Warehouse BLE Mesh Climate Monitoring System

Submitted by

Syed Mustafa Amjad	19i-0807
Abdullah bin Zahid	19i-0904
M. Zeeshan Ansar	19i-0772

Supervised by

Engr. Aamer Hafeez	Internal Supervisor

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Islamabad

2023

Developer’s Submission

“This report is being submitted to the Department of Electrical Engineering of the National University of Computer and Emerging Sciences in partial fulfillment of the requirements for the degree of BS in Electrical Engineering”

Developer’s Declaration

“We take full responsibility for the project work conducted during the Final Year Project (FYP) titled **“Designing and Implementing a Warehouse Climate Monitoring System using Bluetooth Low Energy (BLE) Mesh Network”.** We solemnly declare that the project work presented in the FYP report is done solely by us with no significant help from any other person; however, small help wherever taken is duly acknowledged. We have also written the complete FYP report by ourselves. Moreover, we have not presented this FYP (or substantially similar project work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

We understand that the management of the Department of Electrical Engineering of National University of Computer and Emerging Sciences has a zero-tolerance policy towards plagiarism. Therefore, we as an author of the above-mentioned FYP report solemnly declare that no portion of our report has been plagiarized and any material used in the report from other sources is properly referenced. Moreover, the report does not contain any literal citing of more than 70 words (total) even by giving a reference unless we have obtained the written permission of the publisher to do so. Furthermore, the work presented in the report is our own work and we have positively cited the related work of the other projects by clearly differentiating our work from their relevant work.

We further understand that if we are found guilty of any form of plagiarism in our FYP report even after our graduation, the University reserves the right to withdraw our BS degree. Moreover, the University will also have the right to publish our names on its website that keeps a record of the students who committed plagiarism in their FYP reports.”

____________________	____________________	____________________
Syed Mustafa Amjad	Abdullah Bin Zahid	M. Zeeshan Ansar
BS(EE) 2019-0807	BS(EE) 2019-0904	BS(EE)2019-0772

____

Certified by Supervisor

____

Verified by Plagiarism Cell Officer

Dated: __________

Abstract

This final year project report presents the development and implementation of a Warehouse Climate Monitoring BLE (Bluetooth Low Energy) Mesh Network System. The system offers an efficient and effective solution for monitoring temperature and humidity levels in warehouses, surpassing traditional WiFi-based alternatives. Leveraging BLE technology, the system captures real-time data from strategically placed sensor nodes and aggregates it through a mesh network topology. Compared to WiFi-based solutions, the BLE Mesh Network System requires less hardware, provides scalability, and reduces power consumption. The successful implementation of the system demonstrates its potential to revolutionize warehouse climate monitoring. With proactive monitoring, warehouse owners and operators can ensure optimal storage conditions, minimizing spoilage risks and enhancing supply chain management. The project highlights the advantages of BLE mesh networks and their ability to transform warehouse climate monitoring systems.

Acknowledgements

We would like to express our sincere gratitude to our supervisors, Engr.Aamer Hafeez, Engr. Hamza Ali Imran and Engr. Saffiullah Hussaini for providing their invaluable guidance, comments and suggestions throughout the course of this project. We would specially thank Engr. Aamer Hafeez for constantly motivating us to work hard and the importance of group work and Engr. Hamza Ali Imran for authorizing our access to Emumba.

Table of Contents

List of Figures

1. Problem Statement

Warehouse goods can easily become spoiled or filled with mold under inappropriate temperature and humidity levels. Climate-sensitive goods like pharmaceuticals, dairy, and beverages are especially susceptible. Millions of dollars worth of products can be lost unexpectedly and disrupt the supply chain. A constant temperature and humidity monitoring system can help circumvent these losses.

2. Introduction and Background

Warehouse owners usually underestimate humidity and temperature as key elements in keeping goods in optimal and safe condition while in inventory. High humidity and temperature damage products causing unexpected losses. Excessive humidity causes mildew and mold growth on products, shelves, boxes, and walls whilst metal parts start corroding [1].

Mold is a type of fungi, some are edible such as mushrooms, and others are used in the baking of bread and brewing alcohol. Most fungi live off of decaying biological matter, however, some can be damaging to health. Microscopic spores are produced and used for the reproduction of the fungus which leads to respiratory illnesses in a closed environment. These spores travel through the air, settle on a suitable surface and begin to grow [1]. When spores settle on a surface, to thrive there should be sufficient food to sustain the fungi. Mold can grow on any biological material. It can attack the most common materials used in warehouse packings such as wooden crates, and cardboard boxes. Eliminating all sources of food for mold is also an almost impossible task. High temperatures are also extremely dangerous for a large variety of products especially dairy or dairy-based products as these will spoil instantly under extreme temperatures [2].

This is not a new problem. Many companies have temperature and humidity sensors installed all over the warehouse. Specially trained personnel are employed to watch over these sensors and record data at regular time intervals. This is not an optimal solution to the issue. Firstly, we have human error involved which means forgetting to log data of a specific room or incorrectly reading data from thermometer/humidity sensors.

Secondly, we have the issue of a constant watch as hiring multiple workers for reading sensor data in each room is a waste of resources. Due to these reasons, we need a system where we can gather all the sensor's data at short time intervals and log them into a database. Parameters can be set for dangerous/concerning levels of humidity and temperature thus sounding alarms for immediate action. This will reduce the number of workers needed and it will provide the convenience of monitoring the sensors through a singular device.

3. Conceptualization

Sensor nodes for monitoring warehouse climate will be installed at various locations in the warehouse. Each node is capable of measuring and sending sensor data through a wireless protocol to a smart device. The maintenance crew can monitor temperature and humidity by connecting their smart devices with the sensor network. Each sensor node will publish its measurements periodically to all connected smart devices. We can specify upper and lower threshold values for temperature and humidity using the mobile application. If the temperature or humidity readings are not within the threshold limits, then the mobile app will immediately notify the crew. Upon receiving a timely notification, the crew can quickly handle the situation before it causes severe damage.

3.1. Existing Solutions

Currently, there are 4 wireless communication technologies that can be used for warehouse monitoring systems; Wi-Fi, Zigbee, Z-Wave, and LoRaWAN. Multiple microcontrollers support these technologies. Wi-Fi is the most commonly used wireless technology, especially in warehouse management applications [3]. Wi-Fi allows for 300 Mbps of maximum bandwidth, cloud connectivity and direct support for any smart device with Wi-Fi enabled. Power usage is between 5 and 10 watts per device, the range is only 15 to 45 meters, and the mesh network is not supported [4]. A lot more routers will be required for the topology due to no mesh network support resulting in a high per-unit cost.

Zigbee can also be used, it has an amazing range of up to 100 meters, power usage of only 0.01 to 0.1 watts, and mesh network support [5]. These factors allow it to be an excellent fit for our project but the biggest downside is its incompatibility with smart devices through a direct connection.

Z-Wave technology also suffers only in this regard, the range is 100 meters, mesh support is enabled and power consumption is only 0.08 watts per device but no direct connection with smart devices [6]. The last of the technologies is LoRaWAN, it has the highest signal range of all available technologies at a staggering 5 to 10 kilometers, with a power usage of 0.25 watts and mesh network support. LoRaWAN would have been the perfect candidate if not for the fact that each module costs a lot more than the rest and it does not have direct smart device support [7].

3.2. Proposed Solution

Bluetooth Low Energy (BLE) is the most appropriate wireless communication technology for our sensor devices. Firstly, this project requires battery-powered sensor nodes and highly power-efficient communication hardware. BLE is designed for such applications where power consumption is a high priority such as battery-powered devices [24]. Other wireless technologies like Zigbee, Z-wave, and Wi-Fi are power-hungry alternatives so they don’t fit well in this case. Power-efficient devices are required for our use case and BLE happens to be the only technology that meets this requirement.

Secondly, the data sent by sensor nodes are small and infrequent, so there is a need for wireless technology which is specifically designed for such applications. BLE is most suitable for applications with relatively short range and infrequent low-bandwidth data transfers. Other possible alternatives are simply not satisfactory for this project because some of them are designed for high data rates such as Wi-Fi while others are specifically designed for industrial or home automation applications.

Finally, every smart device running iOS or Android supports BLE which makes it a better choice. We can easily connect the sensor network with smart devices like mobile phones or tablets. Other alternatives like Zigbee and Z-wave can’t directly connect to smartphones.

The proposed solution uses a DHT22 temperature and humidity sensor which has an operating temperature of -40 to 60 Celcius. It is not suitable for such warehouses whose climate conditions don't fall under the operating range of our device. The optimum operating temperature for the battery is -20 to 60 Celcius. So the warehouse climate conditions must be under the range.

4. Block Diagram of Main Modules

Project Block Diagram

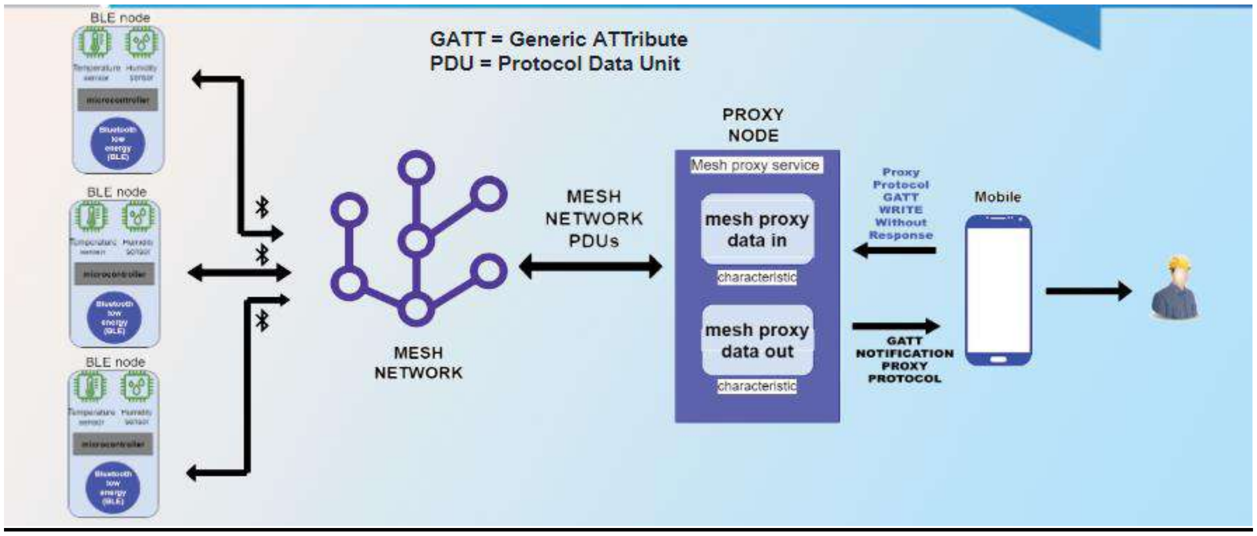


Figure 1: Block diagram of the project

Project Network Block Diagram

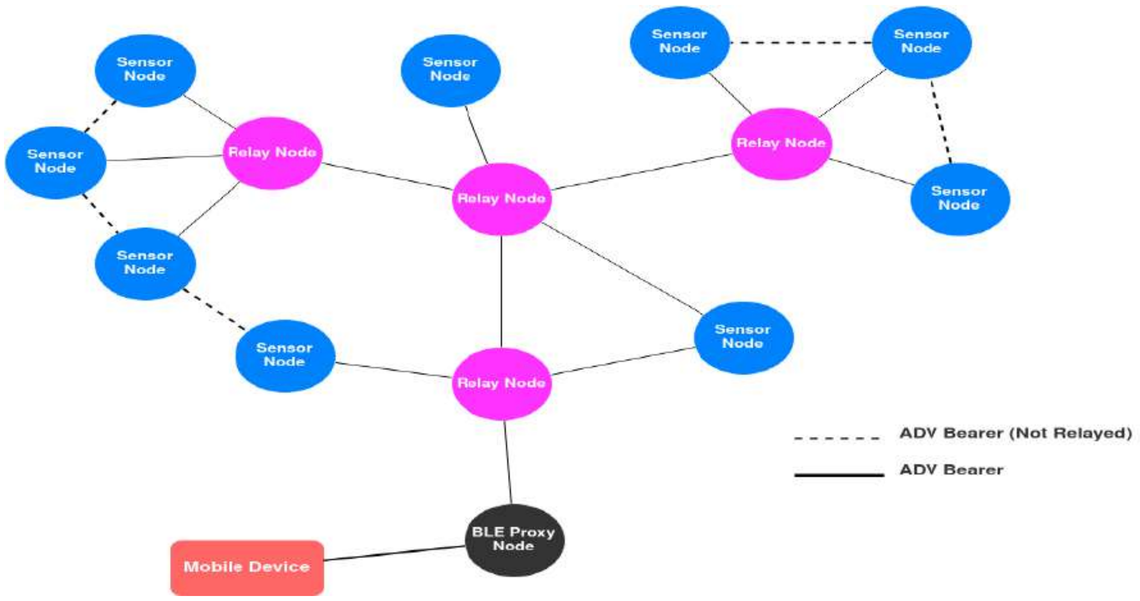


Figure 2: Network block diagram of the project

Proxy Node Block Diagram

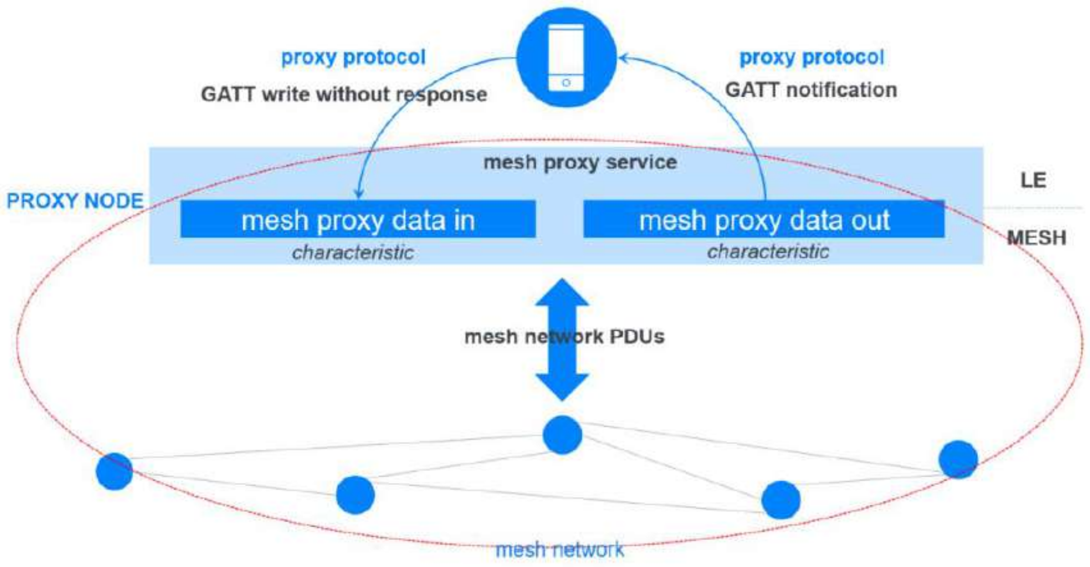


Figure 3: Block diagram of the proxy node

5. Flow Charts

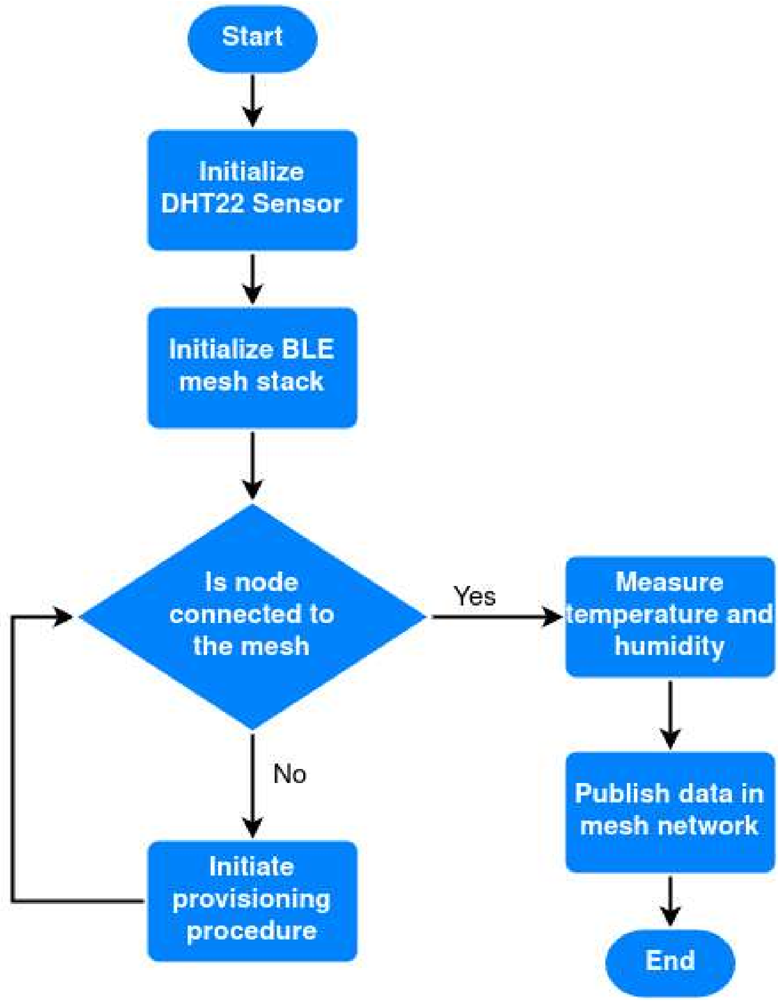

Start
Initialize DHT22 Sensor
Initialize BLE mesh stack
Is node connected to the mesh
Yes
No
Measure temperature and humidity
Publish data in mesh network
End
Initiate provisioning procedure

Figure 4: Flow chart for sensor node

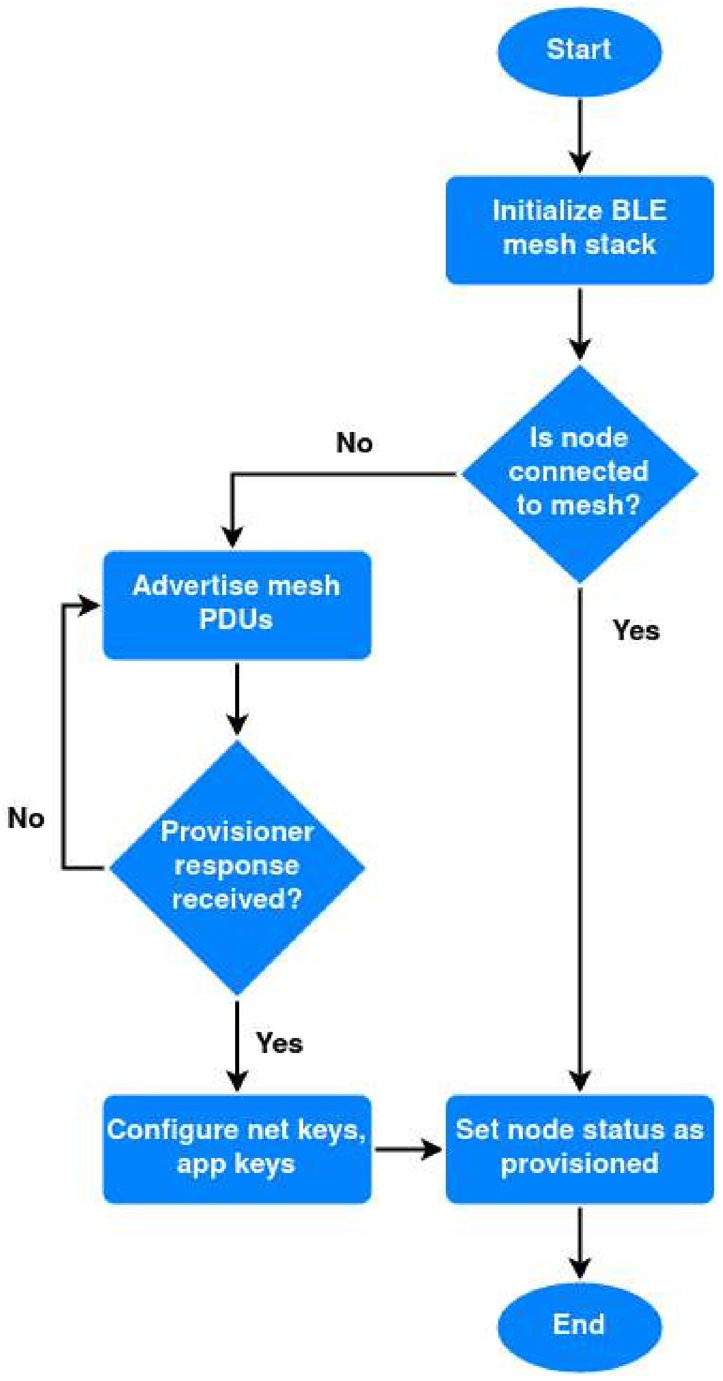

Start
Initialize BLE mesh stack
Is node connected to mesh?
No
Yes
Advertise mesh PDUs
Provisioner response received?
No
Yes
Configure net keys, app keys
Set node status as provisioned
End

Figure 5: Flow chart for node provisioning procedure

6. Hardware

Three types of computing platforms are applicable; System-on-Chip (SOC), Single-Board Computers (SBC), and Microcontrollers. All three have their benefits and drawbacks. The SOC incorporates many system components into a single silicon chip allowing it to be extremely compact and lightweight. Power requirements are lower, processors are fast and memory capacity is huge. However, the cost of these boards is extremely high and they need to be specifically manufactured for the task at hand.

