IMPLEMENTATION OF A LOCATION-BASED WIRELESS SENSOR NETWORK

A Thesis By

DOMINIC JACOBO
ORCID iD: 0009-0000-0985-5775

California State University, Fullerton
Fall, 2024

In partial fulfillment of the degree:
Master of Science in Computer Engineering

Department:
Department of Electrical and Computer Engineering

DOI:
10.5281/zenodo.14549955

Keywords:
WSN, localization, P2P, Bluetooth Mesh, single hop, anchor nodes

Abstract:

Wireless sensor networks (WSNs) are distributed in various areas, including military, industrial, and consumer applications, for tasks like surveillance, environmental monitoring, or machine control. They are an emerging technology with heavy investment and many hours of research going into the topic, and because of this many WSNs have already been created and dispersed. This paper proposes the implementation of a WSN with environmental monitoring features and with the added feature of localization of nodes in the network. The localization feature performs single hop distance measurement using peer-to-peer (P2P) measurements between the mesh nodes in a Bluetooth mesh network. This gives selected nodes in the WSN the ability to locate other nodes of interest across the network and provide the estimated distance between the two nodes.

TABLE OF CONTENTS

LIST OF TABLES iii

LIST OF FIGURES iv

ACKNOWLEDGMENTS v

1. INTRODUCTION 1

2. BLUETOOTH LE, BLUETOOTH MESH, AND DISTANCE MEASUREMENT PHYSICS 4

    Bluetooth LE or Bluetooth Classic? 4
    Bluetooth Low Energy 5
        BLE Controller 6
        BLE Host 10
    Bluetooth Mesh 11
        Bluetooth Mesh Fundamental Concepts 12
        Provisioning How Bluetooth Mesh Networks Are Created 17
    Distance Measurement Physics 20
        Received Signal Strength Indicator (RSSI) 20
        Time-of-Flight (ToF) 21
        Multi-Carrier Phase Slope Estimation (MCPD) 21
        Inverse Fast Fourier Transform (IFFT) 22

3. IMPLEMENTATION, PEER-TO-PEER, AND RESULTS 24

    Hardware setup and implementation 24
    Software setup and implementation 27
    One-Hop Distance Measurement Algorithm 31
    Results 34

4. CONCLUSION AND FUTURE WORK 38

    Conclusion 38
    Future Work 38

REFERENCES 40

LIST OF TABLES

Table Page

1. Comparison of the three PHY variants 7
2. Mesh address encoding 16
3. Phase-based characteristics of different frequencies 22
4. Results of first distance measurement test scenario. 35
5. Results of second distance measurement test scenario. 36
6. Results of third distance measurement test scenario. 37

LIST OF FIGURES

Figure Page

1. Example network topologies commonly used by WSNs.. 2
2. The Bluetooth LE Stack.. 6
3. State diagram of the link layer state machine 9
4. Nodes in a mesh network.. 13
5. Structure of a mesh node using elements and models 15
6. Confirmation value generation 19
7. The Thingy 53 Environmental and Color Sensors... 26
8. Distance measurement procedure 30
9. The structure of the location-based WSN... 34

ACKNOWLEDGMENTS

To my parents, Victor and Amalia, who have given me so much and whom I could write a document just as long thanking, I hope I have made you proud. To my sister, Jackie, who has been my biggest support system, an ocean of wisdom, and who led the path forward into higher education so that I could follow. To my brother, Beek, who was my childhood hero, the man I respect the most today, and whose strength and character I hope to emulate in myself. To Dr. Mike Turi, who was the first to spark the fire in me for learning Computer Engineering, an amazing teacher in so many respects, and one I’m glad to call my mentor. Lastly, I thank Samantha Rehome, whose hard work, dedication, and belief in me were inspirations to keep going even when I didn’t think I had it in me.

CHAPTER 1

INTRODUCTION

Wireless sensor networks (WSNs) are currently a very active and significant field of research and development. They are used in numerous applications, such as environmental monitoring, healthcare systems, and military surveillance [1]. A wireless sensor network is a network of various sensors that collect, monitor, and store data that can be shared with other devices in the network using a wireless communication protocol. The data measured from the sensors is later sent across the network to a base station, a more powerful device that further processes and evaluates the data to be later used by a user or administrator of the network [1].

WSNs can be structured in a myriad of different ways, but each WSN needs to have a few common factors. First, WSNs are built of nodes. There can be hundreds or thousands of nodes that construct a network. Each node requires a few mandatory components to function properly as a node in the network. First, nodes require a microcontroller or other kind of processing device that can run the necessary firmware and connect to the necessary peripherals [1]. One of those necessary peripherals that the node requires is a radio transceiver and antenna to send and receive wireless communication packets. The node also requires a power source, which can be a small battery or connected to an energy harvesting system, depending on the role and function of the node [1].

Another factor needed for the construction of the network is the implementation of its topology. A network topology is the layout of the physical and logical connections of the nodes [2]. This layout is critical because it will affect many aspects of the network's performance, like its latency, scalability, security, power consumption, etc. The topology used should be based highly on the application's specifications and the environment in which it is placed. Some examples of common topologies used for WSNs are star, tree, mesh, or hybrid topologies. A visual summary of these topologies' general nodes and link configurations can be seen in Figure 1.

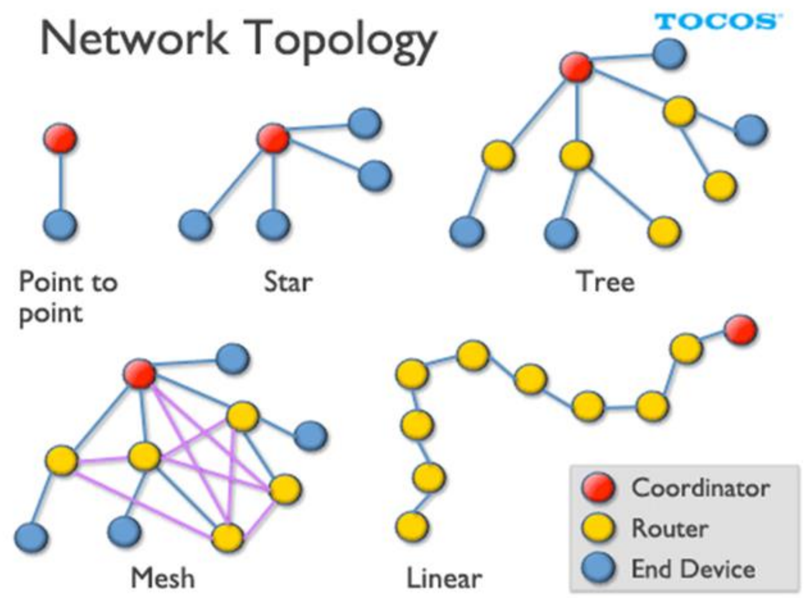

Figure 1. Example network topologies commonly used by WSNs

When looking at the network from a topology view, there are three primary node types: the coordinator, router, and end device [2]. The coordinator in a WSN will be our base station, so it's a place in the network for the aggregation and analysis of data collected elsewhere in the network. The router's function is for moving data from a source node to a destination node. Lastly, the end device in a WSN will be a sensor node; it will collect relevant data with the goal of sending this data back across the network to the coordinator.

The wireless communication protocol is one of the last essential features to consider when creating a WSN. There are many, but the four most used protocols for WSNs are Bluetooth Low Energy (BLE), LoRa, ZigBee, and Wi-Fi. These Wireless network protocols are used to connect devices wirelessly by setting rules that define how the devices communicate and exchange data. An example would be LoRa's long-range radio technology that uses chirp spread spectrum modulation to send information over long distances or ZigBee's low-power, low-range radio technology that natively supports mesh networking.

Although a developer is free to use any topology they like, wireless protocols are designed in such a way that some are more suited to specific topologies than others. For example, LoRa is ideal for a star topology due to its long range, which eliminates the need for routers. On the other hand, ZigBee fits seamlessly into a mesh topology due to its low power consumption and built-in mesh capability.

All the mentioned criteria were considered when constructing the location-based wireless sensor network. This paper will begin by discussing BLE, Bluetooth mesh, and the physics involved in performing distance measurements between nodes in a network. This chapter will review the wireless protocol rules for configuring node functionality, link configuration, and data traversal in a mesh network, as well as the physics behind Nordic's distance measurement toolbox, which will be used in the final design to perform one hop node localization. The following chapter will detail the WSN setup, including the hardware required, its placement in the network, each node's function, the technique used to do one-hop node location, the sensor node and its readings, and the peer-to-peer (P2P) distance measurement results. The last chapter will explore future work, such as adding more anchor nodes to the network for greater scalability and the option for localization beyond single hop, as well as other valuable capabilities that the WSN would benefit from.

CHAPTER 2

BLUETOOTH LE, BLUETOOTH MESH, AND DISTANCE MEASUREMENT PHYSICS

Bluetooth LE or Bluetooth Classic?

Bluetooth is a prevalent technology today because of its widespread implementation into smartphones for audio streaming and connecting devices like speakers and headphones. When it comes to Bluetooth, there are two distinct architectures: Bluetooth Classic and BLE [3]. Bluetooth Classic is the precursor to BLE, and many devices that use Bluetooth have both. When choosing between these two architectures, it’s essential to think about which of the two best fits a WSN’s goals.

Both architectures have many similarities, such as performing short-range wireless communication and operating in the 2.4 GHz industrial, scientific, and medical (ISM) band [3]. Where they differ is their initial design goals and their concern over power consumption. Bluetooth Classic does not really concern itself with power consumption. It was developed for continuous wireless voice and data transmission. It operates on a high data rate of up to 3 Mbps and uses a robust error correction feature to send a steady data stream without interruption, which consumes a lot of power [3]. BLE, on the other hand, is tailored for low power consumption and intermittent data transmission. BLE achieves its energy efficiency by allowing devices to remain in sleep mode most of the time and wake up only when communication is necessary [3].

Power consumption is a critically important aspect to consider for WSNs. WSNs typically have hundreds or thousands of sensor nodes, which are expected to run on a limited battery. Also, many WSNs, including the one implemented in this paper, send small data packets at irregular intervals. So many of Bluetooth Classic’s extra features, like high over-the-air data rates, high application throughput, and robust error correction, are antithetical to the WSN’s goal of low power consumption and it’s overkill for many WSN use cases.

BLE was chosen as this network's wireless communication technology for a variety of reasons, including those mentioned. It was chosen not only for its low power consumption and intermittent data transmission but also for its rich evolution of features that will be explored in the coming sections. The final reason it was chosen was because of its vast and easily accessible documentation, and its extensive online community support and resources.

