**
     *
     * @return receiver the destination node
     */
    public Device getReceiver() {
        return receiver;
    }
    /**
     *
     * @return messageFrequency
     */
    public int getMessageFrequency() {
        return messageFrequency;
    }
    /**
     *
     * @param messageFrequency
     */
    public void setMessageFrequency(int messageFrequency) {
        this.messageFrequency = messageFrequency;
    }
     /**
     * @return the payload
     */
    public String getMessage() {
        return message;
    }
    /**
     * @param message the payload
     */
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Network Class: Network

```
package bluetoothmesh;

import java.util.ArrayList;

/**
 *
 * @author jobmwesigwa
 */
public class Network {
    public ArrayList<Device> network = new ArrayList();
    int counter =0;

    /**
     * Adds the new device to the network arraylist and adds its
reference to
```

```
     * devices connected to it
     * @param newDev the new device to be added
     */
    public void addDevice(Device newDev){
        network.add(newDev);
    }
    /**
     * removed the identified device from the network
     * @param newDev the new device to be added
     */
    public void deleteDevice(Device newDev){
        network.remove(newDev);
    }

}
```

OSPF Class: OSPF

```
package bluetoothmesh;

import java.util.ArrayList;

/**
 *
 * @author jobmwesigwa
 */
public class OSPF {
    Network myNet;
    ArrayList<Integer> messagePath;

    /**
     * @param network the network of nodes to be traversed
     * @param message the message to be sent
     *
     * gets the shortest path
     * Uses it to send the message
     */
    public void sendMessage(Network network, BTMessage message){
        Device sender = message.getSender();
        Device destination = message.getReceiver();
        myNet = network;
        messagePath = findPath(myNet,sender,destination);
        if (!messagePath.contains(destination.getDeviceID()))
            messagePath.add(destination.getDeviceID());
        for (int i = 0; i < messagePath.size(); i++) {
            System.out.println("Message sent to " +
messagePath.get(i));
            if (messagePath.get(i)==destination.getDeviceID()){
                destination.setTxt(message.getMessage());
                System.out.println();
                System.out.println("Message Successfully delivered to
"+sender.getDeviceID());
```

```
                System.out.println("Message Path:
"+messagePath.toString());
                System.out.println("Total Hop count:
"+messagePath.size());
                System.out.println("Hop count: "+messagePath.size());
            }
        }
    }

    /**
     * @param network the network of nodes to be traversed
     * @param source the source node
     * @param dest the destination node
     * @return List of shortest path from source to destination
     * initiate the distance of the fist node and add it to the
visited nodes
     * Pick a node from the unvisited list
     * Update the path of its neighbors and add them to the unvisited
list
     * add the node to the visited list
     */
    private ArrayList findPath(Network network, Device source, Device
dest){

        //Keeping track of the visted and unvisted devices in the
network
        ArrayList<Device> visited = new ArrayList();
        ArrayList<Device> unvisted = new ArrayList();

        //setting the distance of the first device and addindg it to
the unvisited
        source.setDist(0);
        unvisted.add(source);

        //Iterating through the unvisted list
        while (!unvisted.isEmpty()) {

            Device curDev = null;
            int minDist = Integer.MAX_VALUE;

            for (int i=0; i<unvisted.size();i++) {
                int devDist = unvisted.get(i).getDist();
                if (devDist < minDist) {
                    minDist = devDist;
                    curDev = unvisted.get(i);
                }
            }

            unvisted.remove(curDev);

            if (curDev.getNeighbors()!=null){
                int trackVisted =0;
```

```
                for(int i=0; i<curDev.getNeighbors().size();i++) {
                    //considering the weight of each node to be 1
                    int dist = 1;
                    Device neighbour =
network.network.get(curDev.getNeighbors().get(i));

                    //updating the distance to neighbour and adding it
to unvisited list
                    if (!visited.contains(neighbour)) {
                        getMinDistance(neighbour, dist, curDev);
                        unvisted.add(neighbour);
                    }
                    else if (visited.contains(neighbour))
                        trackVisted++;

                    //adding current node to shortest path
                    if (trackVisted==curDev.getNeighbors().size() &&
!curDev.getShortestPath().contains(curDev.getDeviceID()))

curDev.getShortestPath().add(curDev.getDeviceID());

                }
            }
            visited.add(curDev);
        }

        return dest.getShortestPath();
    }
    /**
     * @param neigh the next node
     * @param current the node being visited
     * @param dist distance to next node
     *
     * Checks if the new distance will improve the distance to the
next node,
     * if it does, then it updates the path to the next distance
     */
    private void getMinDistance(Device neigh, int dist, Device
current){
        int prevDistance = current.getDist();
        if (prevDistance + dist < neigh.getDist()) {
            neigh.setDist(prevDistance + dist);
            ArrayList<Integer> shortestPath =
(ArrayList)current.getShortestPath().clone();
            shortestPath.add(current.getDeviceID());
            neigh.setShortestPath(shortestPath);
        }
    }

}
```