SBC is an entire computer built on a single circuit board making it very easy to use and modify for various applications but the cost of these computing devices is high [8].

Another alternative is a microcontroller which is a microcircuit (IC) that is programmed to regulate electronic devices which makes it easy to use and the cost is very low. Due to its simplicity, it cannot carry out extremely complex computations. Microcontrollers are the appropriate choice for our purposes as we only need simple computations, low energy, and low costs [9-10].

6.1. Microcontroller Unit (MCU) Selection Criteria

Choosing the right microcontroller is also challenging as there are a plethora of MCUs on the market. We have devised four main selection criteria for our boards; complete BLE support, availability, cost, and software support.

6.1.1. BLE Support

Some of the most optimal boards for our task are; ESP32, Raspberry Pi Pico W, Arduino Nano 33 BLE, and Adafruit Feather NRF52. All of these boards support BLE Mesh Network technology [11-14].

6.1.2. Availability

ESP32 is readily available in Pakistan on sites like hallroad.org, daraz.com, and more [15-16]. The Raspberry Pi Pico W is a new release this year but it is also present in Pakistan [17]. Arduino Nano 33 and Adafruit Feather NRF52 are also readily available [18-19]. All the boards in our selection are easy to source so we can approve all the boards under the availability criteria.

6.1.3. Cost

All the boards selected have varying price tags. The prices in order from the highest to the lowest are as follows; Arduino Nano 33 BLE, Adafruit feather NRF52, Raspberry Pi Pico W, and ESP32 [10-13]. As we can see the Adafruit Feather and Nano 33 BLE both have prices above 5000 PKR while the ESP32 and Pico W remain below 2000 PKR. In our mesh network, we require a large number of sensor nodes due to which the Adafruit Feather and Nano 33 BLE fail the cost criteria.

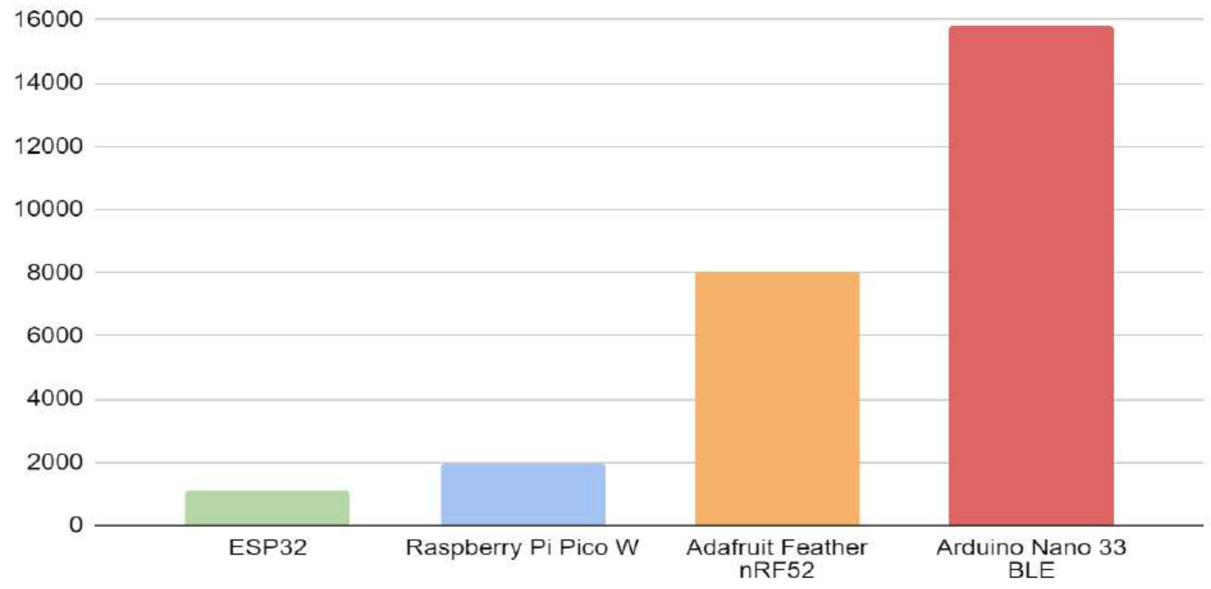


Figure 6: Cost comparison of various embedded computing platforms.

6.1.4. Software Support

For our project, we want the chosen board to have an extensive library for implementing the BLE mesh network. The board should support a good compiler for efficient programming. ESP32 is supported by the Arduino IDE allowing us to write the code in C, and have it compiled and loaded easily. It also possesses the ESP32 BLE library for Arduino IDE which is extremely useful when using advanced BLE features [20].

The Raspberry Pi Pico W is a new board as such it currently supports Micro Python UF2 but BLE libraries for the device are not officially supported [21]. Due to it being a new board very few people have worked on it thus its compatibility and applications are not currently well-defined. The Arduino Nano 33 BLE is officially supported by the Arduino IDE, and also it uses the same chip as other Arduino boards making the software library support very extensive. Not only that but it can also be programmed in Python through the OpenMV IDE [22].

The Adafruit Feather nRF52 has a massive number of apps and compilers for its programming, some include; Bluefruit LE Connect for Android/iOS, ABLE for cross-platform, and Bluefruit LE Python Wrapper for programming in python [23]. According to our research, only the Raspberry Pi Pico W fails this criterion due to it being new and having very few online resources for BLE and BLE Mesh support.

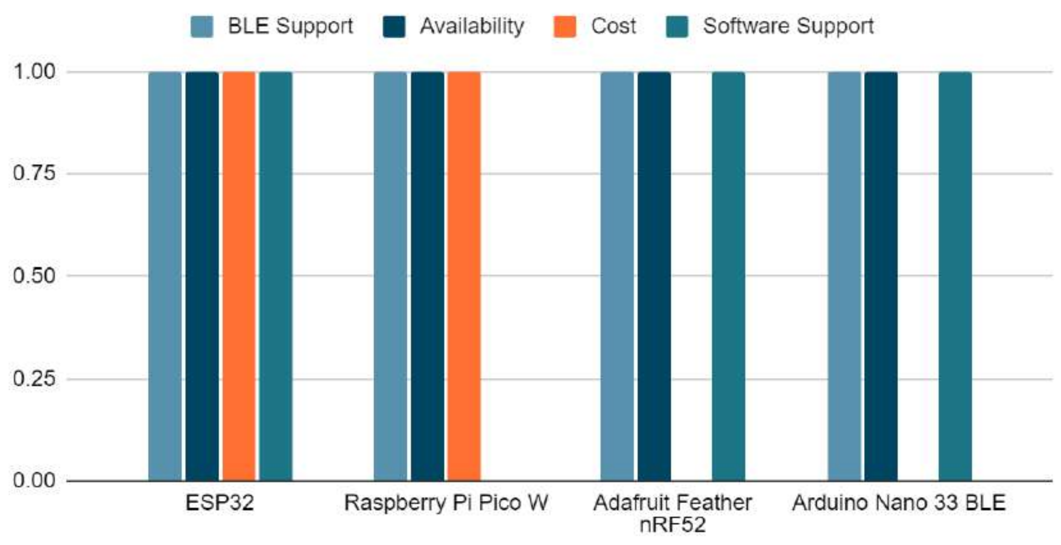


Figure 7: Hardware selection criterion score of various microcontrollers

In summary, the ESP32 has a perfect score of four as it has a cheap price, complete BLE support, availability, and extensive software support. Hence, we will be using the ESP32 for our project.

7. Design Methodology

7.1. Stepwise Refinement

We implored a stepwise refinement methodology. We noted down the features of the final product and then traced it back to individual tasks. The final product had three key features; wireless long-range connection, low power utilization, and easy connectivity to android devices.

To achieve a long-range wireless connection we selected BLE and LoraWAN. Both of these also fulfill low power consumption requirements. When it came to connectivity to android devices we could only go forward with BLE as LoraWAN has no direct support.

As we selected BLE for wireless communication, we now have to build a device around it. For this, we used the Esp32 microcontroller as it fully supports all BLE modes. Secondly, we chose Android Studio to make an application to read the data of the BLE sensor devices.

For the Esp32 we decided to code in ESP IDF as it allows us to fully customize the microcontroller for our various needs. For android studio, we chose to develop in Java as it is the industry standard with multiple libraries and coding techniques.

7.2. Stepwise Refinement Diagram

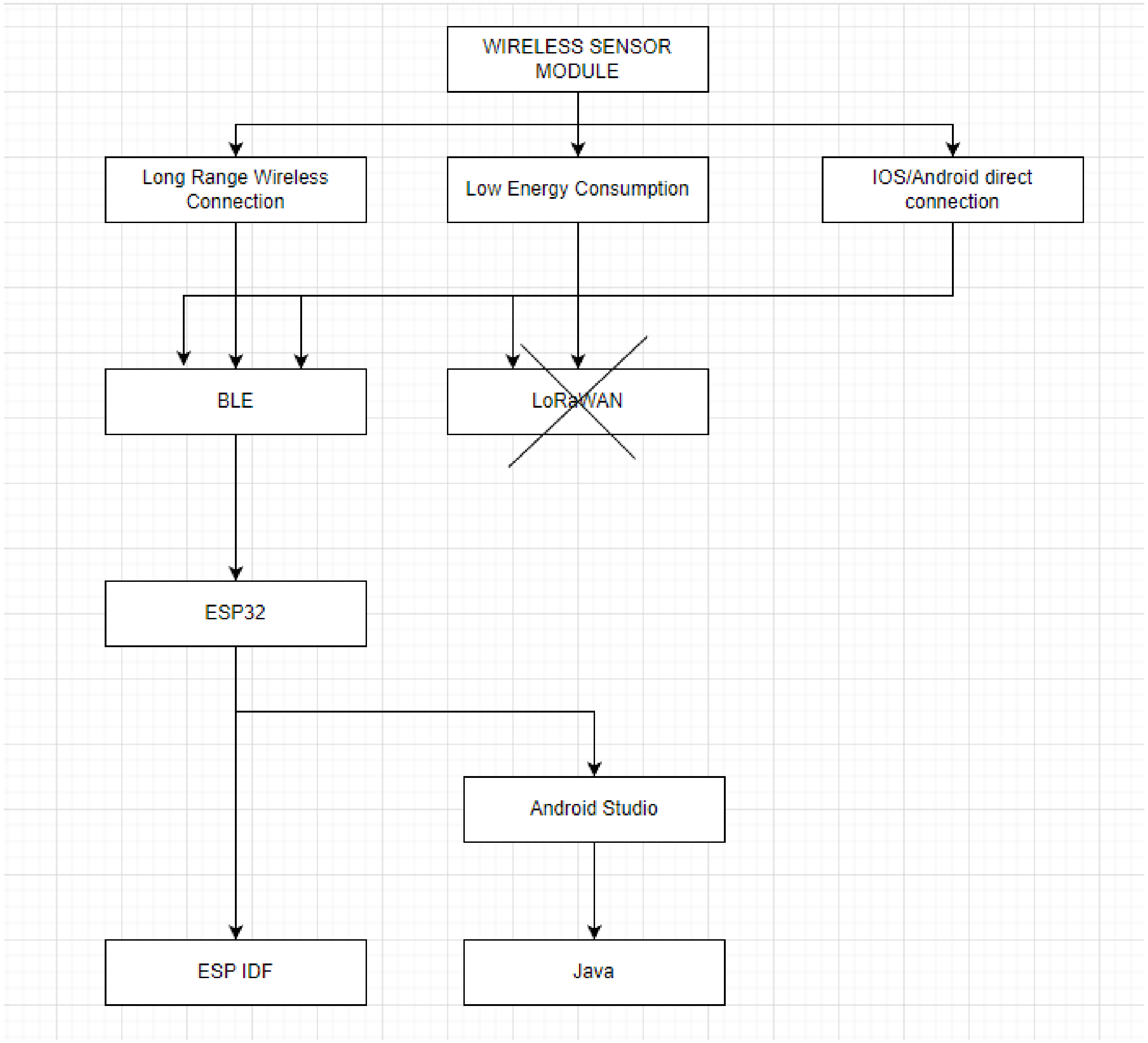


Figure 8: Tree diagram for step-wise refinement of final solution

8. Project Scope

8.1. Project Objective

Develop a warehouse climate monitoring system using BLE(Bluetooth Low Energy) technology within 8 months at a cost not to exceed Rs. 2500 per sensor node.

8.2. Milestones

- Sensor nodes are programmed to connect using BLE mesh technology.
- Proxy node programmed to connect with the sensor nodes.

- Software created to establish a connection with all the nodes and log data on the phone.
- Testing of the entire system in FAST or at home.

8.3. Technical Requirements

- Sensor node devices need to have two months of battery backup.
- Surge protector needs to be installed with the proxy node device.
- Android application needs to be approved and published on the google play store.
- Signal strength between all devices has to be above certain levels at all times.

8.4. Limits and Exclusions

- The sensor nodes record and post temperature every 15 minutes due to an incredibly high battery time of 2 months
- Battery time is limited to 2 months as any higher would require a very large battery
- Application for the system is only built for android due to the complexity.
- Engineers will only come to set up the system once and one return visit to check the system for faults.
- A warranty of 2 months will be provided on the system.
- Physical damage to devices will not be covered under warranty.

9. Project Design

The project aims to design a warehouse climate monitoring system. To develop such a system, we need specific hardware components and software frameworks. The project hardware consists of an ESP32-S2 series microcontroller, a dht22 temperature and humidity sensor, and a rechargeable lithium-ion battery. The software framework required is the Iot Development Framework (IDF) from Espressif Systems. A detailed description of each component is provided below:

9.1. DHT22 Sensor

The DHT22 is one of the widely used types of sensors in climate monitoring applications. It is a capacitive-type digital sensor that can measure relative humidity and temperature. The major features of this sensor are as follows:

- The out of the sensor is a calibrated digital signal which removes the processing overhead from the µcontroller
- This sensor can operate on extremely low power which makes it a perfect choice for wireless sensing applications which require low power consumption.
- This sensor has a sufficiently long range for data transmission and excellent signal strength.
- This sensor can operate in a variety of environments without affecting the measurement sensitivity.

The DHT22 sensor has an on-chip processing unit that outputs calibrated data signals. It eliminates the overhead of signal processing on the µcontroller. Moreover, it also allows convenient interfacing of the sensor with the µcontroller. This sensor uses polymer-type capacitor technology for sensing temperature and humidity. This sensing technology provides highly accurate and stable measurements in a variety of environments making it suitable for warehouse climate monitoring applications.

The DHT22 sensor is already calibrated by the manufacturer and the calibration factor is stored in the memory of the sensor module. The sensor uses this factor to correct the measured temperature. The time taken by the sensor for each measurement is 5 milliseconds. The sensing frequency of DHT22 is 1 hertz for accurate measurements.

When µcontroller asserts the data line, the DHT22 sensor transitions from a low-power state to a running state. Shortly after the data line assertion, the µcontroller de-asserts the data line and the sensor starts data transmission. The sensor sends a 40-bit data packet containing the temperature and humidity data.

9.2. ESP32 SoC

For sensor interfacing and BLE communication, we are using an ESP32-S2 series system-on-chip (SoC) from Espressif Systems. It is a low-cost BLE chip with excellent processing performance. Due to its low power consumption and processing performance, it is a perfect choice for sensor network applications.

ESP32 chip supports multiple modes of operation based on the power consumption of the chip. These are active mode, light–sleep mode, and deep sleep mode. It can dynamically switch between these power modes during the program execution.

We'll use the chip in light sleep mode to increase battery life. In the light sleep mode, it only consumes 750µA.

9.3. ESP-IDF Framework

For building the software for sensor nodes, we are using the ESP-IDF version 4.0 framework developed by Espressif Systems for its ESP-S2 series devices. It provides a rich set of APIs for BLE, light sleep mode, and GPIOs.

9.4. BLE Mesh

We're using Bluetooth mesh technology for establishing a wireless sensor network (WSN). In the mesh topology, each sensor node is connected to all other nodes that lie within the radio range. The main advantages of using mesh network topology are the long range of communication and the robustness of the network. Sensor nodes can relay messages to other nodes in the network that are not directly reachable. This allows a mesh network to increase its range of communication.

Mesh network also supports dynamic removal and addition of new sensor nodes to the network without affecting the performance of the network. The mesh network can re-configure the routes whenever the network topology changes due to the addition or removal of any node.

9.4.1. BLE Nodes

BLE mesh stack supports optional node roles to meet the networking requirements. These are relay nodes and proxy nodes. These roles can only be enabled at the time of programming and they can be disabled anytime when the node is in a running state.

The relay nodes are used for relaying messages in the network to increase the range of the network. They do so by receiving the broadcasted messages and then retransmitting them to other nodes.

Proxy nodes are used to allow non-mesh BLE devices to communicate with the mesh network. A proxy node acts as an intermediary in the network. It does so by simultaneously connecting itself with the mobile device and mesh network. Both the mesh and non-mesh devices use different formats for data. Mesh devices use mesh PDUs and non-mesh devices use GATT. The mobile device uses GATT messages to send data to the proxy node. Then the proxy node converts this data to a mesh PDU and transmits it to the mesh network. Similarly, when a mobile device requests data from the mesh network. The proxy node converts the incoming data from the mesh network to GATT and sends it to the mobile device.

9.4.2. Node Provisioning Procedure

The main task of provisioning is to configure new sensor nodes and add them to the network. Mostly, the provisioner, a device that adds new nodes to the network, is a smartphone. Usually, an app is used for handling all the provisioning-related tasks like assigning UUID addresses and configuring encryption keys. Node Provisioning is a two-way communication procedure that is normally initiated by an un-provisioned node.

The un-provisioned node starts the process by becoming a BLE beacon. A beacon normally advertises itself by broadcasting mesh advertisement packets. These packets are then received by a nearby provisioner device. The provisioner device upon receiving the advertisement packets can either drop those packets or send an invitation.

To start the provisioning process, the provisioner sends an invitation to the un-provisioned node. Upon receiving the invitation, the un-provisioned device starts sending device information to the provisioner. This information normally consists of mesh sensor models and the elements supported by the device. Information regarding the security algorithms is also shared between the provisioner and the node.