Bluetooth Low Energy

In this section, BLE’s internal workings will be discussed because it is the foundation of its mesh networking technology, Bluetooth Mesh, which will be used as this WSN’s primary technology for node communication. The discussion will go over the BLE’s Bluetooth Core Specification. This will take us through the BLE stack layers and profiles, and end with how it relates to Bluetooth Mesh. The BLE stack defines the architecture, layers, and many features of Bluetooth LE. Whereas the Profiles define the rules for using BLE technology for a particular product or application type.

In the Bluetooth Core Specification, the BLE Stack is split between two major architectural blocks known as the host and the controller. The host is a subsystem that sits below the application layer and is made up of the topmost layers of the BLE stack [4]. These topmost layers comprise multiple network and transport protocols that enable applications to communicate with peer devices in a standard way [5]. The controller is another subsystem, but it sits above the BLE radio and is made up of the bottom layers of the stack. These bottom layers are real-time protocols that provide, with the help of the radio hardware, standard-interoperable over-the-air communication [5]. The BLE’s split architecture and layers can be seen in Figure 2.

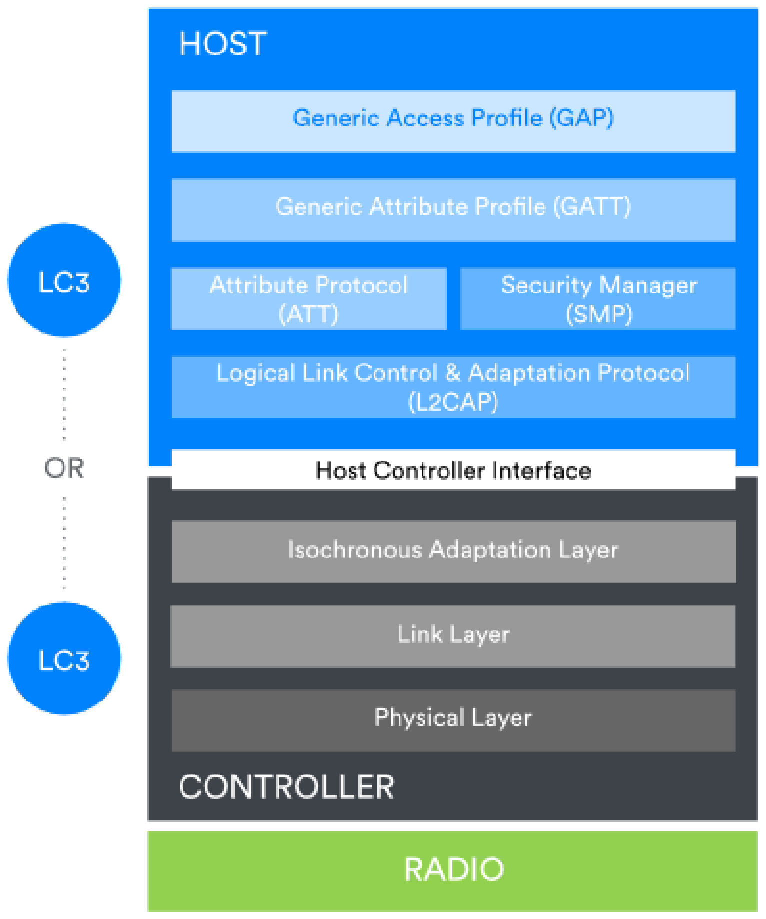
HOST
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Attribute Protocol (ATT)
Security Manager (SMP)
Logical Link Control & Adaptation Protocol (L2CAP)
Host Controller Interface
Isochronous Adaptation Layer
Link Layer
Physical Layer
CONTROLLER
RADIO
LC3
OR
LC3

Figure 2. The Bluetooth LE Stack

BLE Controller

The physical layer (PHY) is the lowest layer of the controller and BLE stack. This layer defines all Bluetooth technology aspects involving radio use, including frequency bands, modulation schemes, and transmitter and receiver characteristics [4]. As mentioned, BLE operates in the 2.4GHz ISM band, ranging from 2400 MHz to 2483.5 MHz. This range is divided into 40 separate channels, where each is 2 MHz wide [4]. The use of these channels will be discussed later in the data transport architecture in the link layer.

As for modulation schemes, these are set by the PHY to be used for both encoding digital data from higher layers of the stack and for decoding incoming radio signals from the radio layer. BLE uses a modulation scheme called gaussian frequency shift keying (GFSK) [4]. It works by shifting a signal on a central frequency channel up or down by a specific amount (the frequency deviation) to represent a digital value of one or zero. So, GFSK is used to encode digital information that will be set over the air or to decode received radio signals over the air [4].

The major transmitter and receiver characteristics are defined by the PHY variants. PHY variants change aspects of the coding and modulation schemes, which results in physical changes to the radio signal used by the radio layer [4]. At the PHY, there are three of these variants: LE 1M, LE 2M, and LE coded. They change multiple features like the protocol data rate, error control, range strength, and symbol rate. Symbol rate is the means of measuring analog radio data being sent or received from the Radio Layer and is measured in symbols per second [4]. Table 1 shows in more detail the changes that are implemented based on the use of the PHY variant.

Table 1. Comparison of the three PHY variants [4]

	LE 1M	LE Coded S = 2	LE Coded S = 8	LE 2M
Symbol Rate	1 Ms/s	1 Ms/s	1 Ms/s	2 Ms/s
Protocol Data Rate	1 Mbit/s	500 Kbit/s	125 Kbit/s	2 Mbit/s
Approximate Max. Application Data Rate	800 kbps	400 kbps	100 kbps	1400 kbps
Error Detection	CRC	CRC	CRC	CRC
Error Correction	NONE	FEC	FEC	NONE
Range Multiplier (approx.)	1	2	4	0.8
Requirement	Mandatory	Optional	Optional	Optional

The link layer (LL) will be the next and final controller layer discussed in this paper, and because of its complexity, it will not be addressed in full. This layer has many responsibilities, from defining types of packets that are transmitted over the air to a state machine that controls the function of the layer to the data transport architecture, etc. This discussion of the LL will go over packet structure, the state machine, and channel selection because these features are some of the most impactful and relevant to standard BLE functions.

The LL defines two packet types that are used to exchange information between devices. One is used for the uncoded PHYs LE 1M and LE 2M, and the other is used for the coded PHY. Both packets include the following four fields: Preamble, Access Address, CRC, and PDU [6]. The purpose of the Preamble in a packet is to allow the receiver to synchronize with the frequency of the incoming signal. The receiver also uses the Access Address; its use is to determine if the incoming signal is from a relevant device, for example, one it is currently paired with. CRC is an error detection feature that checks if the received packet may have been corrupted on its way to the device. Lastly, the PDU is used to transmit the encapsulated payload from the higher layers and can be of variable size [6].

As mentioned before, the LL's actions are governed by a state machine. There are five states that are relevant to standard BLE function; those are standby, initiating, advertising, connection, and scanning [6]. The structure of the state machine can be seen in Figure 3. In the standby state, the device radio will act idle, meaning it neither transmits nor receives packets, which will make it consume very little power. This state is also special because it’s the only one that can be switched to from any other state. In the initiating state, a device attempts to connect with a specific advertising device. Upon receiving an advertising packet from a device, the radio will send back a connection request [6].

As for the advertising state, its function is to broadcast advertising packets to announce its presence and provide its information to nearby devices. The radio will transmit these packets using channels 37, 38, and 39. The scanning state is simply the state the device will enter if it wants to listen for incoming advertising packets from other devices [6].

The connection state is correlated to both the initiating and advertising states. This is because, in the connection state, there are two important device roles defined: the central role and the peripheral role [6]. A device that initiates a connection and transitions from the initiating state to the connection state assumes the central role. A device that accepts a connection request, transitioning from the advertising state to the connection state, assumes the peripheral role. Centrals initiate connections, scan for peripherals, and receive data from peripherals, whereas the peripheral advertises its presence and shares information with the central. An example of this relationship would be a smartphone working as the central and a Bluetooth LE speaker working as the peripheral [6].

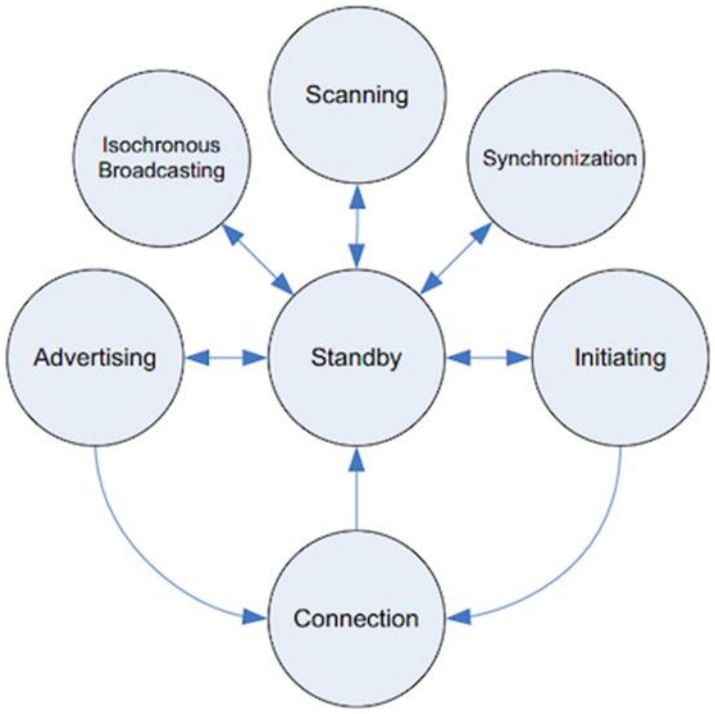

Figure 3. State diagram of the link layer state machine.

As previously stated, the LL controls channel selection, meaning which of the BLE 40 channels of the 2.4 GHz frequency band are chosen and what they are used for. There are two types of link-layer channels: advertising channels and data channels [6]. Three of the 40 channels are designed as advertising channels, and the rest are reserved as data channels. Which of the channels are selected and what they are used depends entirely on how BLE communication is being used [6].

The BLE LL employs an algorithm that makes use of data tables known as the channel map to carry out channel selection. Every channel on the channel map is categorized as either used or unused. The channel map can be updated to change a channel's classification to unused, which guarantees that the algorithm will no longer choose the channel if it is discovered that a channel is underperforming—possibly as a result of interference from outside sources. As a result, the BLE LL channel selection algorithm adjusts to the current environment and maximizes performance [6].