Flooding and Managed Flooding Class: Flooding

```
package bluetoothmesh;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

/**
 *
 * @author jobmwesigwa
 */
public class Flooding {
    Network myNet;
    int hopCount = 0;
    int maxHopCount = 150;
    public Flooding (Network network){
        myNet = network;
    }

    /**
     * Loops through the network to find destination or else stops
when the
     * maximum hope count is reached.
     * At each node, goes through all neighbors to send message
     * discards a message that has already been received
     * adds the neighbors to queues of nodes to be checked out if it's
a relay node
     * delivers the message if one of the neighbors is the destination
     * removes the current node from the queue
     * @param message the message to be sent
     */
    public void floodManaged(BTMessage message){
        String text = message.getMessage();
        int freq = message.getMessageFrequency();
        Device sender =message.getSender();
        int receiverID = message.getReceiver().getDeviceID();
        refresh();
        //for bfs search
        Queue<Integer> queue = new LinkedList();
        int senderId = sender.getDeviceID();
        hopCount = 0;
        sender.getCache().add(freq);
        //loop to run through the entire network
        do {
            sender.setVisited(true);
            //neigbours of the current node and adding the frequency
if the message to node
            int neigh = sender.getNeighbors().size();
            System.out.println("Message is currently at "+
sender.getDeviceID());
            //going through neighbours
```

```
            for (int i=0;i<neigh;i++){
                int curAdd = (sender.getNeighbors().get(i));
                int senderID = sender.getDeviceID();

                System.out.println("\nMessage sent to " + curAdd);
                //deliver message if destination is reached
                if (curAdd== receiverID){


(myNet.network.get(curAdd)).setPath(senderID,(myNet.network.get(sender
ID).path));
                    myNet.network.get(curAdd).path.push(curAdd);
                    myNet.network.get(receiverID).txt=text;
                    unicast("Message Delivered",
myNet.network.get(receiverID).path, myNet.network.get(receiverID),
senderId);
                    return;
                }


                //ignore node if it has already been visted
                if
((myNet.network.get(curAdd)).getCache().contains(freq))
                    System.out.println("The message has already
arrived at node "+ (myNet.network.get(curAdd)).getDeviceID() + ",
Discard please");

                //add current node to the visited nodes and to the
path taken by the message
                if (!(myNet.network.get(curAdd).isVisited())){

(myNet.network.get(curAdd)).setPath(senderID,(myNet.network.get(sender
ID).path));
                    if (myNet.network.get(curAdd).isRelay())
                        queue.add(curAdd);
                }

               myNet.network.get(curAdd).getCache().add(freq);
               hopCount++;
            }
            //updating the sender

            sender = myNet.network.get(queue.remove());
            System.out.println();
            System.out.println();
        } while (!queue.isEmpty() && hopCount<maxHopCount);
    }
    /**
     * Loops through the network to find destination or else stops
when the
     * maximum hope count is reached.
     * At each node, goes through all neighbors to send message
```

```
     * discards a message that has already been received
     * adds the neighbors to queues of nodes to be checked out
     * delivers the message if one of the neighbors is the destination
     * removes the current node from the queue
     * @param message the message to be sent
     */
    public void flood(BTMessage message){
        String text = message.getMessage();
        int freq = message.getMessageFrequency();
        Device sender =message.getSender();
        int receiverID = message.getReceiver().getDeviceID();
        //for bfs search
        refresh();
        Queue<Integer> queue = new LinkedList();
        int senderId = sender.getDeviceID();
        hopCount = 0;
        sender.getCache().add(freq);
        //loop to run through the entire network
        do {
            sender.setVisited(true);
            //neigbours of the current node and adding the frequency
if the message to node
            int neigh = sender.getNeighbors().size();
            System.out.println("Message is currently at "+
sender.getDeviceID());
            //going through neighbours
            for (int i=0;i<neigh;i++){
                int curAdd = (sender.getNeighbors().get(i));
                int senderID = sender.getDeviceID();

                System.out.println("\nMessage sent to " + curAdd);
                //deliver message if destination is reached
                if (curAdd== receiverID){
(myNet.network.get(curAdd)).setPath(senderID,(myNet.network.get(sender
ID).path));
                    myNet.network.get(curAdd).path.push(curAdd);
                    myNet.network.get(receiverID).txt=text;
                    unicast("Message Delivered",
myNet.network.get(receiverID).path, myNet.network.get(receiverID),
senderId);
                    return;
                }


                //ignore node if it has already been visted
                if
((myNet.network.get(curAdd)).getCache().contains(freq))
                    System.out.println("The message has already
arrived at node "+ (myNet.network.get(curAdd)).getDeviceID() + ",
Discard please");
```

```
                //add current node to the visited nodes and to the
path taken by the message
                if (!(myNet.network.get(curAdd).isVisited())){

(myNet.network.get(curAdd)).setPath(senderID,(myNet.network.get(sender
ID).path));
                    queue.add(curAdd);
                }

               myNet.network.get(curAdd).getCache().add(freq);
               hopCount++;
            }
            //updating the sender

            sender = myNet.network.get(queue.remove());
            System.out.println();
            System.out.println();
        } while (!queue.isEmpty() && hopCount<maxHopCount);
    }

    /**
     * @param msg the message to be delivered
     * @param backPath the path of nodes back to source
     * @param sender the source node
     * @param receiverId the address of the destination node
     * @return Boolean the confirmation whether the message is
delivered.
     * Sends the acknowledgement  message back to source using the
path it took
     * on delivery
    */
    public boolean unicast(String msg, Stack<Integer> backPath, Device
sender, int receiverId){
        int size =backPath.size();
        System.out.println();
        System.out.println();
        System.out.println("Message Successfully delivered to
"+sender.getDeviceID());
        System.out.println("The message path is "+
backPath.toString());
        while (!backPath.isEmpty()){
          if (receiverId == backPath.pop()){
              System.out.println("The total hope count is:
"+hopCount);
              System.out.println("The path length is: " + size);
              return true;
          }
      }
      System.out.println("Feedback from Receiver not received");
      return false;
    }
```

```
    /*
    *Refreshes the nodes from being visted after runing the algorithm
    */
    public void refresh(){
        for (int i =0; i<myNet.network.size();i++)
            if (myNet.network.get(i).isVisited())
                myNet.network.get(i).setVisited(false);
    }
}
```