To encrypt the communication, both devices exchange keys using the Diffie-Hellman key exchange algorithm. Both devices use the exchanged keys to encrypt the communication process. The exchange of additional keys along with the provisioning data is carried out next. The provisioner device and the node share the networking information with each other. Finally, a unicast address is assigned to the node, and the provisioning procedure terminates.

9.4.3. BLE Mesh Network Security

The BLE mesh technology mandates security in the network by enforcing the encryption of data packets. It uses Advanced Encryption Standard (AES) for the encryption of mesh data packets. It implements security at three levels: network, application, and device level. Due to implementation at three independent levels, the mesh specifies the usage of three different encryption keys. These are network keys, application keys, and device keys.

Since the communication at the network level is encrypted, a node requires a network key to decrypt it. So, sensor nodes must possess a network key to become a part of the network. Network-level encryption helps to keep malicious nodes off the network. A node possessing a network key can only decrypt data from the network layer.

Similarly, the data at the application level is also encrypted using application keys. Application keys are used for encrypting the data of applications running on the sensor node. In our case, it is the data of sensor nodes i.e, temperature and humidity which is encrypted. Sensor nodes running the same applications share the same application keys with each other for encryption and decryption. This helps to prevent any spoofed application data from being transmitted in the network by a malicious node.

During the initial phase of provisioning, the communication between the provisioner and the un-provisioned device is secured using the device key. It helps to encrypt the flow of provisioning data which includes device UUID, services, and their characteristics.

9.5. Android Application

Our goal is to develop an Android application that serves as a valuable tool for monitoring and managing devices within a BLE mesh network. The app will provide a user-friendly interface to track and control various sensors connected to the network. The primary objective is to enable users to stay informed about environmental conditions and receive notifications when specific thresholds, such as temperature or humidity, are exceeded.

To achieve these goals, we will leverage the power of Flutter, a cross-platform development framework. Flutter's versatility and extensive widget library will enable us to create an intuitive and visually appealing user interface. The application will also integrate BLE Mesh functionality using appropriate packages and libraries to establish seamless communication with the devices in the network.

The app's core features will include a comprehensive dashboard displaying all devices within the BLE mesh network. Users will be able to monitor sensor data in real-time, configure threshold values, and receive notifications. Additionally, the application will support provisioning of new sensor nodes, ensuring easy integration of new devices into the network. Network security will also be prioritized, allowing users to manage encryption keys for enhanced data protection.

9.5.1. Flowcharts

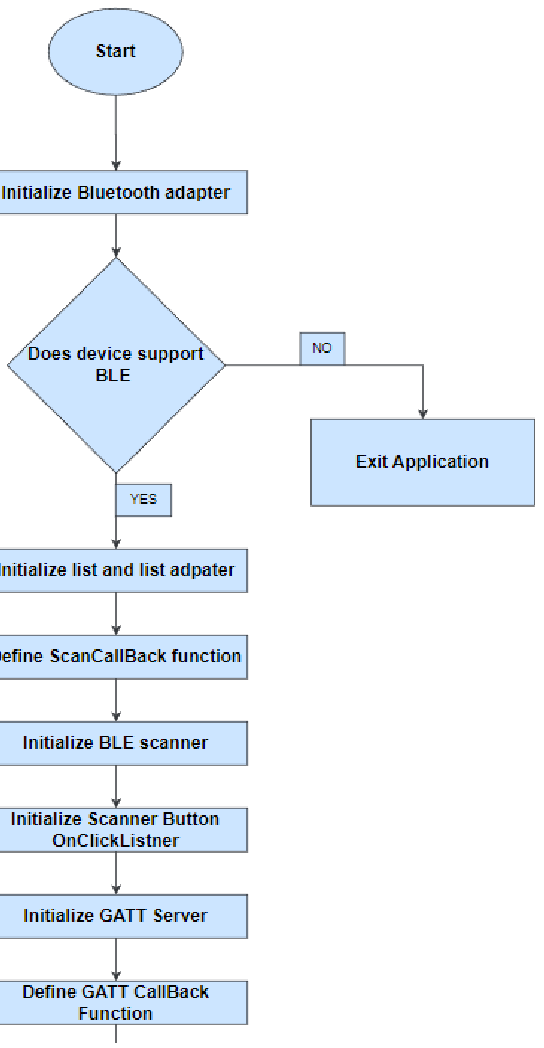

Start
Initialize Bluetooth adapter
Does device support BLE
NO
Exit Application
YES
Initialize list and list adpater
Define ScanCallBack function
Initialize BLE scanner
Initialize Scanner Button OnClickListner
Initialize GATT Server
Define GATT CallBack Function

Figure 9 (continued): Flow chart for the android based application

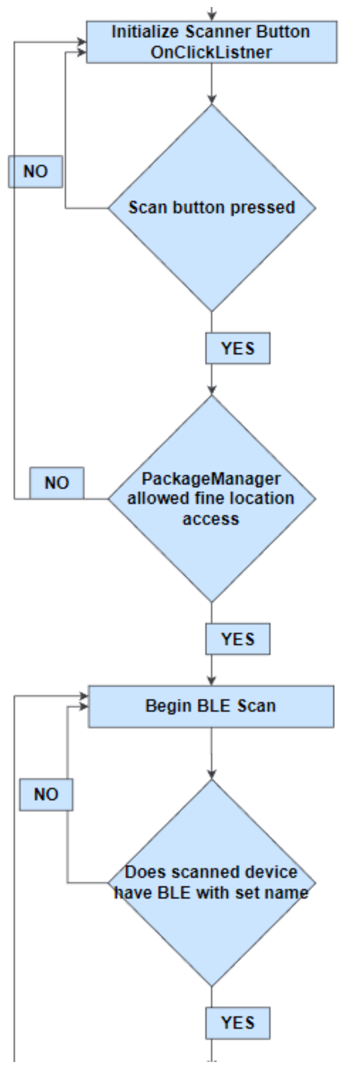

Initialize Scanner Button OnClickListner
Scan button pressed
NO
YES
PackageManager allowed fine location access
NO
YES
Begin BLE Scan
NO
Does scanned device have BLE with set name
YES

Figure 9 (continued): Flow chart for the android based application

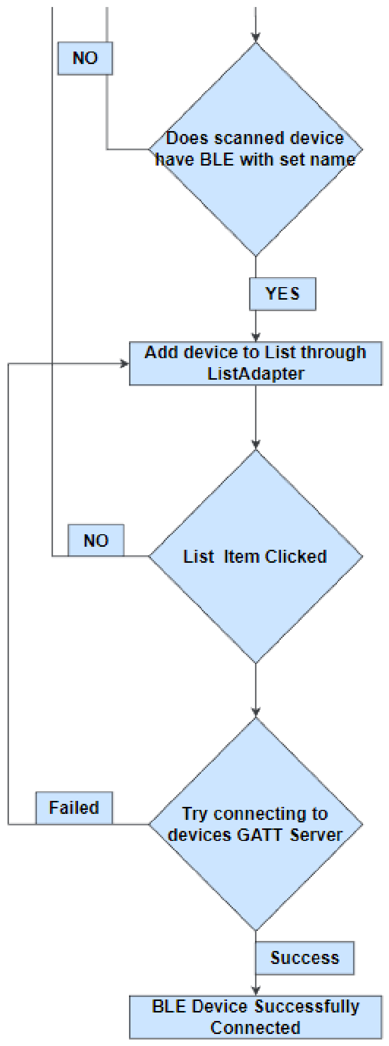

NO
Does scanned device have BLE with set name
YES
Add device to List through ListAdapter
NO
List Item Clicked
Failed
Try connecting to devices GATT Server
Success
BLE Device Successfully Connected

Figure 9: Flow chart for the android based application

10. Design Testing and Data Collection

The firmware for sensor nodes is tested in real-time under various conditions to check whether it meets our requirements or not. Initial testing involved the operation of mesh topology with a variable number of nodes in the network. In this test, we created a topology consisting of four sensor nodes. Each sensor node measured climate data and sent it over the mesh network. Each sensor node also had a relay feature enabled which means it can forward data coming from other sensor nodes.

During the test, we connected an android based mobile device to the proxy node. The android device could not directly communicate with the mesh network. So, we connected the android device with a proxy node which enabled bidirectional communication between the mesh network and the android device. The proxy node acted as a bridge between the mobile device and the mesh network.

Whenever the sensor nodes had to send the data to a mobile device, they would send it to the proxy node. And then the proxy node would send it to the mobile device. The same is true when the mobile device wants to send data to the mesh network. It would send its data to the proxy node, and then the proxy node would forward it to the respective node in the mesh network.

We also tested the extended range of mesh networks. The minimum distance at which devices can communicate within a building is approximately 15 meters. However, by using the mesh network, this range could be extended. The sensor nodes can also function as relay nodes and forward the network traffic between nodes thus effectively increasing the range of the network.

We also tested the self-healing ability of the mesh network. It can dynamically add new nodes and reconfigure network paths. It also supports the dynamic removal of nodes without affecting the communication process or requiring any reconfiguration.

Each sensor node can accurately measure the temperature and humidity with an accuracy of 95% in humidity measurement and 98% in temperature measurement.

11. Result

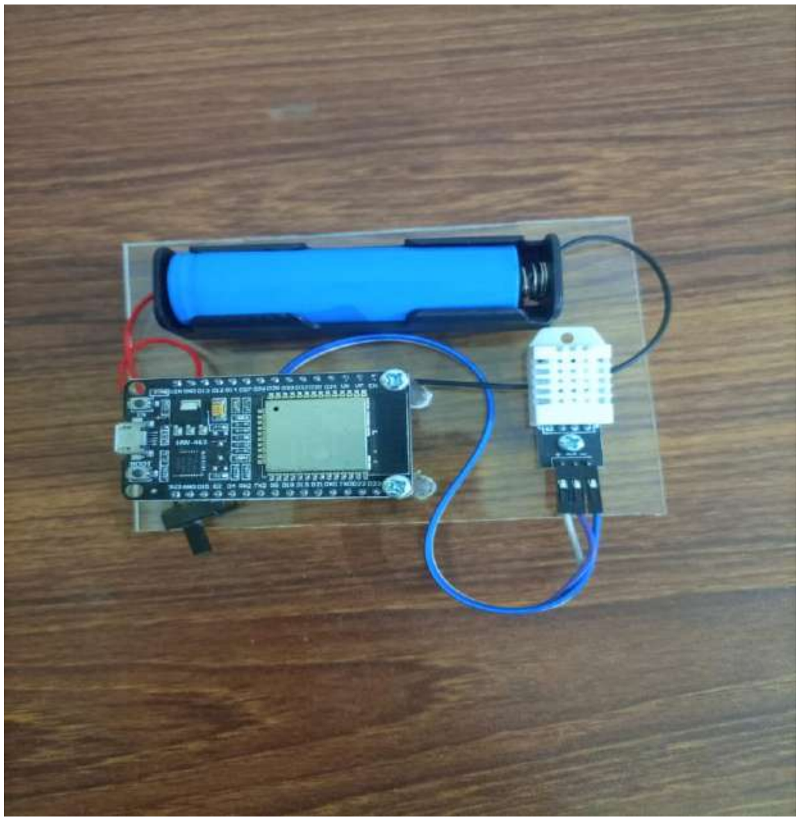


Figure 10: Prototype of a climate monitoring node

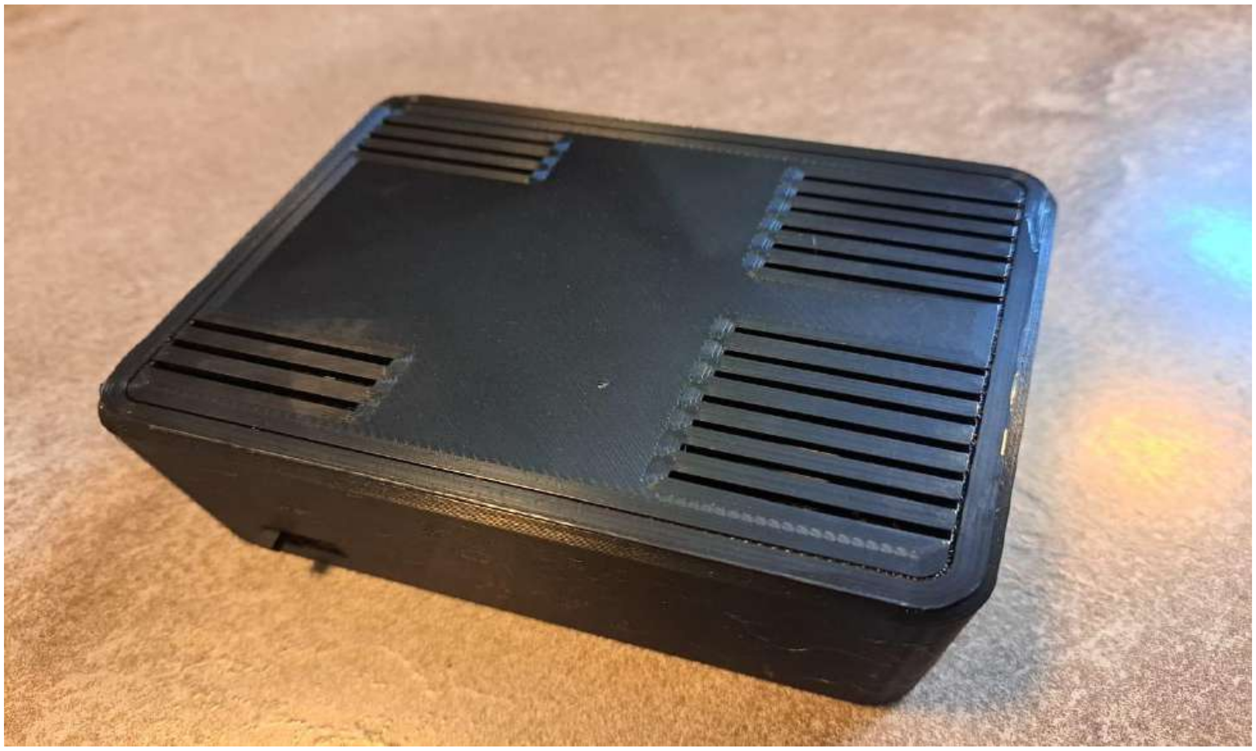
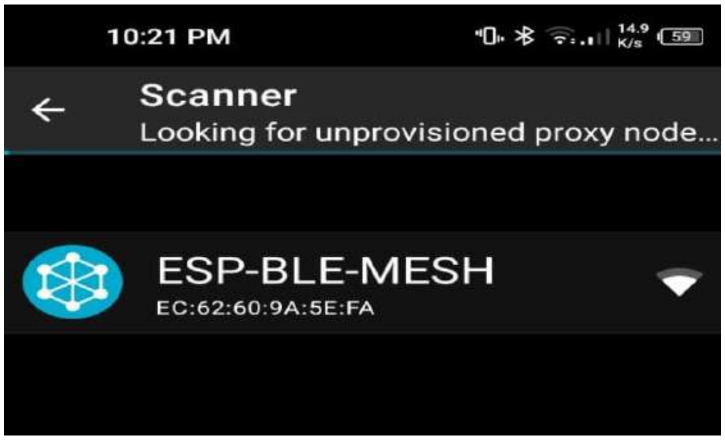


Figure 11: 3-D printed housing for climate monitoring node

11.1.1. Device Configuration

Step: 1
During the configuration phase, we initiate the process by conducting a thorough scan of the nearby BLE devices within the mobile phone's range. As soon as the nRF Mesh mobile application identifies a new device within its proximity, it promptly displays the device's name and MAC address in the scanner, as depicted in the figure.



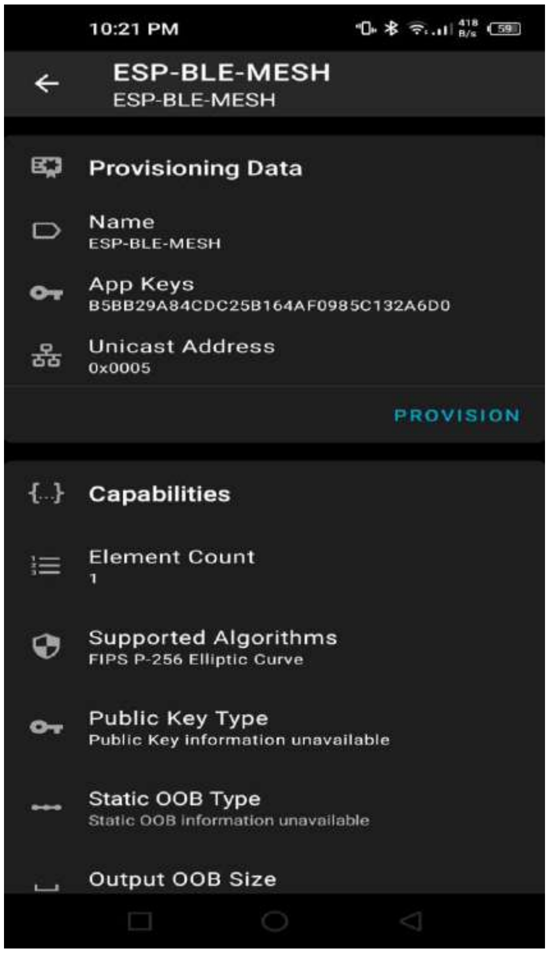
Looking for unprovisioned proxy node...

ESP-BLE-MESH
EC:62:60:9A:5E:FA

Figure 12: Node discovery phase

Step: 2

Upon tapping on the device showcased in the scanner, you'll be directed to a user-friendly interface that presents comprehensive information about the device, including its name, unicast address, and app keys. To seamlessly incorporate this device into the BLE network, simply tap on the provision button, and watch as this device seamlessly becomes an integral part of the network.



10:21 PM

ESP-BLE-MESH
ESP-BLE-MESH

Provisioning Data

Name
ESP-BLE-MESH

App Keys
B5BB29A84CDC25B164AF0985C132A6D0

Unicast Address
0x0005

PROVISION

Capabilities

Element Count
1

Supported Algorithms
FIPS P-256 Elliptic Curve

Public Key Type
Public Key information unavailable

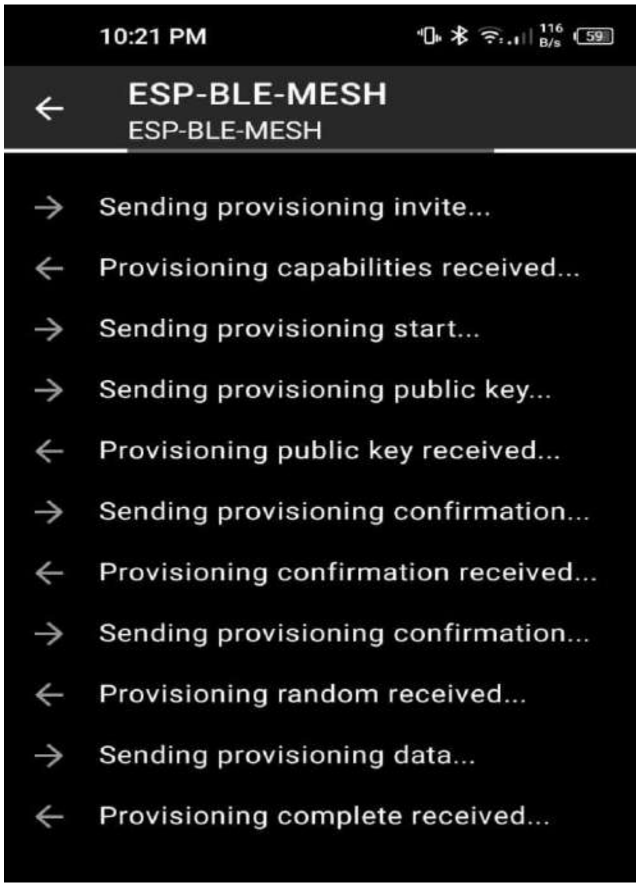
Static OOB Type
Static OOB information unavailable

Output OOB Size

Figure 13: Device provisioning menu

Step: 3

Once you initiate the provisioning process by tapping on the provision button, a seamless exchange of configuration messages between the mobile app and the BLE device ensues. This meticulously orchestrated process is designed to ensure a smooth and successful provisioning of the device, enabling it to seamlessly integrate into the network.