BLE Host

The BLE host is responsible for managing the upper layers of the protocol stack and is located right below the application layer. Regarding the host, we are most interested in its control over the profiles, such as the generic attribute profile (GATT) and the generic access profile (GAP) layers. These profiles are both central to BLE's functionality. In short, the GAP defines device discovery, connection, and communication in BLE devices, while the GATT manages how data is transferred once a connection between devices has been made [7].

The GATT is responsible for how data is formatted, packed, and exchanged between devices that have already connected. It does this by defining hierarchical data types based on attributes. Attributes are just any kind of data that the device can store and share with another [7]. GATT will group these attributes to form a characteristic, which represents a piece of information that a server wants to expose to a client. These characteristics can be further grouped again to create a service that allows a device to perform a specific functionality like a heart rate or battery service [7].

The GATT defines two different types of roles that a device can have: the GATT client and the GATT server [7]. The GATT client initiates requests to the GATT server to read, write, or subscribe to data from the server. Conversely, the GATT server stores attributes, characteristics, and services, but only when requested [7].

The GAP establishes the two mechanisms that a BLE device can use to communicate to the outside world: connecting or broadcasting. These modes of communication establish how devices running BLE make themselves available and how the two devices can create a connection [7]. These modes of communication have two further roles that a device can take depending on its function. For broadcasting, the two roles are the broadcaster and the observer. For connecting, the two roles are the peripheral and the central.

What distinguishes devices using the broadcasting mode is that they don't need to be directly connected to each other for data to be transferred between them [7]. This is because of the function of each role. The broadcaster sends public advertising data packets through the air without concern about how many devices read. On the other hand, the observer can choose to listen to the data being sent from the broadcaster in an on-demand manner [7].

In the connecting mode, both roles must explicitly be connected and handshake to transfer data between each other [7]. Regarding BLE usage, the most commonly used mode is the connecting mode. In the connecting mode, the peripheral role is that of a device that advertises its presence so that central devices can establish a connection. When a connection to a central device has been made, the peripheral will no longer advertise itself to other potential central devices [7]. Due to the peripherals being devices that only periodically advertise and only communicate when requested by a central, peripherals are usually implemented as low-power devices.

As for the central role, it’s a device that initiates a connection with a peripheral device by first listening to its incoming advertising packets and then sending a connection request. Unlike the peripheral, the central device has the option to connect to multiple peripheral devices simultaneously. Because of the aforementioned reasons, a central device is expected to be a more robust and higher battery capacity device. Lastly, when it comes to the termination of the connection between devices, either central or peripheral can perform the action [7].

Bluetooth Mesh

This section will discuss the Bluetooth Mesh (BT Mesh) networking technology in depth. Specifically, this discussion will cover fundamental concepts related to BT Mesh, like architecture, management of devices, and provisioning. Before beginning that discussion, it's important to highlight how BT Mesh incorporates BLE and why it is better suited for this WSN implementation. First, BT Mesh is closely related to BLE because it is built on top of it. BT Mesh uses the same BLE protocol stack as the lowest layer in its architecture. So, because BT Mesh uses BLE, it has many of the same functions and is governed by the same Bluetooth Core Specification rules discussed in the last section.

The two differ in their very different purposes and design characteristics. One of these key characteristics is the communication structures they employ. As described in the BLE section, BLE is designed for short-range, low-power communication between devices. It does this through one-to-one or one-to-many communication between central and peripheral devices. BT Mesh, on the other hand, is designed for large-scale networks where many devices need to communicate with each other. This is done through the implementation of many-to-many communication using the mesh topology.

The Mesh topology enables many-to-many communication by allowing devices to send to and receive from any other device in the network. This cross-network data transmission relies on a node type called relays [8]. While we'll explore relays in more detail later, they essentially facilitate data transfer by receiving data from one node and passing it to a neighboring node, ensuring it eventually reaches the desired destination.

This data-hopping functionality extends the network's effective range beyond what BLE's single-hop star topology could achieve and offers multiple routes for data to reach its destination. Together, these features provide the network with scalability and reliability. Due to its capacity to support large numbers of nodes, extend range via multi-hop routing, and ensure reliable data transfer for large systems, BT Mesh was selected as the network topology for this proof of concept.

Fundamental Workings of Bluetooth Mesh

BT Mesh networks are made up of devices called nodes. Each node sends and receives information that can be relayed from node to node, creating a web of nodes, as can be seen in Figure 4 [9]. This web of nodes can be used to span large areas like manufacturing facilities, environmental areas, office buildings, etc. The nodes themselves can also have a variety of different functions within the network. Some examples of BT Mesh nodes could be light fixtures, machinery sensors, security cameras, environmental sensors, and much more [9].

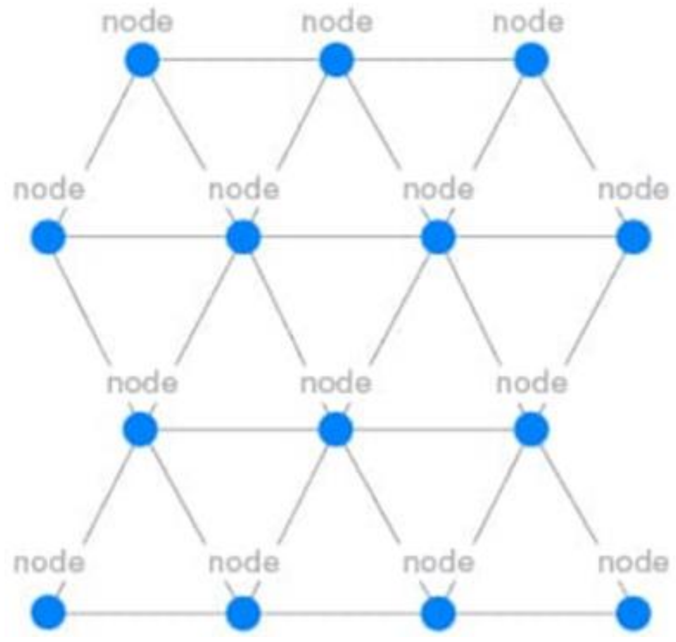

Figure 4. Nodes in a mesh network [9]

Because nodes in a mesh network can serve a wide variety of functions and can vary significantly in aspects like processing power and battery capacity, BT Mesh defines four distinct node roles. These roles are Relay, Proxy, Low Power, and Friend nodes, each with a specialized function in the network [9]. For example, the Low Power node (LPN) operates at a significantly reduced power level by minimizing its receiver duty cycle [9]. Since nodes must be constantly on and listening for messages from other nodes, reducing the time the radio receiver is active lowers power consumption. This low-power feature is commonly used for nodes relying on small batteries.

The LPN still needs to be able to receive messages when its radio is not on, just in case a request for it comes in; for this reason, it works in conjunction with the friend node. The friend node will form a friendship with the LPN and store incoming messages and security updates destined for the LPN [9]. When the LPN turns back on its radio, it will poll its friend node for any stored messages. Because the friend node must always be on and store data for an LPN, it's expected that the friend node should be a non-power-constrained device.

The relay node is another nonpower-constrained device that receives and retransmit messages [9]. This node is a fundamental requirement of the mesh network because relaying is the mechanism that is utilized for messages to traverse the entire mesh network by making multiple “hops” between devices till they reach their destination [9]. Although relaying is required for the network, the feature is optional for individual nodes.

The final node type is the proxy node. The proxy node enables transmitting and receiving mesh messages between GATT/GAP and BT Mesh nodes [11]. It uses the proxy protocol to translate between BT Mesh PDUs and proxy protocol PDUs. So, it acts as an intermediary, allowing devices that support BLE but don’t support BT Mesh to communicate with the network [9]. This is, again, another device that should have high processing and power capacity.

Another essential concept in BT Mesh is that of elements. Some nodes in a BT Mesh network may consist of multiple independent parts within a single device. In BT Mesh, elements represent these independent parts [9]. Each node must have at least one element, known as the primary element, and may include additional elements as needed [9]. Elements are composed of entities that define the node’s functionality and the condition of each element. For example, a light bulb might have one element with two functions: turning it on/off and adjusting its brightness. Each element within a node has a unique address, called a unicast address, making each element individually addressable—more on this will be discussed later.

A model is the functionality that makes up a node’s elements, which we previously discussed. So, models are the inner works of elements that define and implement the functional behavior of a node. The models are composed of states, which are the values or positions the model element can be in [10]. So, using the prior example, the light bulb’s on/off model can either be in the on state or in the off state, while the brightness model can be in a state between its lowest and maximum possible light intensity. Figure 5 gives a visual explanation of the mesh node structure just discussed.

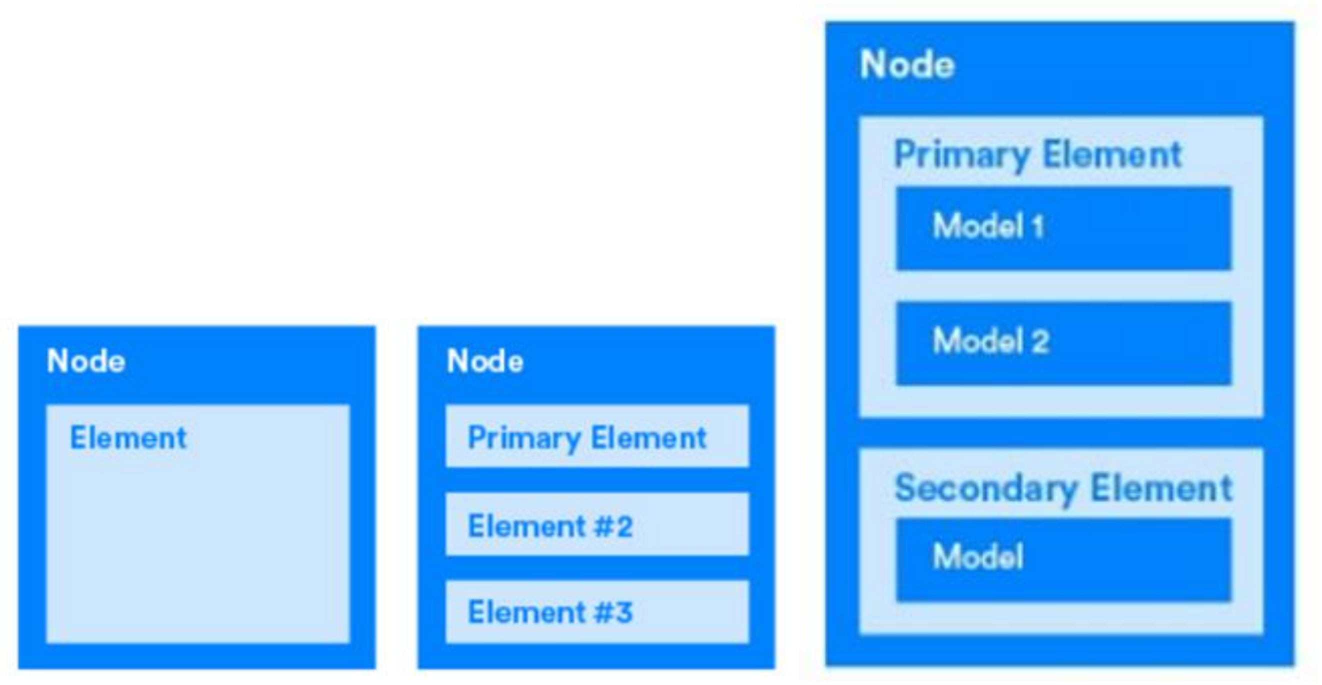

Figure 5. Structure of a mesh node using elements and models.