Appendix B (Configuration Files)

3 edges Configuration file: Configmesh3.txt

Name	ID	Neighbors	Relay
A	0	1,2,3	0
B	1	0,2,4	1
C	2	0,1,5	0
D	3	0,6,7	1
E	4	1,8,9	1
F	5	2,6,9	1
G	6	3,5,10	1
H	7	3,11,12	0
I	8	4,13,14	1
J	9	4,5,15	1
K	10	6,11,15	1
L	11	7,10,16	1
M	12	7,16,17	0
N	13	8,18,19	1
O	14	8,19,20	1
P	15	9,10,20	1
Q	16	10,11,21	1
R	17	12,22,23	1
S	18	13,19,25	1
T	19	13,14,18	1
U	20	14,15,25	1
V	21	16,22,24	1
W	22	17,21,23	1
X	23	17,22,24	0
Y	24	21,23,25	1
Z	25	18,20,24	0

4 edges Configuration file: Configmesh4.txt

Name	ID	Neighbors	Relay
A	0	1,2,3,5	1
B	1	0,5,8,20	1
C	2	0,5,6,9	1
D	3	0,4,6,7	0
E	4	0,3,7,12	1
F	5	1,2,8,9	0
G	6	2,3,10,14	1
H	7	3,4,10,11	0
I	8	1,5,13,20	0
J	9	2,5,13,14	1
K	10	6,7,15,18	1
L	11	7,12,15,19	0
M	12	4,11,19,23	0
N	13	8,9,16,17	0
O	14	6,9,17,18	1
P	15	10,11,19,23	0
Q	16	13,20,21,25	1
R	17	13,14,21,22	0
S	18	10,14,22,24	1
T	19	11,12,15,23	0
U	20	1,8,16,21	1
V	21	16,17,20,25	0
W	22	17,18,24,25	1
X	23	11,12,15,24	0
Y	24	18,22,23,25	0
Z	25	16,21,22,24	1

5 edges Configuration file: Configmesh5.txt

Name	ID	Neighbors	Relay
A	0	1,2,3,4,5	1
B	1	0,2,6,7,8	1
C	2	0,1,3,6,8	0
D	3	0,2,4,8,9	1
E	4	0,3,9,10,11	0
F	5	0,10,11,16,18	1
G	6	1,2,7,12,13	1
H	7	1,6,13,15,16	0
I	8	1,2,3,9,15	0
J	9	3,4,8,10,16	0
K	10	4,5,9,11,17	0
L	11	4,5,10,17,24	1
M	12	6,13,19,20,25	1
N	13	6,7,12,14,19	1
O	14	7,13,15,20,21	1
P	15	7,8,14,16,21	0
Q	16	5,9,15,17,22	1
R	17	10,11,16,18,22	1
S	18	5,17,22,24,25	1
T	19	12,13,20,23,24	1
U	20	12,14,19,21,25	1
V	21	14,15,20,22,23	0
W	22	16,17,18,21,23	1
X	23	19,21,22,24,25	0
Y	24	11,18,19,23,25	0
Z	25	12,18,20,23,24	1