10:21 PM

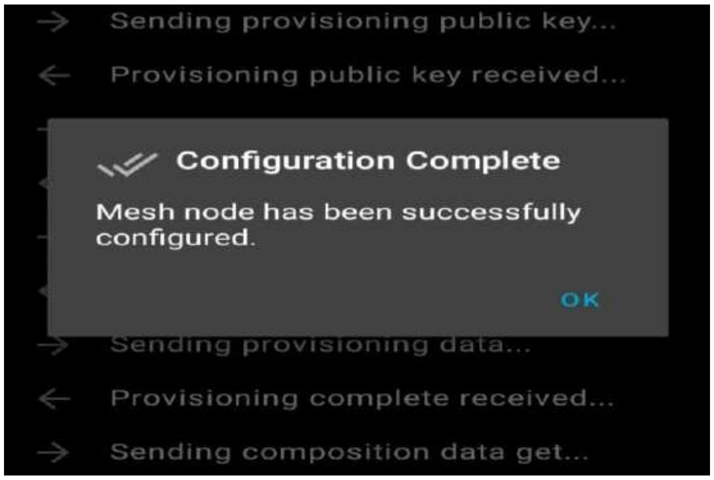
ESP-BLE-MESH
ESP-BLE-MESH

→ Sending provisioning invite...
← Provisioning capabilities received...
→ Sending provisioning start...
→ Sending provisioning public key...
← Provisioning public key received...
→ Sending provisioning confirmation...
← Provisioning confirmation received...
→ Sending provisioning confirmation...
← Provisioning random received...
→ Sending provisioning data...
← Provisioning complete received...

Figure 14: Configuration messages of BLE sensor node

Step: 4

Following the exchange of messages, once the device has been successfully configured, the application will present a message stating **"Configuration Complete."** This indicates that the device has now become an integral and fully functional part of the mesh network, ready to participate in its network operations.



→ Sending provisioning public key...
← Provisioning public key received...

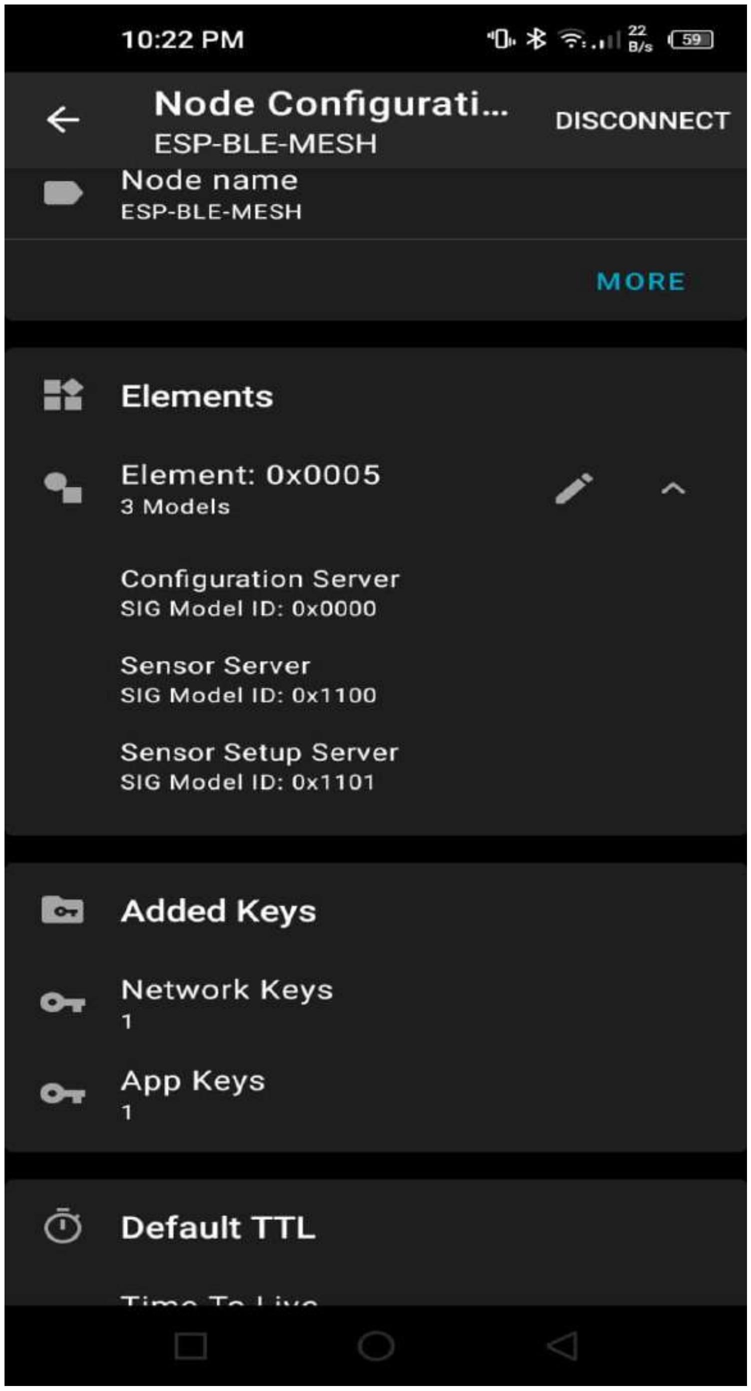
Configuration Complete
Mesh node has been successfully configured.
OK

→ Sending provisioning data...
← Provisioning complete received...
→ Sending composition data get...

Figure 15: Message indicating configuration has completed

Step: 5

Once the device has been successfully added to the network, the application seamlessly transitions you to the node configuration section. Within this section, you have the flexibility to personalize the device name and configure all the elements associated with it. Presently, this device encompasses three models: the Configuration Server, responsible for managing all node configuration messages, the Sensor Server, utilized for retrieving data from the node, and the Sensor Setup Server, designed specifically for configuring and fine-tuning the sensor settings within the node. With these models at your disposal, you have complete control over optimizing and customizing the functionality of the device to suit your specific requirements.



Node name
ESP-BLE-MESH

Element: 0x0005
3 Models

Configuration Server
SIG Model ID: 0x0000

Sensor Server
SIG Model ID: 0x1100

Sensor Setup Server
SIG Model ID: 0x1101

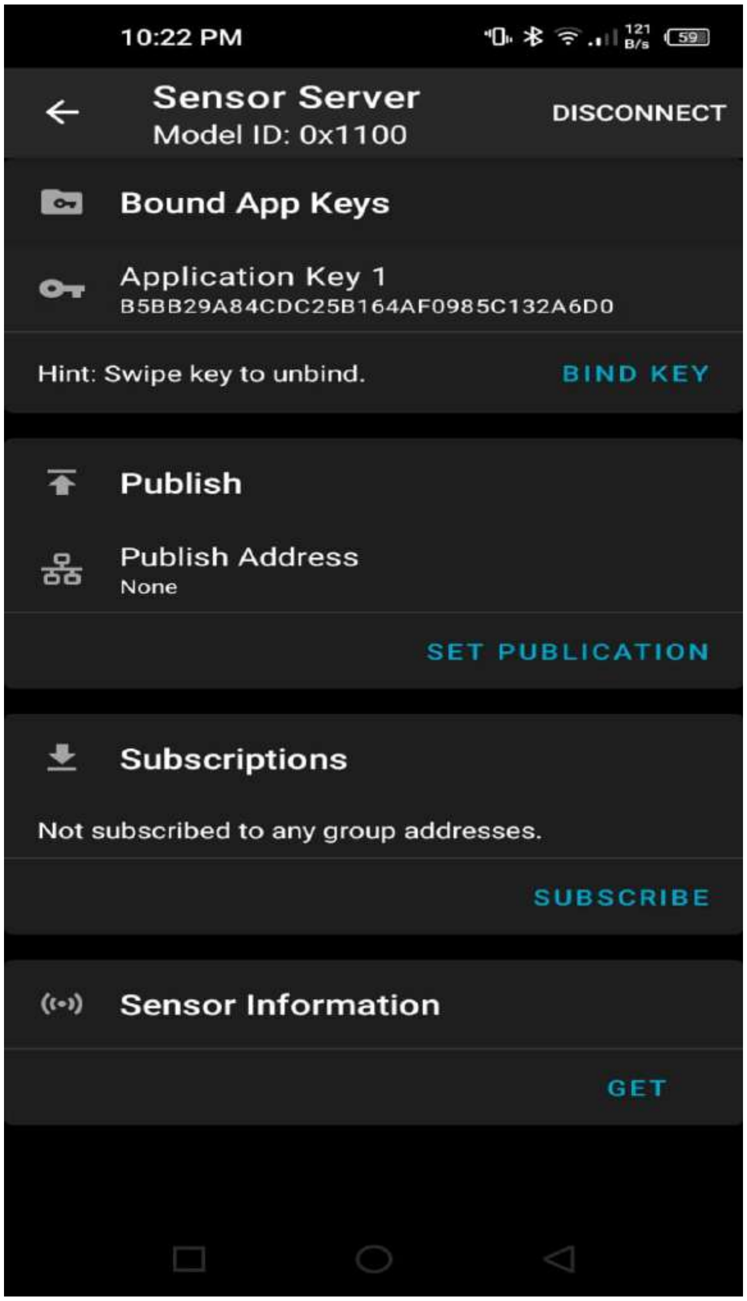
Network Keys
1

App Keys
1

Figure 16: BLE Device model configuration menu

Step: 6

During the node configuration step, simply tap on the Sensor Server to access its settings. From there, you can proceed to bind an application key to it. An application key plays a crucial role in encrypting the data transmitted by the sensor server, ensuring its security and privacy. By associating an application key with the sensor server, you establish a robust encryption mechanism that safeguards the integrity of the data being exchanged within the network.



10:22 PM

Sensor Server
Model ID: 0x1100
DISCONNECT

Bound App Keys

Application Key 1
B5BB29A84CDC25B164AF0985C132A6D0

Hint: Swipe key to unbind.
BIND KEY

Publish

Publish Address
None

SET PUBLICATION

Subscriptions

Not subscribed to any group addresses.

SUBSCRIBE

Sensor Information

GET

Figure 17: BLE sensor server configuration menu

Step: 7

Similarly, during the node configuration process, you can navigate to the Sensor Setup Server and bind an application key to it as well. Binding an application key to the Sensor Setup Server enables you to establish a secure communication channel specifically for setting up and configuring the sensor within the node. This ensures that sensitive configuration data remains encrypted and protected, enhancing the overall security and reliability of the system.

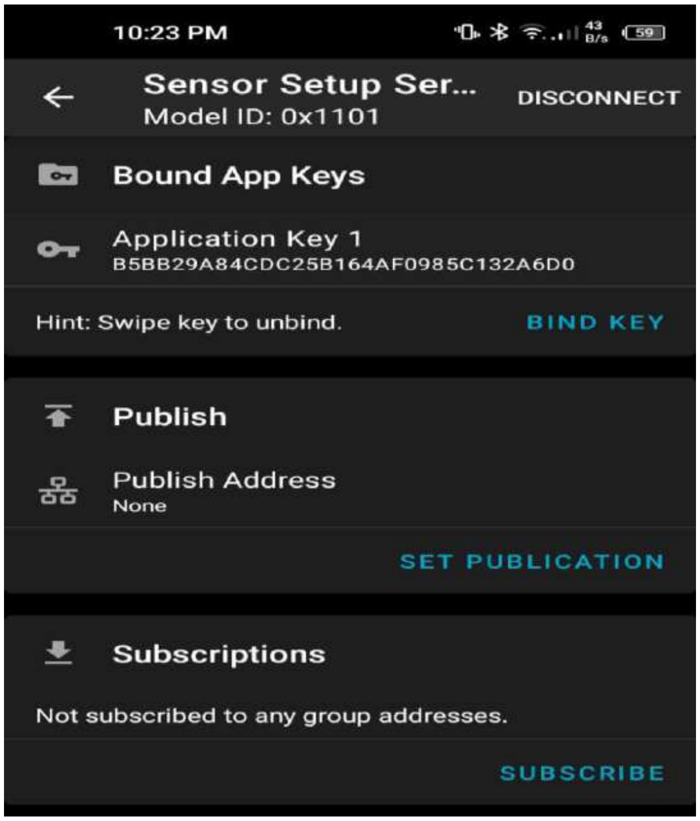


Figure 18: BLE sensor setup server configuration menu

Step: 8

Once all the configuration steps have been successfully completed, you can proceed to tap on the "Get Sensor Information" option to retrieve data from the sensor. Upon selecting this option, the node promptly sends the latest readings of the humidity and temperature parameters. By accessing this information, you gain valuable insights into the current environmental conditions, allowing you to monitor and analyze the sensor data in real-time.

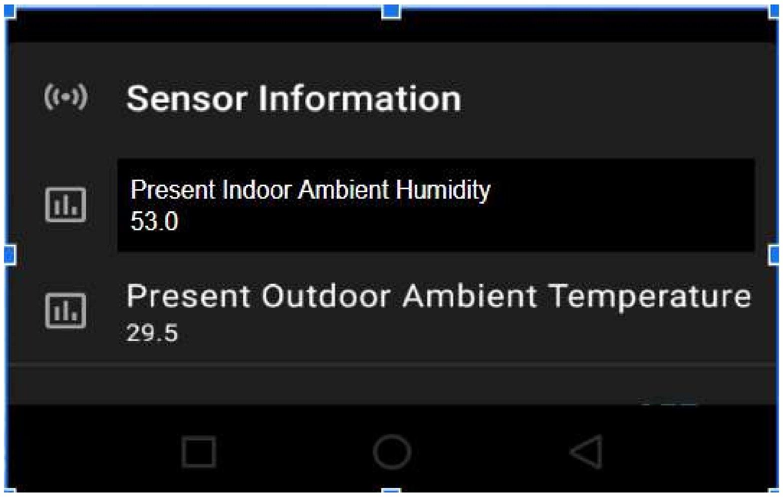

Sensor Information
Present Indoor Ambient Humidity
53.0
Present Outdoor Ambient Temperature
29.5

Figure 19: App dashboard displaying sensor data

12. Problems Faced

We’ve faced the following problems during the development of firmware for sensor nodes:

- One of the problems we faced during the early phase of development was an unexpected chip reset. After conducting research on the underlying real-time operating system (RTOS), it was discovered that the cause of the problem was thread deadlock. We used FreeRTOS for developing the firmware. It is a real-time operating system that can run multiple threads at the same time. For some OS-related tasks, it dispatches a background thread called IDLE TASK at the chip start-up. The job of this thread is to reset the WD timer before expiration. If the WD timer expires, the chip will be reset. This problem arose when we didn’t let the IDLE TASK execute and the WD timer expired. The reason was a deadlock that occurred right after provisioning. When the IDLE TASK tried to access the processor core, it was already occupied by the main thread. It led to the expiration of the WD timer and the chip kept resetting itself. Later on, we identified the flaw in the problem and fixed it.

- Another problem we encountered was the error while interfacing the sensor driver with the BLE sensor model. The BLE mesh stack uses sensor models to automate the process of getting data from the sensor. The official mesh documentation specifies a specific way of interfacing the sensor driver with the corresponding sensor model. The driver code we designed was flawed because it was incompatible with the semantics rules of the BLE sensor model. We solved this problem by rewriting the driver code according to the rules specified in the official mesh documentation.

- Another problem we encountered after the integration of the sensor driver with the sensor model was the buffer overflow. Mesh stack has buffers of specific size which are used to temporarily store the sensor data. The main thread kept pushing the values from the sensor to the buffer which overflowed it. Later on, we solved this problem by limiting the sensor updates and keeping the buffer capacity in check.

- Another problem occurred when receiving the data on the mobile device. The data we received on the nRF Mesh app was corrupted. The reason was that we accidentally misconfigured the sensor property IDs in the firmware program. BLE uses sensor property IDs for adding a contextual meaning to the data of the sensor. Moreover, the property IDs are also used to specify the way a data payload is constructed. If the property IDs are misconfigured then the sensor data won’t be processed properly by the mobile device. We solved this problem by properly configuring the property IDs.

- During the development process of our app, we encountered several challenges that significantly impacted our timeline. One of the major obstacles we faced was the limited knowledge of Android Studio, which caused delays in handling the BLE mesh data. Despite our best efforts, the complexities of integrating the BLE Mesh library into Android Studio proved to be a roadblock. This setback led us to make a strategic decision to shift our development to Flutter, a cross-platform framework. While this change allowed us to overcome the library compatibility issues, it also introduced a new learning curve and necessitated adjustments to our development approach. As a result, even though we have now finalized our app in Flutter, we are still in the process of ironing out the remaining kinks to ensure a smooth and seamless user experience.

13. Personal Growth and the Society

13.1. The Engineer and Society

As we know the engineer is nothing without the society for whom he solves problems. Our project conforms to this belief as it will allow for cheap and effective monitoring of goods in warehouses which will help to remove any chances of spoiled goods and a large disruption in the supply chain. The economy will improve and enable the common folk to obtain daily necessities at all times.

13.2. Compliance with 17 SDGs

Out of the 17 goals our project can achieve 3. The first is Good Health and Well Being which is achieved due to our system eliminating the potential for mold growth. Molds produce spores for reproduction but these spores cause allergic reactions and other lung diseases in humans. Second is Decent Work and Economic Growth, which is possible due to our system monitoring the goods and making sure they do not spoil. If left unchecked this could cause huge losses for a company and disrupt its supply chain leading to a decrease in economic growth. Lastly, we have Responsible Consumption and Production which is possible due to the extremely low power consumption of our system.

13.3. Skills Learned through the Journey

The first skill we learned was the ability to research documents. Over the course of our project, we spent the majority of our time researching all the possible technologies and microcontrollers that we could use. Secondly, we learned management skills as we had to decide how to divide up the work amongst ourselves and make sure to put the right man for each job/task. Thirdly, we learned how to use the ESP IDF which allows us to program the ESP32 microcontrollers and set up BLE. Android studio and java will also be used later on to develop the app for our project’s interface.

14. Conclusion and Future Work

In conclusion, this final year project successfully developed and implemented a Warehouse Climate Monitoring BLE Mesh Network System, showcasing its potential to revolutionize warehouse climate monitoring. The system's utilization of BLE technology and mesh network topology enables real-time data collection from strategically placed sensor nodes, surpassing traditional WiFi-based alternatives. The project demonstrated the advantages of the BLE Mesh Network System, including reduced hardware requirements, scalability, and lower power consumption.

Moving forward, there are several avenues for future work and enhancements to further improve the system. Firstly, focusing on enhancing the user experience, the app will undergo continuous improvements to make it more user-friendly. This includes refining the user interface, simplifying the setup process, and optimizing the app's overall performance.

Additionally, the inclusion of a buzzer on each module presents an opportunity to enhance the system's alerting capabilities. By incorporating an alarm functionality, the system will be able to emit audible alerts in addition to the existing notifications on the user's phone, ensuring immediate attention and response to critical temperature or humidity deviations.

Expanding the sensor capabilities is another aspect of future work. The addition of an oxygen sensor to the existing temperature and humidity sensors will enable comprehensive monitoring of the warehouse environment, providing valuable insights into potential oxygen-related risks and ensuring the safety of stored goods.

Furthermore, implementing a database will allow for the storage and retrieval of historical data collected by the system. By integrating a database system, users will be able to access and analyze older data, generate graphs and reports, and gain valuable insights into trends and patterns over time. This will facilitate informed decision-making and enable better management of warehouse climate conditions.

Overall, these future enhancements and developments will further solidify the Warehouse Climate Monitoring BLE Mesh Network System's effectiveness and usability, contributing to improved warehouse climate monitoring and supply chain management practices.