In a BT Mesh network, the models are split into a client-server architecture similar to BLE. The server model defines the messages that can be sent and received and can show or change the states of its model at the request of a client model [10]. The client defines the set of messages that can be used to request, change, or consume the states of a server [10]. There is also a third model type, called the control model, which combines the functionality of both client and server models; however, it is not covered further here, as it was not used in this implementation.

In BT Mesh, devices communicate with each other by sending to and receiving from addresses. Addresses are used to add a unique identifier to each element in the network. Three types of addresses are used for working with BT Mesh devices: unicast, virtual, and group addresses [10]. There is a fourth address type known as an unassigned address, but this type of address is only used for unconfigured elements or elements that don't use a designated address. Regardless, this type of addressing cannot be used in messaging of any kind.

As for unicast addressing, as the name implies, it's a unique address for an individual element. This unique address is assigned to each element in the BT Mesh network. This address can be used as either a source or destination address field when messaging. This address is given to a device by the provisioner during the provisioning process, which will be discussed later. The address given stays with the device for its lifetime on the network [10].

Virtual addressing devices are hashed together and given a 128-bit UUID to represent that hashed group of devices [11]. The given virtual address can be published to or subscribed to by other devices. Group addressing is like virtual addressing in that it's another type of multicast addressing. However, a group address is just a collection of devices that can all be addressed simultaneously. So, if a message is sent to a group address, all devices with that group address will receive that message. All the addressing types discussed are 16 bits long and follow the encoding format in Table 2.

Table 2. Mesh address encoding

Values	Address Type
0b0000000000000000	Unassigned Address
0b0xxxxxxxxxxxxxxx	Unicast Address
0b10xxxxxxxxxxxxxx	Virtual Group
0b11xxxxxxxxxxxxxx	Group Address

In BT Mesh, the messages can take on a few different forms. The two primary forms of messages are the control messages and access messages. The control messages are messages that send data relevant to the operation of the network [10]. Some examples of these kinds of messages would be a friend request message trying to connect with an LPN or a heartbeat message that indicates that it's still active to other nodes in the network. On the other hand, access messages, depending on whether it's a server or client using them, are messages that report their state values or can set or receive state values [10]. Some examples of access messages would be the sensor_get message, which gets the sensor value from a sensor server, or the sensor_settings_set message, which sets the value of a setting for a sensor on a sensor server.

These messages are exchanged through a publish/subscribe model. This means that devices that want to send messages will publish to a specific address like a unicast or group address, and devices that wish to receive messages will have them sent to their unicast address or have to subscribe to a group address. Due to relaying and managed flooding, many nodes may receive messages that are not destined for themselves. Nodes deal with this by using their models' subscription lists when receiving a message; it will use its model's subscription lists to see if any of its models are subscribed to the address to which the incoming message is published.

Provisioning How Bluetooth Mesh Network Are Created

Up to this point, the discussion of BT Mesh has been focused on fundamental concepts of BT Mesh, like the different roles of nodes in the network and how data is transmitted from one device to another. However, the idea of how devices go from being outside the network to being a part of it is lacking. So, this subsection will conclude the discussion of BT Mesh networking by showing how these networks add new nodes. This process is known as Provisioning.

Provisioning is the process of adding new devices, called unprovisioned devices, to the BT Mesh network. For an unprovisioned device to be added, it communicates with a Provisioner, which is a device dedicated to bringing these new devices into the network [12]. Both devices then go through the provisioning process by transmitting the necessary provisioning data back and forth.

The Provisioning process begins with a beacon phase [12]. This phase works much the same as the broadcasting role discussed during the explanation of the GAP layer of the BLE stack. However, rather than peripheral devices broadcasting to scanning central devices through GAP, unprovisioned devices advertise through either the PB-ADV bearer or the PB-GATT bearer [12]. Afterward, the provisioner discovers and picks which device to invite to the network using the same BLE scanning procedure. After the Provisioner and unprovisioned device establish a connection through either PB-ADV or PB-GATT, the provisioner sends an invite PDU, and the device responds with its provisioning capabilities PDU.

The next step in the provisioning process is the exchange of public keys between the provisioner and the unprovisioned device [12]. This step is done to create a secure communication link between the two devices to enable encryption that will protect data exchange during and after the provisioning process is finished. In BT Mesh, these keys are exchanged either through an out-of-band (OOB) channel or directly over BLE [12].

After keys have been exchanged, the provisioner uses the following three authentication methods to begin the authentication process for the unprovisioned device: output OOB, input OOB, and static OOB [13]. In output OOB authentication, the unprovisioned device will output some random sequence, like LED flashes or a number displayed on an LCD. Lastly, the provisioner must input this sequence to perform authentication with the device. The input OOB authentication works similarly but in the opposite way. The provisioner will supply a random number that must be inputted into the unprovisioned device. A common way of inputting this number is through button presses on the device. After inputting this number, a provisioning protocol PDU must be returned to the provisioner for verification.

After an authentication method is picked, confirmation value generation and checking occur. In this process, the provisioner and the unprovisioned device generate their own random numbers called the ConfirmationProvisioner and ConfirmationDevice [13]. These random numbers are used as inputs together with the authentication value and encryption key in several rounds of AES-CMAC and SALT to generate the confirmation value [13]. Both devices then exchange these confirmation values, compute the other’s confirmation value independently, and check it against the received value. If they are a match, that means the authentication step has been completed, and provisioning data can be shared with each other [13]. The abovementioned procedures and steps for configuration value generation are shown in Figure 6.

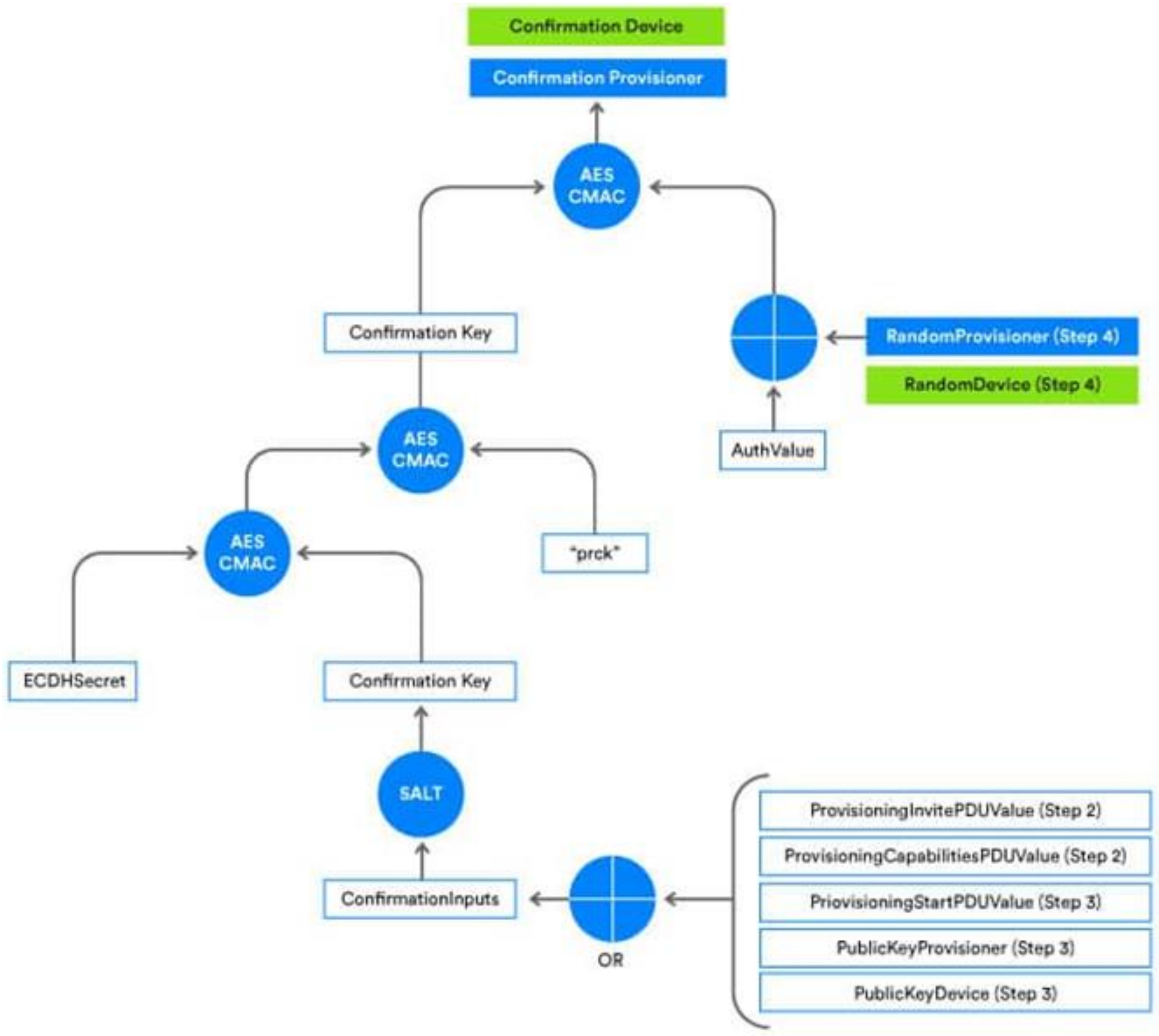

Figure 6. Confirmation value generation [13]

During the Provisioning data exchange, the provisioner is responsible for sharing and generating the provisioning data for the soon-to-be BT Mesh node [13]. One of these is the network key, which the provisioner generates when it provisions the first device to be added to the network. This key is the primary encryption key that all devices in the network share and is what defines membership in the given BT Mesh network. The following data exchanged is the device key, which is unique to each device and is used for communication between itself and the provisioner. Its purpose is to be an encryption key between the provisioner and itself. The provisioner also sends the IV index during this exchange. The purpose of the IV index is to provide randomness in calculating security messages, which helps prevent replay attacks when communicating these messages across the network [11]. Flags are another critical piece of data exchanged in this process. Flags provide configuration information about the network status, like the key refresh and IV updates. Lastly, in this process, the unicast address is given to the newly provisioned device, which identifies the device within the mesh network

Distance Measurement Physics

This subsection will discuss how distance measurements can be performed between two devices using wireless communication like Bluetooth and will go over the Nordic distance measurement toolbox (NDT), which was used to help implement localization in this paper's proof of concept. To begin, there are several techniques for performing distance measurements between devices within radio range of each other. Some of the most common of these techniques and the ones implemented in the NDT are received signal strength indicator (RSSI), time-of-flight (ToF), Multi-carrier phase slope estimation (MCPD), and inverse fast Fourier transform (IFFT) [14].

Received Signal Strength Indicator (RSSI)

The RSSI technique is a simple but cost-effective technique for distance estimation. As the name implies, this technique revolves around measuring the signal strength of the incoming transmission of the device it wants to perform a distance measurement with [14]. Signals begin to weaken and fade as they travel through the air; the higher the RSSI level, the stronger the radio signal, and the stronger the radio signal, the closer the devices are to each other [16]. So, on the transmitting side, the short-range radio will broadcast a 2.4 to 2.5 GHz signal; this will form a sphere expanding at the speed of light. As the sphere propagates outward in a clear space with no obstacles, its intensity will drop off according to the simple equation you can see in Equation 1 [17].

$$recieved\ power \approx 1/r^2 \qquad (1)$$

Equation 1 can be used on the assumption that both the transmission power and incoming signal strength are known by the receiver. Using this received power equation, the receiving device can estimate how far it has traveled from the transmitting device. However, the above equation is also under the assumption of a clear space with no obstacles obstructing the signal's path, and this is not realistic in most implementation scenarios. The intensity in a building or house will drop off much faster [17]. For this reason, Equation 2 improves distance measurement accuracy by considering the path loss factor, n.

$$recieved\ power\ \approx 1/r^n \tag{2}$$

The path loss factor n is based on measurements taken in specific environments. For example, the path loss factor for homes made of wood is around 4.5 [14]. So, if you don't know the exact space you'll be operating in, you will have to estimate which path loss factor to use, leading to more variation in the calculated distance. RSSI's advantages come in the form of an inexpensive and straightforward way to estimate distance, and it has been found to perform well in short distance measurement ranges of a few meters. The downsides come in its path loss factor, ineffectiveness at long ranges, and inaccuracy when people move through its measurement area.

Time-of-Flight (ToF)

ToF is another distance measurement technique that brings greater precision. Rather than measuring signal strength at the receiving end, this technique measures the time it takes for a transmitted signal to travel from itself to a receiving device and back again [16]. So that means that in this process, the transmitting device will propagate a signal, and the receiving device will act as a mirror and reflect the signal. Once the transmitting device receives the signal, it will calculate the distance based on the speed of light, as can be seen in Equation 3 [16].

$$distance\ to\ object\ (meters) = (\frac{ToF(seonds)}{2} * c(meters\ per\ second)) \tag{3}$$

Something to note about this equation is that it doesn't take into account the extra delay that is added to the ToF when the receiving device is switching from receiving the signal to transmitting/mirroring it back. So, for this calculation to be more accurate, especially at shorter distance measurements, this switching constant delay needs to be found and subtracted from the ToF in the equation [14]. Lastly, the round-trip timing is measured over multiple frequencies, and the average value from these results is the one used in the final calculation.

Multi-Carrier Phase Slope Estimation (MCPD)

The third measuring technique uses phase-based metrics to determine distance. The mathematics behind phase-based distance measurement can be quite complex, so rather than delving into equations, let’s explore the concept through its physical properties. This technique operates similarly to Time of Flight (ToF) but differs in that it collects returning phase shifts across multiple frequencies and compares them to detect correlations that correspond to a specific distance.

For example, as Table 3 shows, a 2.4 GHz signal has a wavelength of 0.125 meters. When sent out, the signal rotates 16 times and returns to the transmitter with a phase of 0. Afterward, the transmitter sends additional signals at different frequencies, such as 2.440 GHz and 2.480 GHz. These signals complete 16.26 and 16.53 rotations and return with phases of 0.52π and 1.06π. The phase-based algorithm then analyzes these phase readings and identifies that they correlate with a distance of two meters [18].

Table 3. Phase-based characteristics of different frequencies [16].

Frequency [GHz]	Wavelength [m]	Number of rotations	Remainder rotations	Phase of reflected signal
2.400	0.125	2/0.125=16	0	0
2.440	0.123	2/0.123=16.26	0.26	2π x 0.26 = 0.52π
2.480	0.121	2/0.121=16.53	0.53	2π x 0.53 = 1.06π

Inverse Fast Fourier Transform (IFFT)

The final distance measurement technique we’ll discuss, and the one used for this implementation, is the IFFT. Although IFFT is not a distance measurement technique by itself, it is a signal processing algorithm that can be used alongside other methods, like Time of Flight (ToF), to enhance performance. To use IFFT for ToF distance measurement, the channel frequency response

(CFR) is first measured. The CFR indicates how each frequency component of a transmitted signal is affected by its path from transmitter to receiver. To obtain the CFR, a series of signals is transmitted and collected by the receiver, then processed using the Fast Fourier Transform (FFT) to isolate each frequency's amplitude and phase [14].

After obtaining the CFR, IFFT transforms the frequency data into the time domain, resulting in the channel impulse response (CIR). Each peak in the CIR corresponds to a distinct path taken by the signal to reach the receiver, with the first peak generally representing the line-of-sight path and later peaks indicating reflections [14]. This separation is advantageous for ToF because IFFT can distinguish direct signals from reflected ones. By identifying the first major peak in the CIR as the line-of-sight path, the system can ignore later peaks caused by reflections, thereby improving the accuracy of the distance measurement.

CHAPTER 3

IMPLEMENTATION, PEER-TO-PEER, AND RESULTS

With the preliminary information about WSNs, BT Mesh, and distance measurement covered, this chapter will focus on implementing a location-based WSN using these technologies. The chapter begins with an overview of the hardware: what components were used, how they function, and what function they served in the network. Following that, the next section discusses the software setup, including the programming environment, mesh models used to create the system, and the approach for programming elements and model functionality into the devices.

Subsequent sections delve into the implementation details of the one-hop distance measurement algorithm and the general data flow through the system. The final section presents the system's results, focusing on the best and worst-case scenarios for the client and sensor positions, highlighting the conditions leading to the system's lowest and highest error rates.

Hardware Setup and Implementation

This design incorporates two primary pieces of hardware to form the network: Nordic Semiconductor's nRF5340 Development Kit (nRF53 DK) and the Nordic Thingy 53. The nRF53 DK contains all the previously discussed components to function as a proper node in a WSN. Its processing capabilities come from the nRF5340 System on Chip (SoC), which includes two separate Arm Cortex-M33 processors [19]. One serves as the microcontroller's application processor, while the other handles network processing. The application core manages all application firmware, provides computational power, and interfaces with internal peripherals such as I2C, SPI, and USB.

The network core, meanwhile, is responsible for all network-related tasks, including handling the radio and supporting wireless communication protocols. The nRF53 DK also features a low-power 2.4 GHz antenna for radio communication [19]. Regarding power options, the DK can use a 1.7-5.0 V supply from USB, a Li-Po battery, or a generic two-pin connection for other external batteries.

Additionally, it has a coin cell battery slot for a CR2032 battery, which was used to power the devices in this paper's implementation.

The other primary piece of hardware in the WSN was the Thingy 53. The shared "53" in their names is no coincidence; the Thingy 53 also uses the nRF5340 SoC and, therefore, shares many of the same features as the nRF53 DK. However, the Thingy 53 is designed as an IoT prototyping platform, meaning it comes equipped with a range of onboard sensors that can be readily interfaced with. The sensors that come equipped with the board are an environmental sensor, color/light sensor, and many more [20].

This WSN setup consists of six nodes: five nRF53 DKs and one Thingy 53. The nRF53 DKs serve three different functions within the network. One of the DKs acts as a mesh client device. While its functionality will be discussed in more detail later, the client node essentially sends out various requests using the four onboard push buttons. The first button on the DK initiates P2P distance measurement requests, which will later be used in the one-hop distance measurement algorithm.

The remaining three buttons are used to request different sensor readings from the sensor node. Specifically, button two sends a request for the current ambient temperature reading, button three requests the current humidity reading, and button four requests the sensor's illuminance level in the green spectrum. This setup enables the client node to gather a range of environmental data from the network's sensor node.

Although the sensor is capable of reading red, blue, and green spectrums of visible light, this design focuses on green light readings. Green light falls in the middle of the visible light spectrum and is the easiest for the human eye to detect. Therefore, by measuring green light, the system can serve as a visible light sensor, providing an indication of ambient brightness levels.

The second function of the nRF53 DK is to function as the anchor nodes of the network. In this design, three DKs were used as anchor nodes. In the literature, anchor nodes commonly refer to nodes whose location is known and are used to determine the location of other nodes in the WSN. In this design, anchor nodes have the latter functionality but not the former. Anchor nodes in this design are used to help get the distance measurements required for the one-hop algorithm. They will be explained more later, but these nodes are placed in a straight-line configuration in front of the client, and the sensor will be located behind this line of anchors.

The final role for the nRF53 DK in this setup was as a friend node for the Thingy 53 sensor node. As discussed previously, friend nodes store messages for low-power nodes while their radios are off to conserve energy. This friend node configuration allows the system to obtain both distance measurements relative to other nodes in the network and sensor readings from the Thingy 53, even while it operates in low-power mode. To enable the Thingy 53 to function as a low-power node, implementing a friend node was essential.

The Thingy 53, in turn, served as the WSN’s low-power sensor node. This device was selected for its onboard environmental and color sensors. Specifically, the BME688 environmental sensor provided ambient temperature and humidity readings, while the BH1749NUC color sensor measured illuminance in the green spectrum. Both sensors are directly connected to the I2C bus, so I2C is required for interfacing. The software implementation and setup section will discuss details on using I2C to communicate with these sensors. Both the BME688 and BH1749NUC circuit schematic are given in Figure 7.