15. Codes

15.1. ESP code

```
#include <stdio.h>
#include <string.h>

#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_ble_mesh_defs.h"
#include "esp_ble_mesh_common_api.h"
#include "esp_ble_mesh_networking_api.h"
#include "esp_ble_mesh_provisioning_api.h"
#include "esp_ble_mesh_config_model_api.h"
#include "esp_ble_mesh_sensor_model_api.h"

// contains bluetooth stack initialization code
#include "BLE_stack_init.h"
#include "sensor_driver.h"

#define NODE_CID 0x02E5
#define TAG "Meshsetup"

// sensor descriptor related information
// property id for present outdoor ambient temperature
#define HUM_SENSOR_PROP_ID 0x0056
#define TEMP_SENSOR_PROP_ID 0x005B

// property id 0x0076 doesn't works
//#define HUM_SENSOR_PROP_ID 0x0076
```

```
#define SENSOR_POSITIVE_TOLERANCE
ESP_BLE_MESH_SENSOR_UNSPECIFIED_POS_TOLERANCE
#define SENSOR_NEGATIVE_TOLERANCE
ESP_BLE_MESH_SENSOR_UNSPECIFIED_NEG_TOLERANCE
#define SENSOR_SAMPLE_FUNCTION ESP_BLE_MESH_SAMPLE_FUNC_UNSPECIFIED
#define SENSOR_MEASURE_PERIOD ESP_BLE_MESH_SENSOR_NOT_APPL_MEASURE_PERIOD
#define SENSOR_UPDATE_INTERVAL ESP_BLE_MESH_SENSOR_NOT_APPL_UPDATE_INTERVAL


static uint8_t dev_uuid[ESP_BLE_MESH_OCTET16_LEN] = {0x32, 0x10};

// Define a static net_buf_simple variable.
// This is a helper macro which is used to define a static net_buf_simple object.
// Parameters   NET_BUF_SIMPLE_DEFINE_STATIC(_name, _size)
// _name: Name of the net_buf_simple object.
// _size: Maximum data storage for the buffer.

NET_BUF_SIMPLE_DEFINE_STATIC(humidity_sensor_data, 1);
NET_BUF_SIMPLE_DEFINE_STATIC(temperature_sensor_data, 1);

// parameters of sensor state
static esp_ble_mesh_sensor_state_t sensor_state[2] = {
  [0] = {
    .sensor_property_id = HUM_SENSOR_PROP_ID,
    .descriptor.positive_tolerance = SENSOR_POSITIVE_TOLERANCE,
    .descriptor.negative_tolerance = SENSOR_NEGATIVE_TOLERANCE,
    .descriptor.sampling_function = SENSOR_SAMPLE_FUNCTION,
    .descriptor.measure_period = SENSOR_MEASURE_PERIOD,
    .descriptor.update_interval = SENSOR_UPDATE_INTERVAL,
    .sensor_data.format = ESP_BLE_MESH_SENSOR_DATA_FORMAT_A,
    .sensor_data.length = 0, /* 0 represents the length is 1 */
    .sensor_data.raw_value = &humidity_sensor_data,
  },
  [1] = {
    .sensor_property_id = TEMP_SENSOR_PROP_ID,
    .descriptor.positive_tolerance = SENSOR_POSITIVE_TOLERANCE,
    .descriptor.negative_tolerance = SENSOR_NEGATIVE_TOLERANCE,
    .descriptor.sampling_function = SENSOR_SAMPLE_FUNCTION,
    .descriptor.measure_period = SENSOR_MEASURE_PERIOD,
    .descriptor.update_interval = SENSOR_UPDATE_INTERVAL,
    .sensor_data.format = ESP_BLE_MESH_SENSOR_DATA_FORMAT_A,
    .sensor_data.length = 0, // 0 represents the length is 1
    .sensor_data.raw_value = &temperature_sensor_data,
  },
  };

/*
  Define a model publication context
```

```
   20 octets is large enough to hold two Sensor Descriptor state values. */
ESP_BLE_MESH_MODEL_PUB_DEFINE(sensor_pub, 20, ROLE_NODE);

static esp_ble_mesh_sensor_srv_t sensor_server = {
   .rsp_ctrl.get_auto_rsp = ESP_BLE_MESH_SERVER_RSP_BY_APP,
   .rsp_ctrl.set_auto_rsp = ESP_BLE_MESH_SERVER_RSP_BY_APP,
   .state_count = ARRAY_SIZE(sensor_state),
   .states = sensor_state,
};

ESP_BLE_MESH_MODEL_PUB_DEFINE(sensor_setup_pub, 20, ROLE_NODE);
static esp_ble_mesh_sensor_setup_srv_t sensor_setup_server = {
   .rsp_ctrl.get_auto_rsp = ESP_BLE_MESH_SERVER_RSP_BY_APP,
   .rsp_ctrl.set_auto_rsp = ESP_BLE_MESH_SERVER_RSP_BY_APP,
   .state_count = ARRAY_SIZE(sensor_state),
   .states = sensor_state,
};

// Configuration Server
static esp_ble_mesh_cfg_srv_t configuration_server = {
   .relay = ESP_BLE_MESH_RELAY_ENABLED,
   .beacon = ESP_BLE_MESH_BEACON_ENABLED,
   .gatt_proxy = ESP_BLE_MESH_GATT_PROXY_ENABLED,
   //.gatt_proxy = ESP_BLE_MESH_GATT_PROXY_DISABLED,
   .default_ttl = 10,
   .net_transmit = ESP_BLE_MESH_TRANSMIT(2, 20), // Total of 3 transmissions after every
20ms
   .relay_retransmit = ESP_BLE_MESH_TRANSMIT(2, 20),
};

// Root models
// all the server/client models are initialized here
static esp_ble_mesh_model_t root_models[] = {
   ESP_BLE_MESH_MODEL_CFG_SRV(&configuration_server), // define a configuration server
model
   ESP_BLE_MESH_MODEL_SENSOR_SRV(&sensor_pub, &sensor_server),
   ESP_BLE_MESH_MODEL_SENSOR_SETUP_SRV(&sensor_setup_pub, &sensor_setup_server),
};

// Element definition
static esp_ble_mesh_elem_t elements[] = {
   // Helper to define a BLE Mesh element within an array.
   // In case the element has no SIG or Vendor models, the helper
   // macro ESP_BLE_MESH_MODEL_NONE can be given instead.
   ESP_BLE_MESH_ELEMENT(0, root_models, ESP_BLE_MESH_MODEL_NONE),
};

// Node composition
```

```
static esp_ble_mesh_comp_t composition = {
   .cid = NODE_CID,
   .elements = elements,
   .element_count = ARRAY_SIZE(elements),
};

// contains provisioning related data
static esp_ble_mesh_prov_t prov = {
   .uuid = dev_uuid,
};

esp_ble_mesh_model_t *get_sensor_server_model(void)
{
   return &root_models[1];
}


static esp_err_t initialize_mesh(void)
{
   esp_err_t error_code = ESP_OK;
   ble_mesh_get_dev_uuid(dev_uuid);
   error_code = esp_ble_mesh_init(&prov, &composition);
   error_code = esp_ble_mesh_node_prov_enable(ESP_BLE_MESH_PROV_GATT |
ESP_BLE_MESH_PROV_ADV);
   return error_code;
}

static void mesh_provisioning_callback(esp_ble_mesh_prov_cb_event_t event,
esp_ble_mesh_prov_cb_param_t *param)
{
   switch (event)
   {
   case ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT:
      ESP_LOGI(TAG, "Provisioning complete");
      break;
   case ESP_BLE_MESH_NODE_PROV_RESET_EVT:
      // All provisioning information in this node will be deleted and the node needs to be
reprovisioned. The
      // API function esp_ble_mesh_node_prov_enable() needs to be called to start a new
provisioning procedure.
      esp_ble_mesh_node_local_reset();
   default:
      break;
   }
}

static void config_server_callback(esp_ble_mesh_cfg_server_cb_event_t event,
esp_ble_mesh_cfg_server_cb_param_t *param)
```

```
{
   if (event == ESP_BLE_MESH_CFG_SERVER_STATE_CHANGE_EVT)
   {
      switch (param->ctx.recv_op)
      {
      case ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD:
         ESP_LOGI(TAG, "config: app key added");
         break;
      default:
         break;
      }
   }
}

/**
 * : after variable name is used for specifying the number of bit fields to use for variable
 * ((packed)) means it will use the smallest possible space for struct Ball - i.e. it will cram fields
together without padding
 */
struct sensor_descriptor
{
   uint16_t sensor_prop_id;
   uint32_t pos_tolerance : 12,
      neg_tolerance : 12,
      sample_func : 8;
   uint8_t measure_period;
   uint8_t update_interval;
} __attribute__((packed));


/**
 * @brief Sends sensor descriptor state data in response to sensor descriptor get message
 * Sensor Descriptor Status message ==> Sensor Descriptor Get message
 * @param param
 * Sensor Server Model callback parameters
 * Public Members of esp_ble_mesh_sensor_server_cb_param_t:
 *     esp_ble_mesh_model_t *model - Pointer to Sensor Server Models
 *     esp_ble_mesh_msg_ctx_t ctx - Context of the received messages
 *     esp_ble_mesh_sensor_server_cb_value_t value - Value of the received Sensor Messages
 *
 * @details
 * Message Types:
 * 1) Sensor Descriptor Get
 * Sensor Descriptor Get is an acknowledged message used to get the Sensor Descriptor state
of all
 * sensors within an element.
 *
 * 2) Sensor Descriptor Status
 * The Sensor Descriptor Status is an unacknowledged message used to report a sequence of
the
 * Sensor Descriptor states of an element.
 */
static void
ble_mesh_send_sensor_descriptor_status(esp_ble_mesh_sensor_server_cb_param_t *param)
{
    struct sensor_descriptor descriptor = {0}; // initialize all bits to zero
    uint8_t *status = NULL;     // pointer to memory allocated to descriptor state struct
    uint16_t length = 0;    // length of sensor descriptor state
    esp_err_t err;
    int i;


    status = calloc(1, ARRAY_SIZE(sensor_state) * ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN);
    if (!status)
    {
        ESP_LOGE(TAG, "No memory for sensor descriptor status!");
        return;
    }

    if (param->value.get.sensor_descriptor.op_en == false)
    {
        /* Mesh Model Spec:
         * Upon receiving a Sensor Descriptor Get message with the Property ID field
         * omitted, the Sensor Server shall respond with a Sensor Descriptor Status
         * message containing the Sensor Descriptor states for all sensors within the
         * Sensor Server.
         */
        for (i = 0; i < ARRAY_SIZE(sensor_state); i++)
        {
            descriptor.sensor_prop_id = sensor_state[i].sensor_property_id;
            descriptor.pos_tolerance = sensor_state[i].descriptor.positive_tolerance;
            descriptor.neg_tolerance = sensor_state[i].descriptor.negative_tolerance;
            descriptor.sample_func = sensor_state[i].descriptor.sampling_function;
            descriptor.measure_period = sensor_state[i].descriptor.measure_period;
            descriptor.update_interval = sensor_state[i].descriptor.update_interval;

            memcpy(status + length, &descriptor, ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN);
            length += ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN;
        }
        goto send;
    }
    for (i = 0; i < ARRAY_SIZE(sensor_state); i++)
    {
        /* Send descriptor state when the get message contains the valid sensor property ID */
        if (param->value.get.sensor_descriptor.property_id == sensor_state[i].sensor_property_id)
        {
            descriptor.sensor_prop_id = sensor_state[i].sensor_property_id;
            descriptor.pos_tolerance = sensor_state[i].descriptor.positive_tolerance;
            descriptor.neg_tolerance = sensor_state[i].descriptor.negative_tolerance;
            descriptor.sample_func = sensor_state[i].descriptor.sampling_function;
            descriptor.measure_period = sensor_state[i].descriptor.measure_period;
            descriptor.update_interval = sensor_state[i].descriptor.update_interval;
            memcpy(status, &descriptor, ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN);
            length = ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN;
            goto send;
        }
    }
    /* Mesh Model Spec:
     * When a Sensor Descriptor Get message that identifies a sensor descriptor
     * property that does not exist on the element, the Descriptor field shall
     * contain the requested Property ID value and the other fields of the Sensor
     * Descriptor state shall be omitted.
     */
    memcpy(status, &param->value.get.sensor_descriptor.property_id,
ESP_BLE_MESH_SENSOR_PROPERTY_ID_LEN);
    length = ESP_BLE_MESH_SENSOR_PROPERTY_ID_LEN;

send:
    ESP_LOG_BUFFER_HEX("Sensor Descriptor", status, length);    // Log a buffer of hex bytes at
Info level.

    err = esp_ble_mesh_server_model_send_msg(param->model, &param->ctx,
                                    ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS, length,
status);
    if (err != ESP_OK)
    {
        ESP_LOGE(TAG, "Failed to send Sensor Descriptor Status");
    }
    free(status);   // release the memory held by descriptor state structure
}

static void ble_mesh_send_sensor_cadence_status(esp_ble_mesh_sensor_server_cb_param_t
*param)
{
    esp_err_t err;

    /* Sensor Cadence state is not supported currently. */
    err = esp_ble_mesh_server_model_send_msg(param->model, &param->ctx,
                                    ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_STATUS,
                                    ESP_BLE_MESH_SENSOR_PROPERTY_ID_LEN,
                                    (uint8_t *)&param->value.get.sensor_cadence.property_id);
    if (err != ESP_OK)
    {
        ESP_LOGE(TAG, "Failed to send Sensor Cadence Status");
    }
}

static void ble_mesh_send_sensor_settings_status(esp_ble_mesh_sensor_server_cb_param_t
*param)
{
    esp_err_t err;

    /* Sensor Setting state is not supported currently. */
    err = esp_ble_mesh_server_model_send_msg(param->model, &param->ctx,
                                    ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_STATUS,
                                    ESP_BLE_MESH_SENSOR_PROPERTY_ID_LEN,
                                    (uint8_t *)&param->value.get.sensor_settings.property_id);
    if (err != ESP_OK)
    {
        ESP_LOGE(TAG, "Failed to send Sensor Settings Status");
    }
}

struct sensor_setting
{
    uint16_t sensor_prop_id;
    uint16_t sensor_setting_prop_id;
} __attribute__((packed));

static void ble_mesh_send_sensor_setting_status(esp_ble_mesh_sensor_server_cb_param_t
*param)
{
    struct sensor_setting setting = {0};
    esp_err_t err;

    /* Mesh Model Spec:
     * If the message is sent as a response to the Sensor Setting Get message or
     * a Sensor Setting Set message with an unknown Sensor Property ID field or
     * an unknown Sensor Setting Property ID field, the Sensor Setting Access
     * field and the Sensor Setting Raw field shall be omitted.
     */

    setting.sensor_prop_id = param->value.get.sensor_setting.property_id;
    setting.sensor_setting_prop_id = param->value.get.sensor_setting.setting_property_id;

    err = esp_ble_mesh_server_model_send_msg(param->model, &param->ctx,
                                    ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_STATUS,
                                    sizeof(setting), (uint8_t *)&setting);
    if (err != ESP_OK)
    {
        ESP_LOGE(TAG, "Failed to send Sensor Setting Status");
    }
```

```
}

static uint16_t ble_mesh_get_sensor_data(esp_ble_mesh_sensor_state_t *state, uint8_t
*data)
{
   uint8_t mpid_len = 0, data_len = 0;
   uint32_t mpid = 0;

   if (state == NULL || data == NULL)
   {
      ESP_LOGE(TAG, "%s, Invalid parameter", __func__);
      return 0;
   }

   if (state->sensor_data.length == ESP_BLE_MESH_SENSOR_DATA_ZERO_LEN)
   {
      /* For zero-length sensor data, the length is 0x7F, and the format is Format B. */
      mpid = ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID(state->sensor_data.length,
state->sensor_property_id);
      mpid_len = ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN;
      data_len = 0;
   }
   else
   {
      if (state->sensor_data.format == ESP_BLE_MESH_SENSOR_DATA_FORMAT_A)
      {
         mpid = ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID(state->sensor_data.length,
state->sensor_property_id);
         mpid_len = ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID_LEN;
      }
      else
      {
         mpid = ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID(state->sensor_data.length,
state->sensor_property_id);
         mpid_len = ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN;
      }
      /* Use "state->sensor_data.length + 1" because the length of sensor data is zero-based. */
      data_len = state->sensor_data.length + 1;
   }

   // ???
   memcpy(data, &mpid, mpid_len);
   memcpy(data + mpid_len, state->sensor_data.raw_value->data, data_len);

   return (mpid_len + data_len);
}
```

```
static void ble_mesh_send_sensor_status(esp_ble_mesh_sensor_server_cb_param_t *param)
{
   uint8_t *status = NULL;
   uint16_t buf_size = 0;
   uint16_t length = 0;
   uint32_t mpid = 0;
   esp_err_t err;
   int i;

   /**
    * Sensor Data state from Mesh Model Spec
    * |--------Field--------|-Size (octets)-|------------------------Notes-------------------------|
    * |----Property ID 1----|-------2-------|--ID of the 1st device property of the sensor---------|
    * |-----Raw Value 1-----|----variable---|--Raw Value field defined by the 1st device property--
|
    * |----Property ID 2----|-------2-------|--ID of the 2nd device property of the sensor---------|
    * |-----Raw Value 2-----|----variable---|--Raw Value field defined by the 2nd device property--
|
    * | ...... |
    * |----Property ID n----|-------2-------|--ID of the nth device property of the sensor---------|
    * |-----Raw Value n-----|----variable---|--Raw Value field defined by the nth device property--
|
    */

   // find the buffer size for sensor status
   for (i = 0; i < ARRAY_SIZE(sensor_state); i++)
   {
      esp_ble_mesh_sensor_state_t *state = &sensor_state[i];
      if (state->sensor_data.length == ESP_BLE_MESH_SENSOR_DATA_ZERO_LEN)
      {
         buf_size += ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN;
      }
      else
      {
         /* Use "state->sensor_data.length + 1" because the length of sensor data is zero-based.
*/
         if (state->sensor_data.format == ESP_BLE_MESH_SENSOR_DATA_FORMAT_A)
         {
            buf_size += ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID_LEN + state-
>sensor_data.length + 1;
         }
         else
         {
            buf_size += ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN + state-
>sensor_data.length + 1;
         }
      }
   }
```

```
    status = calloc(1, buf_size);
    if (!status)
    {
        ESP_LOGE(TAG, "No memory for sensor status!");
        return;
    }

    if (param->value.get.sensor_data.op_en == false)
    {
        /* Mesh Model Spec:
         * If the message is sent as a response to the Sensor Get message, and if the
         * Property ID field of the incoming message is omitted, the Marshalled Sensor
         * Data field shall contain data for all device properties within a sensor.
         */
        for (i = 0; i < ARRAY_SIZE(sensor_state); i++)
        {
            length += ble_mesh_get_sensor_data(&sensor_state[i], status + length);
        }
        goto send;
    }

    /* Mesh Model Spec:
     * Otherwise, the Marshalled Sensor Data field shall contain data for the requested
     * device property only.
     */
    for (i = 0; i < ARRAY_SIZE(sensor_state); i++)
    {
        if (param->value.get.sensor_data.property_id == sensor_state[i].sensor_property_id)
        {
            length = ble_mesh_get_sensor_data(&sensor_state[i], status);
            goto send;
        }
    }

    /* Mesh Model Spec:
     * Or the Length shall represent the value of zero and the Raw Value field shall
     * contain only the Property ID if the requested device property is not recognized
     * by the Sensor Server.
     */
    mpid =
ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID(ESP_BLE_MESH_SENSOR_DATA_ZERO_LEN,
                                        param->value.get.sensor_data.property_id);
    memcpy(status, &mpid, ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN);
    length = ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN;

send:
    ESP_LOG_BUFFER_HEX("Sensor Data", status, length);
    err = esp_ble_mesh_server_model_send_msg(param->model, &param->ctx,
                                    ESP_BLE_MESH_MODEL_OP_SENSOR_STATUS, length, status);
    if (err != ESP_OK)
    {
        ESP_LOGE(TAG, "Failed to send Sensor Status");
    }
    free(status);
}

static void publish_data()
{
    uint8_t *status = NULL;
    uint16_t buf_size = 0;
    uint16_t length = 0;
    esp_err_t err;
    int i;

    // find the buffer size for sensor status
    for (i = 0; i < ARRAY_SIZE(sensor_state); i++)
    {
        esp_ble_mesh_sensor_state_t *state = &sensor_state[i];
        if (state->sensor_data.length == ESP_BLE_MESH_SENSOR_DATA_ZERO_LEN)
        {
            buf_size += ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN;
        }
        else
        {
            /* Use "state->sensor_data.length + 1" because the length of sensor data is zero-based.
*/
            if (state->sensor_data.format == ESP_BLE_MESH_SENSOR_DATA_FORMAT_A)
            {
                buf_size += ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID_LEN + state-
>sensor_data.length + 1;
            }
            else
            {
                buf_size += ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN + state-
>sensor_data.length + 1;
            }
        }
    }

    status = calloc(1, buf_size);

    if (!status)
    {
        ESP_LOGE(TAG, "No memory for sensor status!");
        return;
    }

    for (i = 0; i < ARRAY_SIZE(sensor_state); i++)
    {
        length += ble_mesh_get_sensor_data(&sensor_state[i], status + length);
    }

    ESP_LOG_BUFFER_HEX("Published Sensor Data", status, length);

    esp_ble_mesh_model_t *model = get_sensor_server_model();
    err = esp_ble_mesh_model_publish(model, ESP_BLE_MESH_MODEL_OP_SENSOR_STATUS,
length, status, ROLE_NODE);
    if (err != ESP_OK)
    {
        ESP_LOGE(TAG, "Failed to publish Sensor Status");
    }
    free(status);
}




static void ble_mesh_sensor_server_cb(esp_ble_mesh_sensor_server_cb_event_t event,
                                      esp_ble_mesh_sensor_server_cb_param_t *param)
{
    ESP_LOGI(TAG, "Sensor server, event %d, src 0x%04x, dst 0x%04x, model_id 0x%04x",
             event, param->ctx.addr, param->ctx.recv_dst, param->model->model_id);

    switch (event)
    {
    case ESP_BLE_MESH_SENSOR_SERVER_RECV_GET_MSG_EVT:
        switch (param->ctx.recv_op)
        {
        case ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET:
            ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET");
            ble_mesh_send_sensor_descriptor_status(param);
            break;
        case ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET:
            ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET");
            ble_mesh_send_sensor_cadence_status(param);
            break;
        case ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET:
            ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET");
            ble_mesh_send_sensor_settings_status(param);
            break;
        case ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_GET:
            ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET");
        ble_mesh_send_sensor_setting_status(param);
        break;
    case ESP_BLE_MESH_MODEL_OP_SENSOR_GET:
        ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_GET");
        ble_mesh_send_sensor_status(param);
        break;
    default:
        ESP_LOGE(TAG, "Unknown Sensor Get opcode 0x%04x", param->ctx.recv_op);
        return;
    }
    break;
  case ESP_BLE_MESH_SENSOR_SERVER_RECV_SET_MSG_EVT:
    switch (param->ctx.recv_op)
    {
    case ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET:
        ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET");
        ble_mesh_send_sensor_cadence_status(param);
        break;
    case ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK:
        ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK");
        break;
    case ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET:
        ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET");
        ble_mesh_send_sensor_setting_status(param);
        break;
    case ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET_UNACK:
        ESP_LOGI(TAG, "ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET_UNACK");
        break;
    default:
        ESP_LOGE(TAG, "Unknown Sensor Set opcode 0x%04x", param->ctx.recv_op);
        break;
    }
    break;
  default:
    ESP_LOGE(TAG, "Unknown Sensor Server event %d", event);
    break;
  }
}

void app_main()
{

  nvs_flash_erase();
  nvs_flash_init();


  BLE_stack_init();
```

```
   esp_ble_mesh_register_prov_callback(mesh_provisioning_callback);
   esp_ble_mesh_register_config_server_callback(config_server_callback);
   esp_ble_mesh_register_sensor_server_callback(ble_mesh_sensor_server_cb);
   initialize_mesh();

   net_buf_simple_add_u8(&humidity_sensor_data, 2*humidity);
   net_buf_simple_add_u8(&temperature_sensor_data, 2*temperature);

   while (true)
   {
      vTaskDelay(1000 / portTICK_PERIOD_MS);
      get_temperature_and_humidity();
      printf("Humidity: %f%% Temp: %fC\n", humidity, temperature);


      net_buf_simple_pull(&humidity_sensor_data, sizeof(uint8_t));
      net_buf_simple_pull(&temperature_sensor_data, sizeof(uint8_t));

      net_buf_simple_add_u8(&humidity_sensor_data, 2*humidity);
      net_buf_simple_add_u8(&temperature_sensor_data, 2*temperature);

      publish_data();
   }
}
```