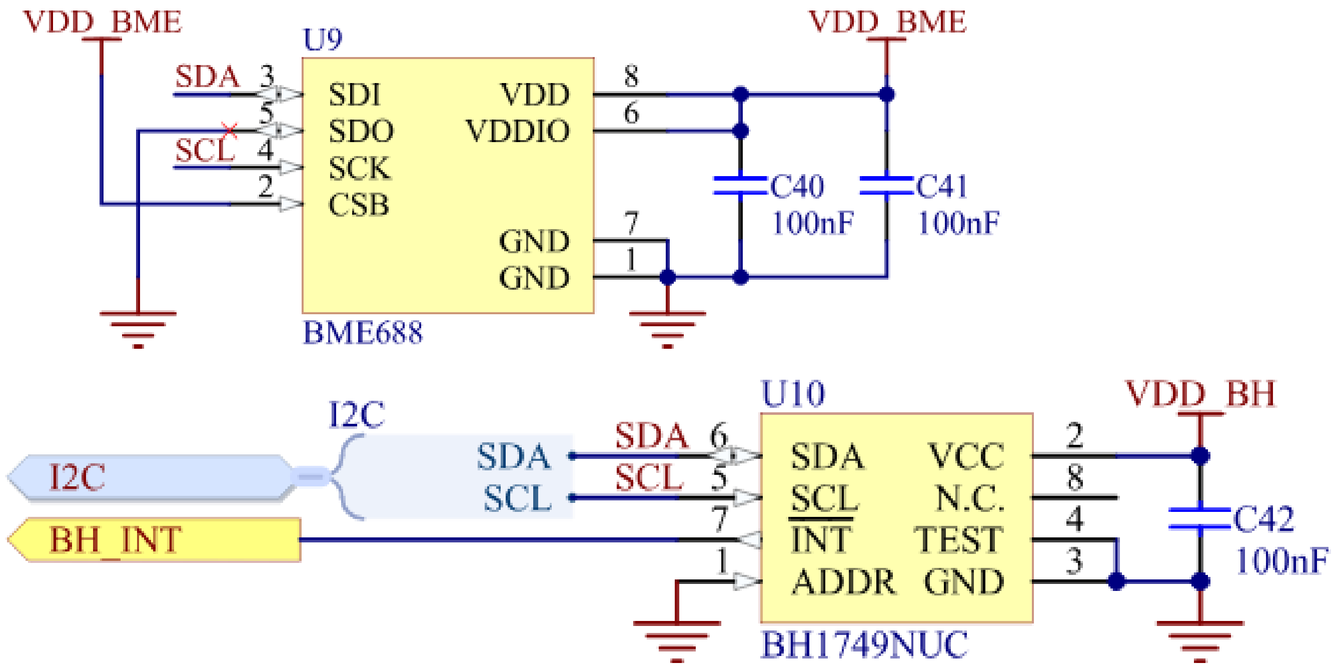

Figure 7. The Thingy 53 Environmental and Color Sensor [20]

Software Setup and Implementation

In order to program the hardware that was just discussed, Nordic Semiconductor offers the nRF Connect SDK. This is a software development kit specifically made for building and creating projects for their wireless devices. The environment provides a single code base for all the devices and software components [21]. This section will discuss the process of how to configure nodes in the mesh network as well as how the devices were programmed to implement features like distance measurement and sensor reading using this SDK. As well as how to put nodes into low-power mode and use I2C for interfacing with the Thingy 53's onboard sensors.

To begin, in the nRF Connect SDK, enabling Bluetooth should be the first procedure that takes place when programming a device to be a BT Mesh node. This is required because otherwise, the use of local Bluetooth hardware and APIs will be unavailable. After this, it's common to initialize your device's onboard LEDs and push buttons. The reason for this is that both LEDs and push buttons are used for provisioning authentication options like input OOB and output OOB and for the attention feature, which is a valuable feature that is called during the provisioning process that visually shows which device is about to start the provisioning process.

The next step, and where devices begin to diverge when it comes to programming them as nodes within the BT Mesh network, is when initializing Mesh support. Every mesh node must initialize this support, as it sets up the elements and models that define the node’s functionality and activates the provisioning handler. During this initialization, the node’s model handler is passed in. This model handler essentially defines the node’s composition, containing pointers to critical features such as the structure defining the node’s elements and essential callback functions, like the button handler (triggered when a button is pressed) and the attention callback (activated during authentication).

For the client node specifically, its model handler is set up to include the button callback function, attention handling function, and the node’s element composition. This element composition is organized as an array of C structures, with each array entry representing an element of the node. In the client node’s case, it consists of one primary element and one secondary element. The primary element includes four models: the configuration server, health server, sensor client, and distance measurement client. The secondary element contains only one model, the distance measurement server.

The configuration server model is a mandatory model for all BT Mesh nodes. Its purpose is to configure the device for mesh network device features like broadcasting as a secure network beacon, or to indicate that it is in the state to preform friend node features. The health server model is also a mandatory model for all devices. It is used to monitor and report on any faults or errors the device may have experience throughout its lifetime. These diagnostics can be read by other devices in the network using the health client model.

The next model used in the client node’s primary element is the sensor client. The sensor client’s role is to send “get” requests to the sensor server on the Thingy 53 and process the data returned by the server. As mentioned earlier, these requests are triggered by button presses on the client device. Each button press calls a globally defined function that sends a specific sensor reading request based on the button pressed. These requests use APIs defined by the BT Mesh sensor client model.

Specifically, this implementation employs the bt_mesh_sensor_cli_get function, which encapsulates a request based on the sensor instance available on the targeted sensor server. When data is received from the server, a callback function on the node’s sensor client model is triggered. This callback reads the sensor ID value in the data to identify the type of sensor reading. Once identified, it displays the reading on the terminal for monitoring.

The last model used in the client node’s primary element is the distance measurement client model. This model’s role is to request distance measurement server models to perform measurements between each other. As with other functions, this request is initiated by pressing a button on the client device, which triggers a global function for distance measurement. This function utilizes two primary APIs: bt_mesh_dm_cli_config and bt_mesh_dm_cli_measurement_start.

The bt_mesh_dm_cli_config function first sends a configuration message to set the parameters that the distance measurement server will use for its measurement. Following this, the bt_mesh_dm_cli_measurement_start function sends a start message to initiate the measurement process. This API also specifies parameters for the server, including which distance measurement server to connect to and the ranging mode to use.

The distance measurement server model is the final model used in the client node, located in its secondary element. This model is critical in the network, as it enables the client to perform distance measurements and serves as the foundational model that allows anchor nodes and the friend node to carry out distance measurements. As mentioned previously, a distance measurement procedure begins when a bt_mesh_dm_cli_measurement_start message is issued.

When Distance Measurement Server 1 (DM SRV 1) receives the start message, it initiates a timeout timer based on either the default timeout or the timeout specified in the start message. Next, it takes on the reflector role, applying the ranging mode defined in the start message, and issues a synchronization message to the second distance measurement server (DM SRV 2). Upon receiving this synchronization message, DM SRV 2 starts its own timeout timer and initiates the initiator role using the same ranging mode. If the ranging procedure is completed successfully, both DM servers store the measurement result, and DM SRV 1 sends the measured data back to the distance measurement client that initiated the process. The full procedure is illustrated in Figure 8.

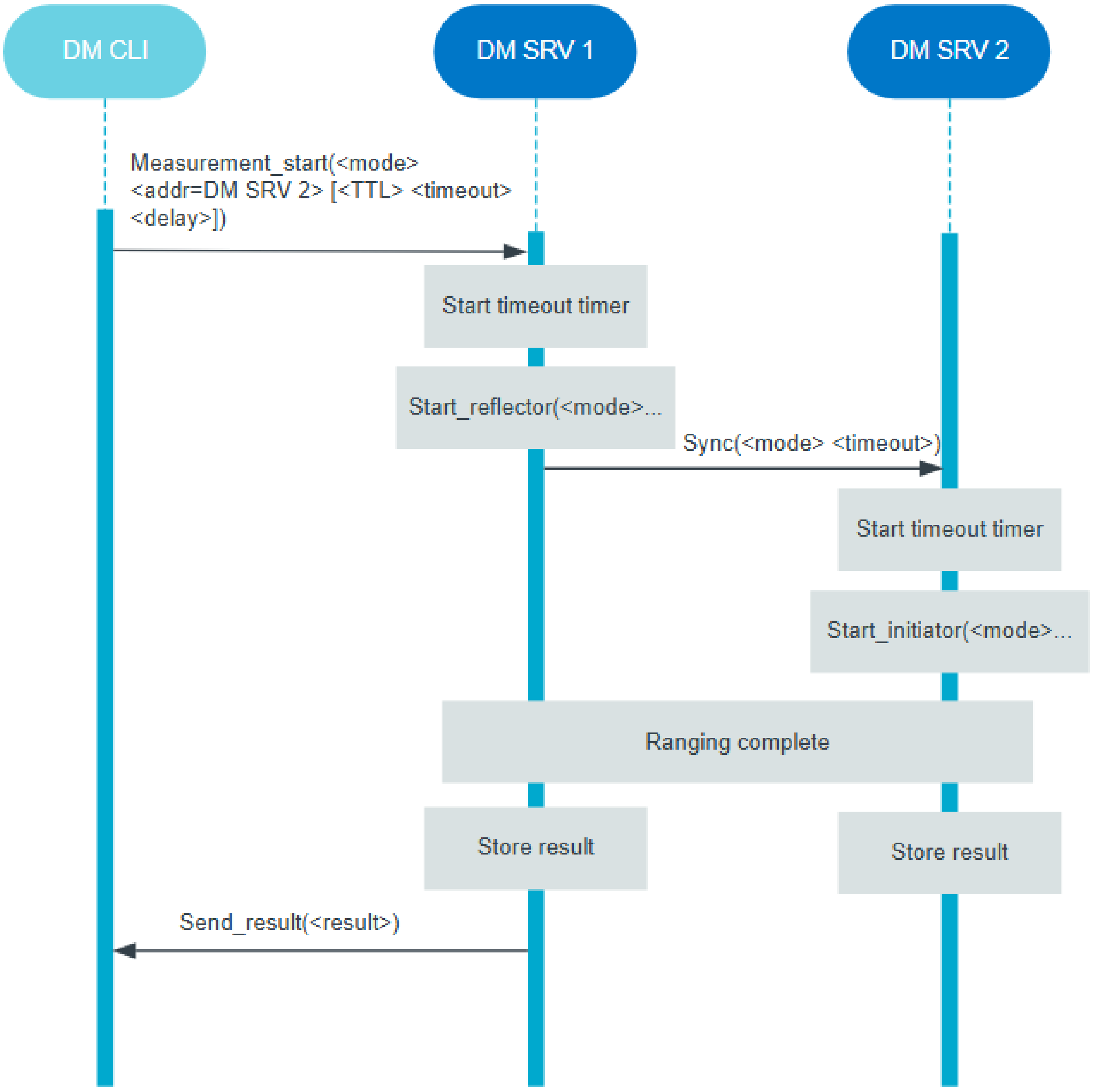

Figure 8. Distance measurement procedure

Before detailing the final procedures for programming and configuring the node as a BT Mesh device, it’s essential to cover the use of configuration files and device tree files, as these are crucial for adding functionalities like low-power mode and enabling the I2C protocol within the nRF Connect SDK. Each application in the SDK requires an application configuration file. These files are text-based rather than written in a conventional programming language, and they contain configuration options called symbols, which are usually assigned values of y (yes), n (no), or specific numerical values. The configuration file defines the software modules, kernel services, and settings used by the application [22].