15.2. Flutter Application Code

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

class SendConfigModelPublicationAdd extends StatefulWidget {
  final MeshManagerApi meshManagerApi;

  const SendConfigModelPublicationAdd(this.meshManagerApi, {Key? key}) : super(key: key);

  @override
  State<SendConfigModelPublicationAdd> createState() => _SendConfigModelPublicationAddState();
}

class _SendConfigModelPublicationAddState extends State<SendConfigModelPublicationAdd> {
  late int selectedElementAddress;
  late int selectedModelId;
  late int selectedSubscriptionAddress;
  @override
  Widget build(BuildContext context) {
    return ExpansionTile(
      title: const Text('Send a config model Publication add'),
      children: <Widget>[
        TextField(
          decoration: const InputDecoration(hintText: 'Element Address'),
          onChanged: (text) {
            selectedElementAddress = int.parse(text);
          },
        ),
        TextField(
          decoration: const InputDecoration(hintText: 'Model id'),
          onChanged: (text) {
            selectedModelId = int.parse(text);
          },
        ),
        TextField(
          decoration: const InputDecoration(hintText: 'publication address'),
          onChanged: (text) {
            selectedSubscriptionAddress = int.parse(text);
          },
        ),
        TextButton(
          onPressed: () async {
            final scaffoldMessenger = ScaffoldMessenger.of(context);
            try {
              await widget.meshManagerApi
                  .sendConfigModelPublicationSet(selectedElementAddress, selectedSubscriptionAddress,
selectedModelId)
                  .timeout(const Duration(seconds: 40));
              scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
            } on TimeoutException catch (_) {
              scaffoldMessenger.showSnackBar(const SnackBar(content: Text('Board didn\'t respond')));
            } on PlatformException catch (e) {
              scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
            } catch (e) {
              scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
            }
          },
          child: const Text('Send config message'),
        )
      ],
    );
  }
}
```

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

class SendConfigModelSubscriptionAdd extends StatefulWidget {
  final MeshManagerApi meshManagerApi;

  const SendConfigModelSubscriptionAdd(this.meshManagerApi, {Key? key}) : super(key: key);

  @override
  State<SendConfigModelSubscriptionAdd> createState() =>
_SendConfigModelSubscriptionAddState();
}

class _SendConfigModelSubscriptionAddState extends State<SendConfigModelSubscriptionAdd> {
  late int selectedElementAddress;
  late int selectedModelId;
  late int selectedSubscriptionAddress;

  @override
  Widget build(BuildContext context) {
    return ExpansionTile(
      title: const Text('Send a config model Subscription add'),
      children: <Widget>[
        TextField(
          decoration: const InputDecoration(hintText: 'Element Address'),
          onChanged: (text) {
            selectedElementAddress = int.parse(text);
          },
        ),
        TextField(
          decoration: const InputDecoration(hintText: 'Model id'),
          onChanged: (text) {
            selectedModelId = int.parse(text);
          },
        ),
        TextField(
          decoration: const InputDecoration(hintText: 'subscription address'),
          onChanged: (text) {
            selectedSubscriptionAddress = int.parse(text);
          },
        ),
        TextButton(
          onPressed: () async {
            final scaffoldMessenger = ScaffoldMessenger.of(context);
            try {
```

```
            await widget.meshManagerApi
                .sendConfigModelSubscriptionAdd(selectedElementAddress, selectedSubscriptionAddress,
selectedModelId)
                .timeout(const Duration(seconds: 40));
            scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
          } on TimeoutException catch (_) {
            scaffoldMessenger.showSnackBar(const SnackBar(content: Text('Board didn\'t respond')));
          } on PlatformException catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
          } catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
          }
        },
        child: const Text('Send config message'),
      )
    ],
  );
 }
}
```

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

class SendDeprovisioning extends StatefulWidget {
  final MeshManagerApi meshManagerApi;

  const SendDeprovisioning({Key? key, required this.meshManagerApi}) : super(key: key);

  @override
  State<SendDeprovisioning> createState() => _SendDeprovisioningState();
}

class _SendDeprovisioningState extends State<SendDeprovisioning> {
  int? selectedElementAddress;

  @override
  Widget build(BuildContext context) {
    return ExpansionTile(
      key: const ValueKey('module-send-deprovisioning-form'),
      title: const Text('Send a deprovisioning'),
      children: <Widget>[
        TextField(
          key: const ValueKey('module-send-deprovisioning-address'),
          decoration: const InputDecoration(hintText: 'Element Address'),
          onChanged: (text) {
```

```
          setState(() {
            selectedElementAddress = int.tryParse(text);
          });
        },
      ),
      TextButton(
        onPressed: selectedElementAddress != null
            ? () async {
                final scaffoldMessenger = ScaffoldMessenger.of(context);
                final node = await
widget.meshManagerApi.meshNetwork!.getNode(selectedElementAddress!);
                final nodes = await widget.meshManagerApi.meshNetwork!.nodes;
                try {
                  final provisionedNode = nodes.firstWhere((element) => element.uuid == node!.uuid);
                  await widget.meshManagerApi.deprovision(provisionedNode).timeout(const
Duration(seconds: 40));
                  scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
                } on TimeoutException catch (_) {
                  scaffoldMessenger.showSnackBar(const SnackBar(content: Text('Board didn\'t respond')));
                } on PlatformException catch (e) {
                  scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
                } on StateError catch (_) {
                  scaffoldMessenger.showSnackBar(const SnackBar(content: Text('No node found with this
uuid')));
                } catch (e) {
                  scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
                }
              }
            : null,
        child: const Text('Send node reset'),
      )
    ],
  );
 }
}
```

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

class SendGenericLevel extends StatefulWidget {
  final MeshManagerApi meshManagerApi;

  const SendGenericLevel({Key? key, required this.meshManagerApi}) : super(key: key);

  @override
  State<SendGenericLevel> createState() => _SendGenericLevelState();
}

class _SendGenericLevelState extends State<SendGenericLevel> {
  int? selectedElementAddress;

  int? selectedLevel;

  @override
  Widget build(BuildContext context) {
    return ExpansionTile(
      key: const ValueKey('module-send-generic-level-form'),
      title: const Text('Send a generic level set'),
      children: <Widget>[
        TextField(
          key: const ValueKey('module-send-generic-level-address'),
          decoration: const InputDecoration(hintText: 'Element Address'),
          onChanged: (text) {
            selectedElementAddress = int.parse(text);
          },
        ),
        TextField(
          key: const ValueKey('module-send-generic-level-value'),
          decoration: const InputDecoration(hintText: 'Level Value'),
          onChanged: (text) {
            setState(() {
              selectedLevel = int.tryParse(text);
            });
          },
        ),
        TextButton(
          onPressed: selectedLevel != null
              ? () async {
                  final scaffoldMessenger = ScaffoldMessenger.of(context);
                  debugPrint('send level $selectedLevel to $selectedElementAddress');
                  try {
                    await widget.meshManagerApi
                        .sendGenericLevelSet(selectedElementAddress!, selectedLevel!)
                        .timeout(const Duration(seconds: 40));
                    scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
                  } on TimeoutException catch (_) {
                    scaffoldMessenger.showSnackBar(const SnackBar(content: Text('Board didn\'t respond')));
                  } on PlatformException catch (e) {
                    scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
                  } catch (e) {
                    scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
                  }
                }
            : null,
          child: const Text('Send level'),
        )
      ],
    );
  }
}
```

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

class SendGenericOnOff extends StatefulWidget {
  final MeshManagerApi meshManagerApi;

  const SendGenericOnOff({Key? key, required this.meshManagerApi}) : super(key: key);

  @override
  State<SendGenericOnOff> createState() => _SendGenericOnOffState();
}

class _SendGenericOnOffState extends State<SendGenericOnOff> {
  int? selectedElementAddress;

  bool onOff = false;

  @override
  Widget build(BuildContext context) {
    return ExpansionTile(
      key: const ValueKey('module-send-generic-on-off-form'),
      title: const Text('Send a generic On Off set'),
      children: <Widget>[
        TextField(
          key: const ValueKey('module-send-generic-on-off-address'),
          decoration: const InputDecoration(hintText: 'Element Address'),
          onChanged: (text) {
            setState(() {
              selectedElementAddress = int.tryParse(text);
            });
          },
        ),
        Checkbox(
          key: const ValueKey('module-send-generic-on-off-value'),
          value: onOff,
          onChanged: (value) {
            setState(() {
```

```
          onOff = value!;
        });
      },
    ),
    TextButton(
      onPressed: selectedElementAddress != null
          ? () async {
              final scaffoldMessenger = ScaffoldMessenger.of(context);
              debugPrint('send level $onOff to $selectedElementAddress');
              final provisionerUuid = await
widget.meshManagerApi.meshNetwork!.selectedProvisionerUuid();
              final nodes = await widget.meshManagerApi.meshNetwork!.nodes;
              try {
                final provisionedNode = nodes.firstWhere((element) => element.uuid == provisionerUuid);
                final sequenceNumber = await
widget.meshManagerApi.getSequenceNumber(provisionedNode);
                await widget.meshManagerApi
                    .sendGenericOnOffSet(selectedElementAddress!, onOff, sequenceNumber)
                    .timeout(const Duration(seconds: 40));
                scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
              } on TimeoutException catch (_) {
                scaffoldMessenger.showSnackBar(const SnackBar(content: Text('Board didn\'t respond')));
              } on StateError catch (_) {
                scaffoldMessenger.showSnackBar(
                    SnackBar(content: Text('No provisioner found with this uuid : $provisionerUuid')));
              } on PlatformException catch (e) {
                scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
              } catch (e) {
                scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
              }
            }
          : null,
      child: const Text('Send on off'),
    )
  ],
 );
}
}
```

```
import 'package:flutter/cupertino.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import 'package:nordic_nrf_mesh_example/src/views/control_module/model.dart';

class MeshElement extends StatelessWidget {
  final ElementData element;

  const MeshElement(this.element, {Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        Text('address : ${element.address}'),
        Row(
          children: <Widget>[const Text('Models: '), ...element.models.map((e) => Model(e))],
        )
      ],
    );
  }
}
```

```
import 'package:flutter/material.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

class Model extends StatelessWidget {
  final ModelData model;

  const Model(this.model, {Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Wrap(
      children: <Widget>[Text(' ${model.modelId} '), appKeyBindIcon(), const Text(', ')],
    );
  }

  bool isAppKeyBound() {
    return model.boundAppKey.isNotEmpty;
  }

  Icon appKeyBindIcon() {
    return isAppKeyBound()
        ? const Icon(
            Icons.check,
            size: 15,
            color: Colors.green,
          )
        : const Icon(
            Icons.clear,
            size: 15,
            color: Colors.red,
          );
  }
}
```

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:flutter_reactive_ble/flutter_reactive_ble.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import
'package:nordic_nrf_mesh_example/src/views/control_module/commands/send_config_model_publ
ication_add.dart';

import 'commands/send_deprovisioning.dart';
import 'commands/send_generic_on_off.dart';
import 'commands/send_config_model_subscription_add.dart';
import 'commands/send_generic_level.dart';
import 'node.dart';

class Module extends StatefulWidget {
  final DiscoveredDevice device;
  final MeshManagerApi meshManagerApi;
  final VoidCallback onDisconnect;

  const Module({
    Key? key,
    required this.device,
    required this.meshManagerApi,
    required this.onDisconnect,
  }) : super(key: key);

  @override
  State<Module> createState() => _ModuleState();
}

class _ModuleState extends State<Module> {
  final bleMeshManager = BleMeshManager();

  bool isLoading = true;
  late List<ProvisionedMeshNode> nodes;

  @override
  void initState() {
    super.initState();
    _init();
  }

  @override
  void dispose() {
    _deinit();
    super.dispose();
  }
  @override
  Widget build(BuildContext context) {
   Widget layout = Center(
    child: Column(
     mainAxisSize: MainAxisSize.min,
     children: const [
      CircularProgressIndicator(),
      Text('Connecting ...'),
     ],
    ),
   );
   if (!isLoading) {
    layout = ListView(
     children: <Widget>[
      TextButton(
       onPressed: () {
        _deinit();
        widget.onDisconnect();
       },
       child: const Text('Disconnect from network'),
      ),
      for (var i = 0; i < nodes.length; i++)
       GestureDetector(
        onTap: () {
         Navigator.of(context).push(
          MaterialPageRoute(
           builder: (context) {
            return Node(
             meshManagerApi: widget.meshManagerApi,
             node: nodes[i],
             name: nodes[i].uuid,
            );
           },
          ),
         );
        },
        child: Card(
         child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Container(
           key: ValueKey('node-$i'),
           child: Text(nodes[i].uuid),
          ),
         ),
        ),
       ),
      const Divider(),
      SendGenericLevel(meshManagerApi: widget.meshManagerApi),
        SendGenericOnOff(meshManagerApi: widget.meshManagerApi),
        SendConfigModelSubscriptionAdd(widget.meshManagerApi),
        SendConfigModelPublicationAdd(widget.meshManagerApi),
        SendDeprovisioning(meshManagerApi: widget.meshManagerApi),
      ],
     );
    }
    return layout;
  }

  Future<void> _init() async {
    bleMeshManager.callbacks = DoozProvisionedBleMeshManagerCallbacks(widget.meshManagerApi,
bleMeshManager);
    await bleMeshManager.connect(widget.device);
    // get nodes (ignore first node which is the default provisioner)
    nodes = (await widget.meshManagerApi.meshNetwork!.nodes).skip(1).toList();
    // will bind app keys (needed to be able to configure node)
    for (final node in nodes) {
      final elements = await node.elements;
      for (final element in elements) {
        for (final model in element.models) {
          if (model.boundAppKey.isEmpty) {
            if (element == elements.first && model == element.models.first) {
              continue;
            }
            final unicast = await node.unicastAddress;
            debugPrint('need to bind app key');
            await widget.meshManagerApi.sendConfigModelAppBind(
              unicast,
              element.address,
              model.modelId,
            );
          }
        }
      }
    }

    setState(() {
      isLoading = false;
    });
  }

  void _deinit() async {
    await bleMeshManager.disconnect();
    await bleMeshManager.callbacks!.dispose();
  }
}
```

```
class DoozProvisionedBleMeshManagerCallbacks extends BleMeshManagerCallbacks {
  final MeshManagerApi meshManagerApi;
  final BleMeshManager bleMeshManager;

  late StreamSubscription<ConnectionStateUpdate> onDeviceConnectingSubscription;
  late StreamSubscription<ConnectionStateUpdate> onDeviceConnectedSubscription;
  late StreamSubscription<BleManagerCallbacksDiscoveredServices>
onServicesDiscoveredSubscription;
  late StreamSubscription<DiscoveredDevice> onDeviceReadySubscription;
  late StreamSubscription<BleMeshManagerCallbacksDataReceived> onDataReceivedSubscription;
  late StreamSubscription<BleMeshManagerCallbacksDataSent> onDataSentSubscription;
  late StreamSubscription<ConnectionStateUpdate> onDeviceDisconnectingSubscription;
  late StreamSubscription<ConnectionStateUpdate> onDeviceDisconnectedSubscription;
  late StreamSubscription<List<int>> onMeshPduCreatedSubscription;

  DoozProvisionedBleMeshManagerCallbacks(this.meshManagerApi, this.bleMeshManager) {
    onDeviceConnectingSubscription = onDeviceConnecting.listen((event) {
      debugPrint('onDeviceConnecting $event');
    });
    onDeviceConnectedSubscription = onDeviceConnected.listen((event) {
      debugPrint('onDeviceConnected $event');
    });

    onServicesDiscoveredSubscription = onServicesDiscovered.listen((event) {
      debugPrint('onServicesDiscovered');
    });

    onDeviceReadySubscription = onDeviceReady.listen((event) async {
      debugPrint('onDeviceReady ${event.id}');
    });

    onDataReceivedSubscription = onDataReceived.listen((event) async {
      debugPrint('onDataReceived ${event.device.id} ${event.pdu} ${event.mtu}');
      await meshManagerApi.handleNotifications(event.mtu, event.pdu);
    });
    onDataSentSubscription = onDataSent.listen((event) async {
      debugPrint('onDataSent ${event.device.id} ${event.pdu} ${event.mtu}');
      await meshManagerApi.handleWriteCallbacks(event.mtu, event.pdu);
    });

    onDeviceDisconnectingSubscription = onDeviceDisconnecting.listen((event) {
      debugPrint('onDeviceDisconnecting $event');
    });
    onDeviceDisconnectedSubscription = onDeviceDisconnected.listen((event) {
      debugPrint('onDeviceDisconnected $event');
    });

    onMeshPduCreatedSubscription = meshManagerApi.onMeshPduCreated.listen((event) async {
```

```
    debugPrint('onMeshPduCreated $event');
    await bleMeshManager.sendPdu(event);
   });
  }

  @override
  Future<void> dispose() => Future.wait([
      onDeviceConnectingSubscription.cancel(),
      onDeviceConnectedSubscription.cancel(),
      onServicesDiscoveredSubscription.cancel(),
      onDeviceReadySubscription.cancel(),
      onDataReceivedSubscription.cancel(),
      onDataSentSubscription.cancel(),
      onDeviceDisconnectingSubscription.cancel(),
      onDeviceDisconnectedSubscription.cancel(),
      onMeshPduCreatedSubscription.cancel(),
      super.dispose(),
    ]);

  @override
  Future<void> sendMtuToMeshManagerApi(int mtu) => meshManagerApi.setMtu(mtu);
}
```

```
import 'package:flutter/material.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

import '../control_module/mesh_element.dart';

class Node extends StatefulWidget {
  final String name;
  final ProvisionedMeshNode node;
  final MeshManagerApi meshManagerApi;

  const Node({Key? key, required this.node, required this.meshManagerApi, required this.name}) :
super(key: key);

  @override
  State<Node> createState() => _NodeState();
}

class _NodeState extends State<Node> {
  bool isLoading = true;
  late int nodeAddress;
  late List<ElementData> elements;

  @override
  void initState() {
    super.initState();
    _init();
  }

  void _init() async {
    nodeAddress = await widget.node.unicastAddress;
    elements = await widget.node.elements;
    setState(() {
      isLoading = false;
    });
  }

  @override
  Widget build(BuildContext context) {
    Widget body = Center(
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: const [
          CircularProgressIndicator(),
          Text('Configuring...'),
        ],
      ),
    );
    if (!isLoading) {
      body = ListView(
        children: [
          Text('Node $nodeAddress'),
          Text(widget.node.uuid),
          ...[
            const Text('Elements :'),
            Column(
              children: <Widget>[
                ...elements.map((element) => MeshElement(element)).toList(),
              ],
            ),
          ],
        ],
      );
    }

    return Scaffold(
      appBar: AppBar(
        title: Text(widget.name),
      ),
      body: body,
    );
  }
}
```

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:flutter_reactive_ble/flutter_reactive_ble.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import 'package:nordic_nrf_mesh_example/src/app.dart';
import 'package:nordic_nrf_mesh_example/src/views/control_module/module.dart';
import 'package:nordic_nrf_mesh_example/src/widgets/device.dart';

class ProvisionedDevices extends StatefulWidget {
  final NordicNrfMesh nordicNrfMesh;

  const ProvisionedDevices({Key? key, required this.nordicNrfMesh}) : super(key: key);

  @override
  State<ProvisionedDevices> createState() => _ProvisionedDevicesState();
}

class _ProvisionedDevicesState extends State<ProvisionedDevices> {
  late MeshManagerApi _meshManagerApi;
  final _devices = <DiscoveredDevice>{};
  bool isScanning = false;
  StreamSubscription<DiscoveredDevice>? _scanSubscription;

  DiscoveredDevice? _device;

  @override
  void initState() {
    super.initState();
    _meshManagerApi = widget.nordicNrfMesh.meshManagerApi;
    _scanProvisionned();
  }

  @override
  void dispose() {
    super.dispose();
    _scanSubscription?.cancel();
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        if (isScanning) const LinearProgressIndicator(),
        if (_device == null) ...[
          if (!isScanning && _devices.isEmpty)
            const Expanded(
              child: Center(
          child: Text('No module found'),
        ),
      ),
    if (_devices.isNotEmpty)
      Expanded(
        child: ListView(
          padding: const EdgeInsets.all(8),
          children: [
            for (var i = 0; i < _devices.length; i++)
              Device(
                key: ValueKey('device-$i'),
                device: _devices.elementAt(i),
                onTap: () {
                  setState(() {
                    _device = _devices.elementAt(i);
                  });
                },
              ),
          ],
        ),
      ),
  ] else
    Expanded(
      child: Module(
          device: _device!,
          meshManagerApi: _meshManagerApi,
          onDisconnect: () {
            _device = null;
            _scanProvisionned();
          }),
    ),
  ],
 );
}

Future<void> _scanProvisionned() async {
 setState(() {
   _devices.clear();
 });
 await checkAndAskPermissions();
 _scanSubscription = widget.nordicNrfMesh.scanForProxy().listen((device) async {
   if (_devices.every((d) => d.id != device.id)) {
     setState(() {
       _devices.add(device);
     });
   }
 });
 setState(() {
      isScanning = true;
    });
    return Future.delayed(const Duration(seconds: 10), _stopScan);
  }

  Future<void> _stopScan() async {
    await _scanSubscription?.cancel();
    isScanning = false;
    if (mounted) {
      setState(() {});
    }
  }
}
```

```
import 'dart:async';
import 'dart:io';

import 'package:file_picker/file_picker.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import 'package:nordic_nrf_mesh_example/src/widgets/mesh_network_widget.dart';

class Home extends StatefulWidget {
  final NordicNrfMesh nordicNrfMesh;

  const Home({Key? key, required this.nordicNrfMesh}) : super(key: key);

  @override
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  late IMeshNetwork? _meshNetwork;
  late final MeshManagerApi _meshManagerApi;
  late final StreamSubscription<IMeshNetwork?> onNetworkUpdateSubscription;
  late final StreamSubscription<IMeshNetwork?> onNetworkImportSubscription;
  late final StreamSubscription<IMeshNetwork?> onNetworkLoadingSubscription;

  @override
  void initState() {
    super.initState();
    _meshManagerApi = widget.nordicNrfMesh.meshManagerApi;
    _meshNetwork = _meshManagerApi.meshNetwork;
    onNetworkUpdateSubscription = _meshManagerApi.onNetworkUpdated.listen((event) {
      setState(() {
        _meshNetwork = event;
      });
```

```
   });
   onNetworkImportSubscription = _meshManagerApi.onNetworkImported.listen((event) {
    setState(() {
      _meshNetwork = event;
    });
   });
   onNetworkLoadingSubscription = _meshManagerApi.onNetworkLoaded.listen((event) {
    setState(() {
      _meshNetwork = event;
    });
   });
  }

  @override
  void dispose() {
   onNetworkUpdateSubscription.cancel();
   onNetworkLoadingSubscription.cancel();
   onNetworkImportSubscription.cancel();
   super.dispose();
  }

  @override
  Widget build(BuildContext context) {
   return Scaffold(
    body: ListView(
      children: [
       PlatformVersion(nordicNrfMesh: widget.nordicNrfMesh),
       ExpansionTile(
        title: const Text('Mesh network database'),
        expandedAlignment: Alignment.topLeft,
        children: [MeshNetworkDatabaseWidget(nordicNrfMesh: widget.nordicNrfMesh)],
       ),
       ExpansionTile(
        title: const Text('Mesh network manager'),
        expandedAlignment: Alignment.topLeft,
        children: [MeshNetworkManagerWidget(nordicNrfMesh: widget.nordicNrfMesh)],
       ),
       const Divider(),
       if (_meshNetwork != null)
        MeshNetworkDataWidget(meshNetwork: _meshNetwork!)
       else
        const Text('No meshNetwork loaded'),
      ],
    ),
   );
  }
}
```

```
class PlatformVersion extends StatefulWidget {
  const PlatformVersion({Key? key, required this.nordicNrfMesh}) : super(key: key);

  final NordicNrfMesh nordicNrfMesh;
  @override
  State<PlatformVersion> createState() => _PlatformVersion();
}

class _PlatformVersion extends State<PlatformVersion> {
  String? _platformVersion;

  @override
  Widget build(BuildContext context) {
    if (_platformVersion != null) {
      return Text('Run on $_platformVersion');
    } else {
      return TextButton(
        onPressed: () async {
          var version = await widget.nordicNrfMesh.platformVersion;
          setState(() {
            _platformVersion = version;
          });
        },
        child: const Text('Get Platform Version'),
      );
    }
  }
}

/// A [Widget] to interact with network database.
///
/// User may either :
///   - import a mesh network using the given JSON scheme
///   - export a mesh network to get the associated JSON String
///   - load a mesh network from the local database
///   - reset a mesh network
class MeshNetworkDatabaseWidget extends StatelessWidget {
  final NordicNrfMesh nordicNrfMesh;

  const MeshNetworkDatabaseWidget({Key? key, required this.nordicNrfMesh}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    MeshManagerApi? meshManagerApi = nordicNrfMesh.meshManagerApi;
    IMeshNetwork? meshNetwork = meshManagerApi.meshNetwork;
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        TextButton(
          onPressed: () async {
            final filePath = await FilePicker.platform.pickFiles(type: FileType.any);
            if (filePath == null) return;
            final file = File(filePath.paths.first!);
            debugPrint('loading and importing json file...');
            final json = await file.readAsString();
            await meshManagerApi.importMeshNetworkJson(json);
            debugPrint('done !');
          },
          child: const Text('Import MeshNetwork (JSON)'),
        ),
        TextButton(
          onPressed: meshManagerApi.loadMeshNetwork,
          child: const Text('Load MeshNetwork'),
        ),
        TextButton(
          onPressed: meshNetwork != null
              ? () async {
                  final meshNetworkJson = await meshManagerApi.exportMeshNetwork();
                  debugPrint(meshNetworkJson);
                }
              : null,
          child: const Text('Export MeshNetwork'),
        ),
        TextButton(
          onPressed: meshNetwork != null ? meshManagerApi.resetMeshNetwork : null,
          child: const Text('Reset MeshNetwork'),
        ),
      ],
    );
  }
}

/// A [Widget] to alter network's data without the need of an open connection.
/// _(Manage provisioners, groups, etc.)_
class MeshNetworkManagerWidget extends StatelessWidget {
  final NordicNrfMesh nordicNrfMesh;

  const MeshNetworkManagerWidget({Key? key, required this.nordicNrfMesh}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    MeshManagerApi? meshManagerApi = nordicNrfMesh.meshManagerApi;
    IMeshNetwork? meshNetwork = meshManagerApi.meshNetwork;
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
TextButton(
 onPressed: meshNetwork != null
    ? () async {
       //13 provisioners are maximum with the below given ranges and the default ttl is 5
       //above that, will throw error
       final result = await meshNetwork.addProvisioner(0x0888, 0x02F6, 0x0888, 5);
       debugPrint('provisioner added : $result');
      }
    : null,
 child: const Text('add provisioner'),
),
TextButton(
 onPressed: meshNetwork != null
    ? () async {
       var provs = await meshNetwork.provisioners;
       debugPrint('# of provs : ${provs.length}');
       for (var value in provs) {
         debugPrint('$value');
       }
      }
    : null,
 child: const Text('get provisioner list'),
),
TextButton(
 onPressed: meshNetwork != null
    ? () async {
       final scaffoldMessenger = ScaffoldMessenger.of(context);
       final groupName = await showDialog<String>(
          context: context,
          builder: (c) {
           String? groupName;
           return Dialog(
             shape: const RoundedRectangleBorder(
              borderRadius: BorderRadius.all(Radius.circular(5.0)),
             ),
             insetPadding: const EdgeInsets.symmetric(horizontal: 40),
             elevation: 0.0,
             child: Padding(
              padding: const EdgeInsets.fromLTRB(24.0, 20.0, 24.0, 24.0),
              child: Container(
               decoration: const BoxDecoration(
                shape: BoxShape.rectangle,
               ),
               child: Column(
                mainAxisSize: MainAxisSize.min,
                crossAxisAlignment: CrossAxisAlignment.stretch,
                children: <Widget>[
                 TextField(
                        decoration: const InputDecoration(labelText: 'Group name'),
                        onChanged: (text) => groupName = text,
                      ),
                      TextButton(
                        onPressed: () => Navigator.pop(c, groupName),
                        child: const Text('OK'),
                      )
                    ],
                  ),
                ),
              ),
            );
          });
        if (groupName != null && groupName.isNotEmpty) {
          try {
            await meshManagerApi.meshNetwork!.addGroupWithName(groupName);
            scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
          } on PlatformException catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
          } catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
          }
        } else {
          scaffoldMessenger.showSnackBar(const SnackBar(content: Text('No name given,
aborting')));
        }
      }
    : null,
  child: const Text('Create group with name'),
),
TextButton(
  onPressed: meshNetwork != null
    ? () async {
        final scaffoldMessenger = ScaffoldMessenger.of(context);
        final groupAdr = await showDialog<String>(
          context: context,
          builder: (c) {
            String? groupAdr;
            return Dialog(
              shape: const RoundedRectangleBorder(
                borderRadius: BorderRadius.all(Radius.circular(5.0)),
              ),
              insetPadding: const EdgeInsets.symmetric(horizontal: 40),
              elevation: 0.0,
              child: Padding(
                padding: const EdgeInsets.fromLTRB(24.0, 20.0, 24.0, 24.0),
                child: Container(
                  decoration: const BoxDecoration(
                    shape: BoxShape.rectangle,
                  ),
                  child: Column(
                    mainAxisSize: MainAxisSize.min,
                    crossAxisAlignment: CrossAxisAlignment.stretch,
                    children: <Widget>[
                      TextField(
                        decoration: const InputDecoration(labelText: 'Group address'),
                        keyboardType: TextInputType.number,
                        onChanged: (text) => groupAdr = text,
                      ),
                      TextButton(
                        onPressed: () => Navigator.pop(c, groupAdr),
                        child: const Text('OK'),
                      )
                    ],
                  ),
                ),
              ),
            );
          });
        if (groupAdr != null && groupAdr.isNotEmpty) {
          try {
            await meshManagerApi.meshNetwork!.removeGroup(int.parse(groupAdr));
            scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
          } on PlatformException catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
          } catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
          }
        } else {
          scaffoldMessenger.showSnackBar(const SnackBar(content: Text('No address given,
aborting')));
        }
      }
    : null,
  child: const Text('Delete group'),
),
TextButton(
  onPressed: meshNetwork != null
    ? () async {
        final scaffoldMessenger = ScaffoldMessenger.of(context);
        final groupAdr = await showDialog<String>(
            context: context,
            builder: (c) {
            String? groupAdr;
            return Dialog(
              shape: const RoundedRectangleBorder(
                borderRadius: BorderRadius.all(Radius.circular(5.0)),
              ),
              insetPadding: const EdgeInsets.symmetric(horizontal: 40),
              elevation: 0.0,
              child: Padding(
                padding: const EdgeInsets.fromLTRB(24.0, 20.0, 24.0, 24.0),
                child: Container(
                  decoration: const BoxDecoration(
                    shape: BoxShape.rectangle,
                  ),
                  child: Column(
                    mainAxisSize: MainAxisSize.min,
                    crossAxisAlignment: CrossAxisAlignment.stretch,
                    children: <Widget>[
                      TextField(
                        decoration: const InputDecoration(labelText: 'Group address'),
                        keyboardType: TextInputType.number,
                        onChanged: (text) => groupAdr = text,
                      ),
                      TextButton(
                        onPressed: () => Navigator.pop(c, groupAdr),
                        child: const Text('OK'),
                      )
                    ],
                  ),
                ),
              ),
            );
            });
        if (groupAdr != null && groupAdr.isNotEmpty) {
          try {
            final subs = await
meshManagerApi.meshNetwork!.elementsForGroup(int.parse(groupAdr));
            debugPrint('$subs');
            scaffoldMessenger.showSnackBar(const SnackBar(content: Text('OK')));
          } on PlatformException catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text('${e.message}')));
          } catch (e) {
            scaffoldMessenger.showSnackBar(SnackBar(content: Text(e.toString())));
          }
        } else {
          scaffoldMessenger.showSnackBar(const SnackBar(content: Text('No address given,
aborting')));
        }
      }
    : null,
  child: const Text('Get group elements'),
),
    ],
   );
  }
}
```

```
import 'dart:async';
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:flutter_reactive_ble/flutter_reactive_ble.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import 'package:nordic_nrf_mesh_example/src/app.dart';
import 'package:nordic_nrf_mesh_example/src/widgets/device.dart';

class ScanningAndProvisioning extends StatefulWidget {
  final NordicNrfMesh nordicNrfMesh;
  final VoidCallback onGoToControl;

  const ScanningAndProvisioning({
    Key? key,
    required this.nordicNrfMesh,
    required this.onGoToControl,
  }) : super(key: key);

  @override
  State<ScanningAndProvisioning> createState() => _ScanningAndProvisioningState();
}

class _ScanningAndProvisioningState extends State<ScanningAndProvisioning> {
  late MeshManagerApi _meshManagerApi;
  bool isScanning = true;
  StreamSubscription? _scanSubscription;
  bool isProvisioning = false;

  final _serviceData = <String, Uuid>{};
  final _devices = <DiscoveredDevice>{};

  @override
  void initState() {
    super.initState();
    _meshManagerApi = widget.nordicNrfMesh.meshManagerApi;
    _scanUnprovisionned();
  }

  @override
  void dispose() {
    super.dispose();
    _scanSubscription?.cancel();
  }

  Future<void> _scanUnprovisionned() async {
    _serviceData.clear();
    setState(() {
      _devices.clear();
    });
    await checkAndAskPermissions();
    _scanSubscription = widget.nordicNrfMesh.scanForUnprovisionedNodes().listen((device) async {
      if (_devices.every((d) => d.id != device.id)) {
        final deviceUuid =

Uuid.parse(_meshManagerApi.getDeviceUuid(device.serviceData[meshProvisioningUuid]!.toList()));
        debugPrint('deviceUuid: $deviceUuid');
        _serviceData[device.id] = deviceUuid;
        _devices.add(device);
        setState(() {});
      }
    });
    setState(() {
      isScanning = true;
    });
    return Future.delayed(const Duration(seconds: 10), _stopScan);
  }

  Future<void> _stopScan() async {
    await _scanSubscription?.cancel();
    isScanning = false;
    if (mounted) {
      setState(() {});
    }
  }

  Future<void> provisionDevice(DiscoveredDevice device) async {
    final scaffoldMessenger = ScaffoldMessenger.of(context);
    if (isScanning) {
      await _stopScan();
    }
    if (isProvisioning) {
      return;
    }
    isProvisioning = true;

    try {
      // Android is sending the mac Adress of the device, but Apple generates
      // an UUID specific by smartphone.