In this implementation, configuration files were used to manage general settings, such as enabling specific drivers and memory allocations. They also handled Bluetooth configuration, allowing for tasks like setting the device name and adding support for roles such as observer and peripheral. Additionally, these files managed BT Mesh configurations by enabling BT Mesh support and configuring node roles such as relay and low-power node. Finally, the configuration files enabled BT Mesh models, including the distance measurement and sensor models needed for this implementation.

The device tree is a hierarchical data structure used in the nRF Connect SDK to describe and configure hardware. Device tree files define hardware details, set initial configurations, and adjust hardware-related settings within an application [22]. This means these files can configure internal peripherals like GPIO, UART, and I2C, which are used here for connecting to the Thingy 53’s onboard sensors. Unlike conventional programming languages, device tree files use a specific syntax known as the device tree language.

In the device tree language, nodes are called and modified by setting their properties within an existing base device tree. For example, to configure I2C, you would call the I2C node, define its pin control functionality, and declare the sensors to be interfaced with—in this case, the BME688 and BH1749 sensors [22].

To complete the software implementation, once BT Mesh initialization is finalized, the next step is to check whether low-power functionality is required for the node. Following this, the settings subsystem configuration is reviewed to confirm it's enabled. The settings subsystem supports saving and restoring the node’s state by serializing data into non-volatile memory and retrieving it as needed. When enabled, any items marked for persistence are automatically loaded from memory upon startup.

The final step is to activate the specific provisioning bearers that the node will use. For all nodes in this network, both PB-ADV and PB-GATT bearers were enabled, supporting provisioning over both advertising and GATT.

One-Hop Distance Measurement Algorithm

When devices within radio range want to get the distance measurement between each other, there are many ways to do this, as has been explained before by using distance measurement techniques like RSSI, TOF, and IFFT. This section will discuss how this implementation takes this one step further and estimates the distance between devices that are one hop away from the other device. This section will explain in detail how this system was able to implement one-hop localization in the BT Mesh Network using P2P distance measurements.

P2P localization is a technique that uses the devices themselves to locate each other rather than relying on a central server or pre-deployed infrastructure for localization like GPS [23]. In this paper’s implementation, nodes perform P2P localization by solely relying on one-to-one distance measurements using the IFFT distance measurement technique that was previously discussed. Specifically, the client node and the sensor’s friend node perform IFFT distance measurements with the network’s anchor nodes.

The positions of the anchor nodes, client nodes, and sensor nodes within the network are crucial for accurately executing the one-hop distance measurement algorithm. First, the anchor nodes are arranged in a horizontal line, evenly spaced from each other. The client node is placed on one side of this line, while the sensor node and its friend node are positioned on the opposite side. Both the client and sensor nodes can move freely within their designated side of the line as long as they remain within radio range of at least two of the three anchor nodes.

To perform the distance measurements, the client node initiates measurements to each anchor node using the onboard push button one on the development kit. As described previously, this is possible due to the client's use of the distance measurement client and server models. Each button press initiates a round of distance measurement requests, beginning with the leftmost anchor and progressing to the rightmost. Once each measurement is complete, the result is sent back to the client, stored in its software, and a flag is raised. After three measurements, a fourth button press will trigger distance measurement between the anchors and the sensor's friend node. When measurements between the friend node and all anchors are completed, all six measurement flags are raised within the client node's software, initiating the distance measurement algorithm.

The distance measurement algorithm begins by identifying the two closest anchor nodes to both the client and sensor. Since the client and sensor nodes are mobile, this initial step helps narrow down their positions relative to the anchors. The algorithm then checks how many anchor nodes are shared between the client and the sensor. This step is essential because it allows the algorithm to identify the most suitable anchor nodes to serve as reference points for accurate distance measurements. Since this system relies on point-to-point measurements rather than angles, it seeks to establish a direct line of measurement from the client to an anchor and then to the sensor—an approach that has proven to be both simple and reliable for one-hop distance estimation.

After determining the number of shared anchor nodes, the software evaluates the best reference point. If the client and sensor share only one anchor node, that anchor is used as the single reference point. The distance calculation is then straightforward: the software retrieves the stored distance from the client to the anchor node and adds it to the stored distance from the anchor to the sensor. If they share two anchor nodes, an additional check determines which of these anchors is closest to either the client or the sensor, using the nearest anchor as the reference point. The final distance measurement is then calculated in the same way as before. Figure 9 illustrates the general structure of this network setup.

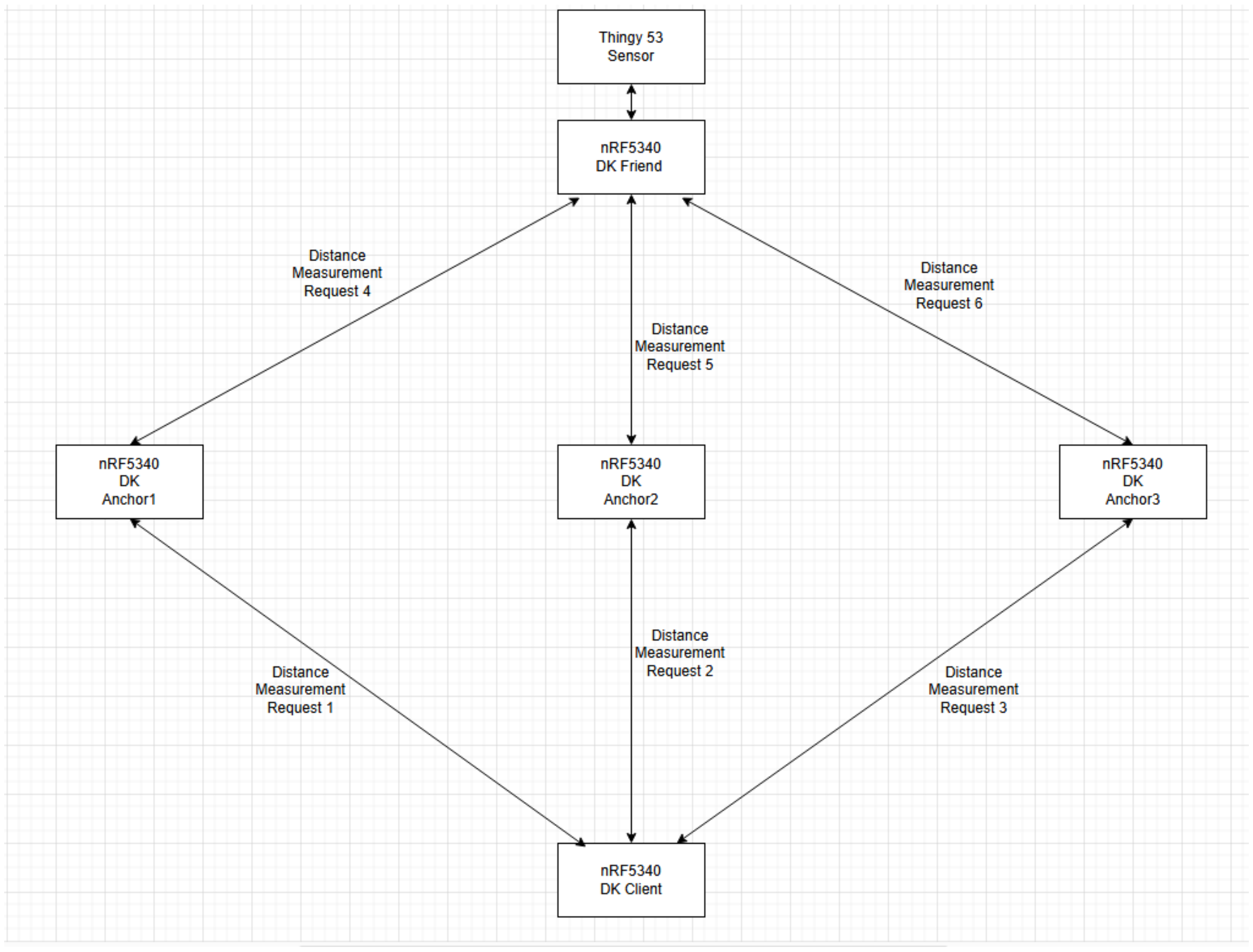

Figure 9. The structure of the location-based WSN

Results

This section discusses the results of this proof of concept, focusing on the WSN's best- and worst-case distance estimations, including percentage error. There are numerous configurations and positions that the client, sensor, and anchor nodes can occupy, each potentially leading to different results. Here, I'll review three specific scenarios to illustrate these outcomes.

The first scenario is a configuration similar to Figure 9, where the client and sensor have a clear line of reference using anchor node two. The second scenario places the sensor node behind, but between, anchors two and three, with the client positioned similarly to Figure 9. In the final configuration, the client and sensor have a direct line of sight to each other and are positioned between anchors two and three.

These measurements were gathered through practical testing conducted in my backyard. For estimating distances, I used the IFFT distance measurement technique between the nodes, while the actual distances were measured with a LiDAR-based laser measurement tool. Although this setup is not a controlled lab experiment, it provides a solid demonstration of the concept.

The results of the first scenario are shown in Table 4. Because there is a clear reference path from the client to the sensor, this represents the network's best-case scenario. As seen in the table, the percent error fluctuates, with the highest error occurring between anchor two and the sensor. This high error was included to illustrate that the IFFT measurement technique can sometimes yield significant inaccuracies, which should be taken into account. However, most of the other measurements exhibit very low error values. The percent error from the client to the sensor is relatively high, but this is due to the compounded error from the anchor 2-to-sensor estimate used in the final calculation.

Table 4. Results of first distance measurement test scenario

	Estimated Distance (M)	Actual Distance (M)	Percent Error (%)
Client to Anchor 1	7.54	7.166	5.22
Client to Anchor 2	6.47	6.518	0.74
Client to Anchor 3	6.83	6.533	4.55
Anchor 1 to Sensor	5.56	5.214	6.64
Anchor 2 to Sensor	5.81	3.869	50.16
Anchor 3 to Sensor	5.83	5.341	9.15
Client to Sensor	12.28	10.189	20.52

The results of the second scenario are shown in Table 5. In this configuration, both the client and sensor share anchors 2 and 3 as their closest nodes. However, anchor three was chosen as the reference point because it is the closest to the client based on relative proximity. Unlike the first scenario, this one does not contain a significant outlier in percentage error, which likely contributes to the close match between the estimated and actual distances.