      String deviceUUID;
      if (Platform.isAndroid) {
        deviceUUID = _serviceData[device.id].toString();
      } else if (Platform.isIOS) {
        deviceUUID = device.id.toString();
      } else {
        throw UnimplementedError('device uuid on platform : ${Platform.operatingSystem}');
      }
      final provisioningEvent = ProvisioningEvent();
      final provisionedMeshNodeF = widget.nordicNrfMesh
          .provisioning(
            _meshManagerApi,
            BleMeshManager(),
            device,
            deviceUUID,
            events: provisioningEvent,
          )
          .timeout(const Duration(minutes: 1));

      unawaited(provisionedMeshNodeF.then((node) {
        Navigator.of(context).pop();
        scaffoldMessenger
            .showSnackBar(const SnackBar(content: Text('Provisionning succeed, redirecting to control
tab...')));
        Future.delayed(const Duration(milliseconds: 500), widget.onGoToControl);
      }).catchError((_) {
        Navigator.of(context).pop();
        scaffoldMessenger.showSnackBar(const SnackBar(content: Text('Provisionning failed')));
        _scanUnprovisionned();
      }));
      await showDialog(
        context: context,
        barrierDismissible: false,
        builder: (_) => ProvisioningDialog(provisioningEvent: provisioningEvent),
      );
    } catch (e) {
      debugPrint('$e');
      scaffoldMessenger.showSnackBar(SnackBar(content: Text('Caught error: $e')));
    } finally {
      isProvisioning = false;
    }
  }

  @override
  Widget build(BuildContext context) {
    return RefreshIndicator(
      onRefresh: () {
        if (isScanning) {
          return Future.value();
        }
        return _scanUnprovisionned();
      },
      child: Column(
        children: [
          if (isScanning) const LinearProgressIndicator(),
          if (!isScanning && _devices.isEmpty)
            const Expanded(
              child: Center(
                child: Text('No module found'),
              ),
            ),
          if (_devices.isNotEmpty)
            Expanded(
              child: ListView(
                padding: const EdgeInsets.all(8),
                children: [
                  for (var i = 0; i < _devices.length; i++)
                    Device(
                      key: ValueKey('device-$i'),
                      device: _devices.elementAt(i),
                      onTap: () => provisionDevice(_devices.elementAt(i)),
                    ),
                ],
              ),
            ),
        ],
      ),
    );
  }
}

class ProvisioningDialog extends StatelessWidget {
  final ProvisioningEvent provisioningEvent;

  const ProvisioningDialog({Key? key, required this.provisioningEvent}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Card(
        margin: const EdgeInsets.all(8),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          mainAxisSize: MainAxisSize.min,
          children: [
            const LinearProgressIndicator(),
            Padding(
              padding: const EdgeInsets.all(8),
              child: Column(
                children: [
                  const Text('Steps :'),
                  Column(
                    children: [
                      ProvisioningState(
                        text: 'onProvisioningCapabilities',
                        stream: provisioningEvent.onProvisioningCapabilities.map((event) => true),
                      ),
                      ProvisioningState(
                        text: 'onProvisioning',
                        stream: provisioningEvent.onProvisioning.map((event) => true),
                      ),
                      ProvisioningState(
                        text: 'onProvisioningReconnect',
                        stream: provisioningEvent.onProvisioningReconnect.map((event) => true),
                      ),
                      ProvisioningState(
                        text: 'onConfigCompositionDataStatus',
                        stream: provisioningEvent.onConfigCompositionDataStatus.map((event) => true),
                      ),
                      ProvisioningState(
                        text: 'onConfigAppKeyStatus',
                        stream: provisioningEvent.onConfigAppKeyStatus.map((event) => true),
                      ),
                    ],
                  )
                ],
              ),
            ),
          ],
        ),
      ),
    );
  }
}

class ProvisioningState extends StatelessWidget {
  final Stream<bool> stream;
  final String text;

  const ProvisioningState({Key? key, required this.stream, required this.text}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<bool>(
      initialData: false,
      stream: stream,
      builder: (context, snapshot) {
        return Row(
          children: [
            Text(text),
            const Spacer(),
            Checkbox(
              value: snapshot.data,
              onChanged: null,
            ),
          ],
        );
      },
    );
  }
}
```

```
import 'package:flutter/material.dart';
import 'package:flutter_reactive_ble/flutter_reactive_ble.dart';

class Device extends StatelessWidget {
  final DiscoveredDevice device;
  final VoidCallback? onTap;

  const Device({Key? key, required this.device, this.onTap}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      child: InkWell(
        onTap: onTap,
        child: Padding(
          padding: const EdgeInsets.all(8),
          child: Text('${device.name} : ${device.id}'),
        ),
      ),
    );
  }
}
```

```
import 'package:flutter/material.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import 'package:nordic_nrf_mesh_example/src/views/control_module/mesh_element.dart';

class Group extends StatefulWidget {
  final GroupData groupData;
  final IMeshNetwork meshNetwork;
  const Group(this.groupData, this.meshNetwork, {Key? key}) : super(key: key);

  @override
  State<Group> createState() => _GroupState();
}

class _GroupState extends State<Group> {
  List<ElementData> elements = [];

  @override
  Widget build(BuildContext context) {
    return ExpansionTile(
      title: Text(widget.groupData.name),
      subtitle: Text(widget.groupData.address.toString()),
      onExpansionChanged: (isOpen) {
        if (isOpen) {
          debugPrint('load elements');
          widget.meshNetwork
              .elementsForGroup(widget.groupData.address)
              .then((value) => setState(() => elements = value));
        }
      },
      children: <Widget>[
        const Text('Elements :'),
        Column(
          children: <Widget>[
            ...elements.map((e) => MeshElement(e)).toList(),
          ],
        ),
      ],
    );
  }
}
```

```
import 'package:flutter/material.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';

import 'group.dart';
import 'node.dart';

class MeshNetworkDataWidget extends StatefulWidget {
  final IMeshNetwork meshNetwork;

  const MeshNetworkDataWidget({Key? key, required this.meshNetwork}) : super(key: key);

  @override
  State<MeshNetworkDataWidget> createState() => _MeshNetworkDataWidgetState();
}
```

```
class _MeshNetworkDataWidgetState extends State<MeshNetworkDataWidget> {
  List<ProvisionedMeshNode> _nodes = [];
  List<GroupData> _groups = [];

  @override
  void initState() {
    super.initState();
    widget.meshNetwork.nodes.then((value) => setState(() => _nodes = value));
    widget.meshNetwork.groups.then((value) => setState(() => _groups = value));
  }

  @override
  void didUpdateWidget(covariant MeshNetworkDataWidget oldWidget) {
    super.didUpdateWidget(oldWidget);
    // nodes, provisioners and groups may have changed
    widget.meshNetwork.nodes.then((value) => setState(() => _nodes = value));
    widget.meshNetwork.groups.then((value) => setState(() => _groups = value));
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        Text('MeshNetwork ID: ${widget.meshNetwork.id}'),
        if (_nodes.isNotEmpty) ...[
          Text('Nodes (${_nodes.length}): '),
          ..._nodes.map((e) => Node(e, widget.meshNetwork, 'node-${_nodes.indexOf(e)}')),
        ],
        if (_groups.isNotEmpty) ...[
          Text('Groups (${_groups.length}): '),
          ..._groups.map((e) => Group(e, widget.meshNetwork)),
        ]
      ],
    );
  }
}
```

```
import 'package:flutter/material.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import 'package:nordic_nrf_mesh_example/src/views/control_module/mesh_element.dart';

class Node extends StatefulWidget {
  final ProvisionedMeshNode node;
  final IMeshNetwork meshNetwork;
  final String testKey; // For flutter driver tests
  const Node(this.node, this.meshNetwork, this.testKey, {Key? key}) : super(key: key);
```

```
 @override
 State<Node> createState() => _NodeState();
}

class _NodeState extends State<Node> {
 String _nodeUuid = 'reaching...';
 String _nodeAdress = 'reaching...';
 List<ElementData> _elements = [];

 @override
 void initState() {
  super.initState();
  _nodeUuid = widget.node.uuid;
  widget.node.unicastAddress.then((value) => setState(() => _nodeAdress = value.toString()));
 }

 @override
 Widget build(BuildContext context) {
  return ExpansionTile(
   key: Key(widget.testKey),
   title: Text(_nodeUuid),
   subtitle: Text(_nodeAdress),
   onExpansionChanged: (isOpen) {
    if (isOpen) {
     debugPrint('load elements');
     widget.node.elements.then((value) => setState(() => _elements = value));
    }
   },
   children: <Widget>[
    const Text('Elements :'),
    Column(
     children: <Widget>[
      ..._elements.map((e) => MeshElement(e)).toList(),
     ],
    ),
   ],
  );
 }
}
```

```
import 'package:device_info_plus/device_info_plus.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:nordic_nrf_mesh/nordic_nrf_mesh.dart';
import 'package:nordic_nrf_mesh_example/src/views/control_module/provisioned_devices.dart';
import 'package:nordic_nrf_mesh_example/src/views/home/home.dart';
import
'package:nordic_nrf_mesh_example/src/views/scan_and_provisionning/scan_and_provisioning.dart';
```

```
import 'package:permission_handler/permission_handler.dart';

const int homeTab = 0;
const int provisioningTab = 1;
const int controlTab = 2;
// used for app's theme
const Map<int, Color> primarySwatch = {
  50: Color.fromRGBO(0, 164, 153, .1),
  100: Color.fromRGBO(0, 164, 153, .2),
  200: Color.fromRGBO(0, 164, 153, .3),
  300: Color.fromRGBO(0, 164, 153, .4),
  400: Color.fromRGBO(0, 164, 153, .5),
  500: Color.fromRGBO(0, 164, 153, .6),
  600: Color.fromRGBO(0, 164, 153, .7),
  700: Color.fromRGBO(0, 164, 153, .8),
  800: Color.fromRGBO(0, 164, 153, .9),
  900: Color.fromRGBO(0, 164, 153, 1),
};

class NordicNrfMeshExampleApp extends StatefulWidget {
  const NordicNrfMeshExampleApp({Key? key}) : super(key: key);

  @override
  State<NordicNrfMeshExampleApp> createState() =>
      _NordicNrfMeshExampleAppState();
}

class _NordicNrfMeshExampleAppState extends State<NordicNrfMeshExampleApp> {
  late final GlobalKey<ScaffoldMessengerState> _scaffoldKey =
      GlobalKey<ScaffoldMessengerState>(debugLabel: 'main_scaffold');
  late final NordicNrfMesh nordicNrfMesh = NordicNrfMesh();

  int _bottomNavigationBarIndex = homeTab;

  @override
  Widget build(BuildContext context) {
    Widget body = const SizedBox.shrink();

    if (_bottomNavigationBarIndex == homeTab) {
      //  home
      body = Home(nordicNrfMesh: nordicNrfMesh);
    } else if (_bottomNavigationBarIndex == provisioningTab) {
      //  scanning & provisionning
      body = ScanningAndProvisioning(
          nordicNrfMesh: nordicNrfMesh,
          onGoToControl: () {
            setState(() {
              _bottomNavigationBarIndex = controlTab;
      });
     });
   } else if (_bottomNavigationBarIndex == controlTab) {
    //  List provisioned devices and then can control/setup them
    body = ProvisionedDevices(nordicNrfMesh: nordicNrfMesh);
   }

   return MaterialApp(
    theme: ThemeData(
     brightness: Brightness.light,
     primarySwatch: const MaterialColor(0xFF00A499, primarySwatch),
     primaryColor: const Color.fromRGBO(0, 164, 153, 1),
     primaryColorDark: const Color.fromRGBO(0, 114, 105, 1),
     scaffoldBackgroundColor: Colors.white,
    ),
    home: ScaffoldMessenger(
     key: _scaffoldKey,
     child: Scaffold(
      appBar: AppBar(),
      body: body,
      bottomNavigationBar: BottomNavigationBar(
       currentIndex: _bottomNavigationBarIndex,
       onTap: (newBottomNavigationBarIndex) {
        if (nordicNrfMesh.meshManagerApi.meshNetwork != null) {
         setState(() {
          _bottomNavigationBarIndex = newBottomNavigationBarIndex;
         });
        } else {
         _scaffoldKey.currentState!.clearSnackBars();
         _scaffoldKey.currentState!.showSnackBar(const SnackBar(
           content: Text('Please load a mesh network')));
        }
       },
       items: const [
        BottomNavigationBarItem(
         icon: Icon(Icons.home),
         label: 'Home',
        ),
        BottomNavigationBarItem(
         icon: Icon(Icons.bluetooth_searching),
         label: 'Provisioning',
        ),
        BottomNavigationBarItem(
         icon: Icon(Icons.videogame_asset),
         label: 'Control',
        ),
       ],
      ),
      ),
    ),
  );
 }
}

void log(Object? msg) => debugPrint(
  '[$NordicNrfMeshExampleApp - ${DateTime.now().toIso8601String()}] $msg');

Future<void> checkAndAskPermissions() async {
 if (defaultTargetPlatform == TargetPlatform.android) {
  final androidInfo = await DeviceInfoPlugin().androidInfo;
  if (androidInfo.version.sdkInt! < 31) {
   // location
   await Permission.locationWhenInUse.request();
   await Permission.locationAlways.request();
   // bluetooth
   await Permission.bluetooth.request();
  } else {
   // bluetooth for Android 12 and up
   await Permission.bluetoothScan.request();
   await Permission.bluetoothConnect.request();
  }
 } else {
  // bluetooth for iOS 13 and up
  await Permission.bluetooth.request();
 }
}
```

16. References

[1] M. M. R. A. S. T. Tamaryn Menneer, "Modelling mould growth in domestic environments using relative humidity and temperature," *Building and Environment,* vol. 208, no. 108583, 2022.

[2] Mach1global, "WHY TEMPERATURE CONTROL IS IMPORTANT IN PHARMACEUTICAL LOGISTICS," Mach1, 4 October 2017. [Online]. Available: https://www.mach1global.com/importance-of-temperature-control-pharmaceutical-logistics/. [Accessed 4 September 2022].

[3] Zhang, Qinghua & Wang, Yi & Cheng, Guoquan & Zhuan, Wang & Shi, Dongmei. (2014). Research on warehouse environment monitoring systems based on wireless sensor networks. Proceedings of the 2014 9th IEEE Conference on Industrial Electronics and Applications, ICIEA 2014. 1639-1644. 10.1109/ICIEA.2014.6931430.

[4] Roomi, M. (2020, March 23). *6 advantages and disadvantages of WIFI: Drawbacks and benefits of wireless networks*. HitechWhizz. Retrieved September 13, 2022, from https://www.hitechwhizz.com/2020/03/6-advantages-and-disadvantages-drawbacks-benefits-of-wifi.html

[5] Prasanna. (2022, January 25). *Zigbee technology advantages and disadvantages: Zigbee Technology Architecture and its applications*. A Plus Topper. Retrieved September 13, 2022, from https://www.aplustopper.com/zigbee-technology-advantages-and-disadvantages/

[6] https://www.konsyse.com/articles/advantages-and-disadvantages-of-z-wave/

[7] https://www.nicerf.com/articles/detail/advantages-and-disadvantages-of-lora-module.html

[8] Editorial, E. (2021, February 10). *System on a chip (SOC) - advantages and disadvantages explained*. RoboticsBiz. Retrieved August 31, 2022, from https://roboticsbiz.com/system-on-a-chip-soc-advantages-and-disadvantages-explained/#:~:text=SoC%20basically%20has%20smaller%20footprint,size%20means%20it%20is%20lightweight.&text=Application%2Dspecific%20SoCs%20can%20be%20cost%2Defficient.

[9] Author, L. M. (2019, December 7). *System-on-A-chip vs. Single Board computers: Linear Microsystems*. Linear Microsystems (LMI). Retrieved August 31, 2022, from

[10] Geeks for Geeks. (2022, July 7). *Advantages and disadvantages of Microcontroller*. GeeksforGeeks. Retrieved August 31, 2022, from https://www.geeksforgeeks.org/advantages-and-disadvantages-of-microcontroller/#:~:text=Advantages%20of%20the%20microcontroller%20%3A&text=At%20an%20equivalent%20time%2C%20many,%2C%20and%20I%2FO%20port.

[11] Espressif Systems, ESP32-WROOM-32, Datasheet Version 2.5, Pages10-11.

[12] Raspberry Pi Ltd. (2022, June 6). Raspberry Pi Pico W Datasheet. Raspberry Pi Ltd. Page

[13] Arduino®. (2022, September 1). Arduino® Nano 33 BLE Datasheet/Product Reference Manual. Page 2

[14] Townsend, K. (n.d.). *Bluefruit NRF52 Feather Learning Guide*. Adafruit Learning System. Retrieved September 1, 2022, from https://learn.adafruit.com/bluefruit-nrf52-feather-learning-guide

[15] WROOM ESP32 WIFI based Microcontroller Development Board. (n.d.). Retrieved September 1, 2022, from https://hallroad.org/esp32.html

[16] CART. (n.d.). Retrieved September 1, 2022, from https://www.daraz.pk/products/esp32-wifi-esp-wroom-32-ch9102x-i222155602-s1437614446.html?spm=a2a0e.searchlist.list.1.29493e67LlFLhr&search=1

[17] *Raspberry pi pico W WIFI in Pakistan*. Hallroad Lahore. (n.d.). Retrieved September 1, 2022, from https://hallroadlahore.pk/raspberry-pi-pico-w-wifi.html

[18] *Arduino: Nano 33 BLE sense NRF52480 development board - ABX00031*. InStock.PK | Pakistan's First Online Electronics Parts Store. (n.d.). Retrieved September 1, 2022, from https://instock.pk/arduino-nano-33-ble-sense-abx00031.html

[19] *Adafruit Feather NRF52 Bluefruit Le price: 8035.00 rs*. Price: 8035.00 Rs. (n.d.). Retrieved September 1, 2022, from https://www.shoppingbag.pk/amazon-adafruit-feather-nrf52-bluefruit-le/O06KKFILYP.html

[20] Duncan, Santos, R., M.T.Mendis, Cullins, J., CobRce, Ton, Ludwig, S., Stefan, Buenaflor, A. R., Santos, S., Saravanan, JDavis, Mutahi, K., Jupp, M., Pableitor, Peter, Gibson, M., Alan, Amos, D., … Behrouz. (2022, April 19). *Installing ESP32 in Arduino IDE (windows, mac OS X, linux)*. Random Nerd Tutorials. Retrieved September 2, 2022, from https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/#:~:text=There's%20an%20add%2Don%20for,Mac%20OS%20X%20or%20 Linux.

[21] Max, Alimiracle, Lynn, H., Upton, L., Ben, Watkiss, S., Geerling, J., Szaja, Calvo, F. G., Felix, Johanson, P., Allan, A., Hellmark, Kamat, P., Pelowitz, D., & Zimmermann, U. (2022, June 30). *Raspberry pi pico W: Your $6 IOT platform*. Raspberry Pi. Retrieved September 2, 2022, from https://www.raspberrypi.com/news/raspberry-pi-pico-w-your-6-iot-platform/

[22] Team, T. A. (n.d.). *Nano 33 ble: Arduino Documentation*. Arduino Documentation | Arduino Documentation. Retrieved September 2, 2022, from https://www.arduino.cc/en/Guide/NANO33BLE

[23] Townsend, K. (n.d.). *Bluefruit NRF52 Feather Learning Guide*. Adafruit Learning System. Retrieved September 2, 2022, from https://learn.adafruit.com/bluefruit-nrf52-feather-learning-guide/software-resources

[24] Blog Post Comparing The Different Wireless Technologies For Condition Monitoring Applications In IIoT There are several wireless technologies to choose from, Video Nordic Demonstrates Auracast™ Broadcast Audio Watch Nordic Semiconductor's Vince Hagen describe the demonstration of Auracast™ broadcast audio Nordic shared… , Case Study Farm to Store with Israel's Largest Retailer, & Bluetooth News Auracast is coming to Bluetooth. (n.d.). *Bluetooth technology overview*. Bluetooth® Technology Website. Retrieved October 2, 2022, from https://www.bluetooth.com/learn-about-bluetooth/tech-overview/