Although this setup produces minimal error, it is not considered an ideal scenario due to the slight angle of the sensor, which introduces a small amount of error in the final calculation. Nonetheless, the error remains manageable and demonstrates the reliability of the distance estimation in less-than-ideal conditions.

Table 5. Results of second distance measurement test scenario.

	Estimated Distance (M)	Actual Distance (M)	Percent Error (%)
Client to Anchor 1	8.08	7.65	5.62
Client to Anchor 2	6.92	6.635	4.23
Client to Anchor 3	6.57	6.859	2.76
Anchor 1 to Sensor	10.03	9.335	7.45
Anchor 2 to Sensor	7.41	7.15	3.64
Anchor 3 to Sensor	7.69	6.87	11.94
Client to Sensor	14.26	13.452	6.01

The results of the final scenario are shown in Table 6. This scenario represents one of the worst cases for the WSN. Unlike scenario one, which had a direct line of sight, or scenario two, which had a partial line of sight with the sensor at an angle, this scenario places the client and sensor facing each other between their potential reference points, anchor two and anchor 3. This positioning introduces additional error due to the added angles, which are not accounted for in the final distance calculation.

Although this scenario shows higher error in each measurement, it still ended in a final estimated distance of approximately double the error percentage compared to scenario two. So, these results demonstrate how factors like node positioning and indirect reference points impact the accuracy of the distance measurement in this wireless sensor network.

Table 6. Results of third distance measurement test

	Estimated Distance (M)	Actual Distance (M)	Percent Error (%)
Client to Anchor 1	9.53	8.975	6.18
Client to Anchor 2	7.42	6.818	8.83
Client to Anchor 3	7.79	6.934	12.35
Anchor 1 to Sensor	9.38	8.87	5.75
Anchor 2 to Sensor	7.47	6.75	10.67
Anchor 3 to Sensor	7.85	6.99	12.30
Client to Sensor	14.89	13.218	12.65

CHAPTER 4

CONCLUSION AND FUTURE WORK

This chapter summarizes the topics discussed, the work that was done, the limitations found, and future work related to the improvement of this WSN.

Conclusion

In conclusion, this paper successfully demonstrated the implementation of a location-based wireless sensor network (WSN). The project involved preliminary research into wireless sensor network architecture, BLE and BT Mesh, distance measurement physics, and programming with the nRF Connect SDK using devices like the nRF5340 DK and Nordic Thingy 53. The proof of concept for the WSN was implemented with six nodes, enabling the client node to obtain temperature, humidity, and visible light readings from its mesh sensor node. Additionally, the WSN implemented P2P, one-hop distance measurement, allowing the client to approximate the distance to the sensor using the network's three anchors as reference points.

While the implementation was a success, there were some areas where the system showed limitations, leading to less accurate distance estimates. Even with the use of IFFT, obstructions between devices during distance measurement affected accuracy and, at times, caused timeout errors, suggesting a limitation of this technique. Furthermore, distance measurement accuracy was best when there was a clear reference anchor node between the client and sensor, whereas accuracy declined when there wasn't a clear line of sight between the client anchor node and sensor.

Future Work

For future work, there are numerous enhancements that could make this system more efficient and scalable. First, the accuracy of the WSN's distance measurement could be improved by adding more anchor nodes. This implementation used three anchors, but increasing this number would reduce the likelihood of encountering the system's worst-case scenario, as there would be a higher chance of having an anchor node positioned directly between the client and the sensor. More anchor nodes would also reduce dependency on individual anchors, making the system more reliable by allowing the exclusion of any anchor experiencing a timeout or high error from the final calculation.

Scalability could be further achieved by introducing multi-hop functionality, which can be accomplished by expanding the network with additional rows of anchor nodes. Instead of a single row of three anchors, adding two more rows would enable three-hop distance measurements, enhancing the network's reach and accuracy. However, this setup would require each anchor node to be aware of the distances to all other anchor nodes within its radio range. This added complexity would reduce the flexibility in anchor placement but would facilitate multi-hop functionality.

REFERENCES

[1] M. J. McGrath and C. N. Scanaill, “Sensor network topologies and design considerations,” in *Sensor Technologies*, pp. 79–95, 2013, doi: 10.1007/978-1-4302-6014-1_4

[2] Q. Mamun, “A qualitative comparison of different logical topologies for wireless sensor networks,” *Sensors (Basel, Switzerland)*, vol. 12, no. 11, pp. 14887–14913, Nov. 2012, doi: 10.3390/s121114887.

[3] “Bluetooth Low Energy vs. Bluetooth Classic: What’s the Difference? | Laird Connectivity is Now Ezurio.” Ezurio.com. Accessed: Jan. 21, 2024 [Online] Available: https://www.ezurio.com/resources/blog/bluetooth-low-energy-vs-bluetooth-classic-what-s-the-difference

[4] “The Bluetooth® Low Energy Primer Document Version: 1.2.0.” Bluetooth.com. Accessed: Feb. 9, 2024 [Online] Available: https://www.bluetooth.com/wp-content/uploads/2022/05/the-bluetooth-le-primer-v1.2.0.pdf

[5] “Bluetooth Stack Architecture — Zephyr Project Documentation.” Zephyr.com. Accessed: March. 17, 2024 [Online] Available: https://docs.zephyrproject.org/latest/connectivity/bluetooth/bluetooth-arch.html

[6] N. Hlapisi. “Breaking Down the LE Link Layer—The 7 States of Bluetooth LE Radio.” Allaboutcircuits.com. Accessed: Jun. 16, 2024 [Online] Available: https://www.allaboutcircuits.com/technical-articles/breaking-down-the-ble-link-layer-the-seven-states-of-bluetooth-low-energy-radio/

[7] R. Ostapiuk. “Bluetooth® Low Energy Channels - Developer Help.” Microchip.com. Accessed: Jan. 13, 2024 [Online] Available: https://developerhelp.microchip.com/xwiki/bin/view/applications/ble/introduction/bluetooth-architecture/bluetooth-controller-layer/bluetooth-link-layer/Channels/

[8] “The Bluetooth® Mesh Primer | Bluetooth® Technology Website.” Bluetooth.com. Accessed: May. 28, 2024 [Online] Available: https://www.bluetooth.com/bluetooth-mesh-primer/

[9] “The Fundamental Concepts of Bluetooth Mesh Networking Part 1.” Bluetooth.com. Accessed: Aug 13, 2024 [Online] Available: https://www.bluetooth.com/blog/the-fundamental-concepts-of-bluetooth-mesh-networking-part-1/

[10] “The Fundamental Concepts of Bluetooth Mesh Networking Part 2 | Bluetooth® Technology Website.” Bluetooth.com. Accessed: Aug 13, 2024 [Online] Available: https://www.bluetooth.com/blog/the-fundamental-concepts-of-bluetooth-mesh-networking-part-2/

[11] “Bluetooth Mesh Glossary of Terms.” Bluetooth.com. Accessed: Sep 18, 2024 [Online] Available: https://www.bluetooth.com/learn-about-bluetooth/feature-enhancements/mesh/mesh-glossary/

[12] “Provisioning a Bluetooth Mesh Network Part 1 | Bluetooth® Technology Website.” Bluetooth.com. Accessed: Sep 28, 2024 [Online] Available: https://www.bluetooth.com/blog/provisioning-a-bluetooth-mesh-network-part-1/

[13] “Provisioning a Bluetooth Mesh Network Part 2 | Bluetooth® Technology Website.” Bluetooth.com. Accessed: Sep 28, 2024 [Online] Available: https://www.bluetooth.com/blog/provisioning-a-bluetooth-mesh-network-part-2/?utm_campaign=mesh&utm_source=internal&utm_medium=blog&utm_content=provisioning-a-bluetooth-mesh-network-part-1

[14] “Using the Nordic Distance Toolbox to measure the distance between short-range radios.” Nordicsemi.com. Accessed: Sep 30, 2024 [Online] Available: https://blog.nordicsemi.com/getconnected/using-the-nordic-distance-toolbox-to-measure-the-distance-between-short-range-radios

[15] “Technical Documentation.” Nordicsemi.com. Accessed: Apr 9, 2024 [Online] Available: https://docs.nordicsemi.com/bundle/ncs-latest/page/nrfxlib/nrf_dm/README.html

[16] Nordic Semiconductor, *Measuring distance with the Nordic Distance Toolbox*. (Jan. 20, 2023) Accessed: May 12, 2024 [Online Video]. Available: https://www.youtube.com/watch?v=5vZIILTWmW0

[17] K. Heurtefeux and F. Valois, "Is RSSI a good choice for localization in wireless sensor networks?," *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, Fukuoka, Japan, 2012, pp. 732–739, doi: 10.1109/AINA.2012.19.

[18] M. Wehner, R. Richter, S. Zeisberg, and O. Michler, “High resolution approach for phase based TOF ranging using compressive sampling,” pp. 28–32, Apr. 2011, doi: 10.1109/wpnc.2011.5961010.

[19] “Technical Documentation.” Nordicsemi.com. Accessed: May 12, 2024 [Online] Available: https://docs.nordicsemi.com/bundle/ug_nrf5340_dk/page/UG/dk/intro.html

[20] “Technical Documentation.” Nordicsemi.com. Accessed: May 14, 2024 [Online] Available: https://docs.nordicsemi.com/bundle/ug_thingy53/page/UG/thingy53/intro/frontpage.html

[21] “Technical Documentation.” Nordicsemi.com. Accessed: May 17, 2024 [Online] Available: https://docs.nordicsemi.com/bundle/ncs-latest/page/nrf/index.html

[22] “Technical Documentation.” Nordicsemi.com. Accessed: May 8, 2024 [Online] Available: https://docs.nordicsemi.com/bundle/nrf-connect-vscode/page/guides/ncs_configure_app.html

[23] X. Zhang, W. Wang, X. Xiao, H. Yang, X. Zhang, and T. Jiang, “Peer-to-peer localization for single-antenna devices,” *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, vol. 4, no. 3, pp. 1–25, Sep. 2020, doi: 10.1145/3411833.

[24] A. Dhekne, U. J. Ravaioli, and R. R. Choudhury, "P2PLoc: Peer-to-peer localization of fast-moving entities," *Computer*, vol. 51, no. 10, pp. 94–98, Oct. 2018, doi: 10.1109/MC.2018.3971349.