Raspberry Pi Mesh Network for IoT Devices

by

Bryan Kyritz, Juan Jimenez

bkyritz@stevens.edu, jjimene6@stevens.edu

September 8, 2023

bkyritz@stevens.edu, jjimene6@stevens.edu

Table of Contents

List of Figures

Chapter 1

Introduction

A mesh network is a type of network topology in which each node in the network can communicate with multiple other nodes in the network, creating a highly redundant and resilient network architecture. In a mesh network, each node acts as both a sender and a receiver, relaying data through the network until it reaches its intended destination. This allows for the creation of a self-configuring and self-healing network that can be quickly deployed and is resilient to failures. Figure 1.1 shows a typical mesh network configuration.

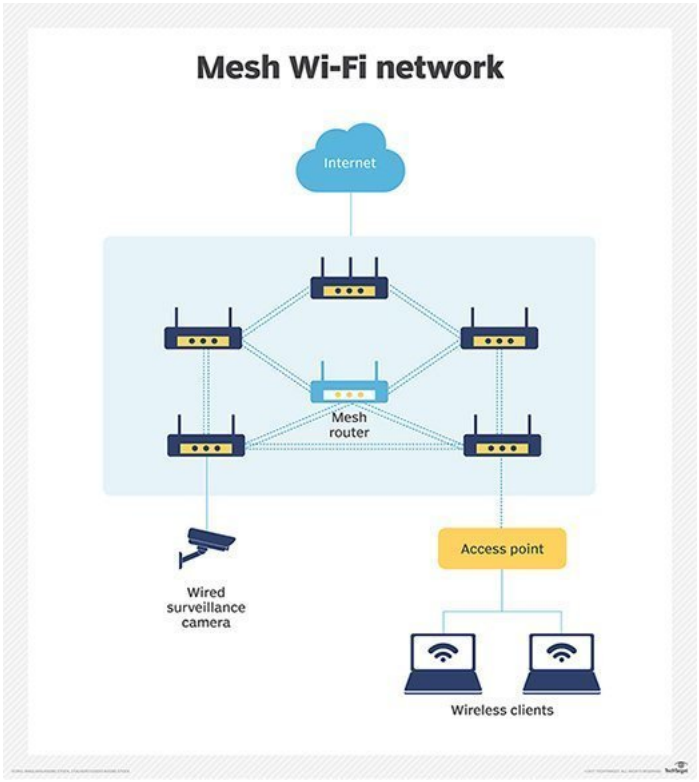

Figure 1.1: Sample Mesh Network Configuration

Mesh networks have numerous applications, and one of the most promising is for Internet

of Things (IoT) devices. IoT devices typically rely on wireless communication, but traditional wireless networks may not be sufficient for large-scale deployments. An ad-hoc mesh network can provide the necessary scalability and reliability for IoT devices, allowing them to communicate with each other and with central servers without relying on a single point of failure. This makes them ideal for scenarios where traditional networks may not be available or feasible, such as in disaster response situations or in remote locations.

Raspberry Pi's are small, low-cost, single-board computers which provide incredible performance despite their small size and cost. Raspberry Pi's are roughly the size of a credit card and come with various input and output ports, including HDMI, USB, Ethernet, and GPIO (General Purpose Input/Output) pins. They can run a variety of operating systems, including Linux-based distributions such as Raspbian and Ubuntu, and can be programmed in various programming languages, including Python, C, and Java. Because of their small size, low power consumption, and versatility, Raspberry Pi's are an excellent platform for the creation of low-cost and flexible ad-hoc mesh networks. This project will utilize a series of Raspberry Pi's along with some USB Wi-Fi dongles to implement an ad-hoc network with IoT capabilities. The project will achieve this by utilizing several Linux software libraries together to create a framework for a Raspberry-Pi centered mesh network with IoT capabilities.

1.1 Specifications

The following are all the necessary elements which are required to setup the Raspberry Pi Mesh network:

- At minimum, two Raspberry Pis (Model 3 or 4) are required, although additional Pis can also be used to extend the mesh network. In our test network, we implemented three Raspberry Pis - one Raspberry Pi 4 and two Raspberry Pi 3s - to demonstrate this capability. Each Pi requires an SD card to hold the operating system and the filesystem, as well as a power supply. While a headerless setup can be configured to access the Pis through the local network, a keyboard and mouse are necessary for troubleshooting and for creating a non-headerless setup.
- A laptop or desktop computer with a modern OS (MacOS, Linux, or Windows)
- The ability to flash the SD card for the Raspberry Pi (USB flashing adapter)
- Ethernet cable and connection to connect Laptop to Ethernet (if this isn't available then WiFi is also possible)
- Internet connectivity is required
- Two Wi-Fi USB dongles. For this project two Edimax EW-7811UN adapters were used, although other USB dongles should also work under the correct setup.
- A breadboard, LED diode, and some cables to implement an IoT device.

Chapter 2

Creating the Mesh Network

As described in the introduction an ad-hoc mesh network is a form of decentralized networking where nodes commmunicated with each other directly, without the need for a centralized access point or router. In an ad-hoc mesh network, each node acts as both a sender and a receiver, forwarding data packets to other nodes in the network until they reach their destination. However, to provide access to the internet or to other networks, the mesh requires the use of two specialized nodes: a gateway node and a bridge node:

- A **Gateway Node** is a special type of node in a mesh network that serves as a bridge between the mesh network and other networks, such as the Internet or other local area networks (LANs). The gateway node acts as a point of contact between the mesh network and external networks, allowing data to flow between them. In a mesh network, each node can communicate with other nodes in the network directly or through intermediate nodes, creating a highly resilient and decentralized network. However, when a node needs to communicate with a device outside of the mesh network, such as a device on the Internet, it needs to go through a gateway node. The gateway node is typically connected to the external network through a wired connection, such as Ethernet or fiber optic cable, for the purposes of this project it will be connected using an Wi-Fi connection. It uses this connection to forward data packets between the mesh network and the external network. In addition to serving as a bridge between the mesh network and external networks, a gateway node may also provide other services, such as security features like firewalls, or network management tools like network monitoring and configuration.

- A **Bridge Node** is a type of node that serves as a connection point between two or more subnets within the mesh network. A subnet is a smaller network within a larger network, and it may contain a group of nodes that have a specific purpose or belong to a particular location. Bridge nodes allow for communication between nodes on different subnets, helping to expand the coverage and reach of the mesh network. For example, imagine a mesh network deployed in a large building with multiple floors. Nodes on each floor may form their own subnet, but they need to communicate with nodes on other floors. Bridge nodes would be used to connect the subnets, allowing nodes on different floors to communicate with each other. Bridge nodes can also be used to connect different types of networks, such as wired and wireless networks, or networks that use different protocols or standards. This

enables devices on different networks to communicate with each other, even if they are not directly connected. In our project the bridge node will be used as an access point to the mesh network that other devices or subnets can use to connect to the mesh network.

To summarize, our mesh network will consist of one Raspberry Pi serving as a gateway node, another Pi serving as a bridge node, and a third Pi serving as an intermediate node in the network running an IoT application. The minimum implementation of our mesh only requires two nodes: a gateway and a bridge node, or two gateway nodes, or two bridge nodes. These are the minimum configurations which can be set up to have a fully functioning mesh network. Figure 2.1 summarizes the setup.

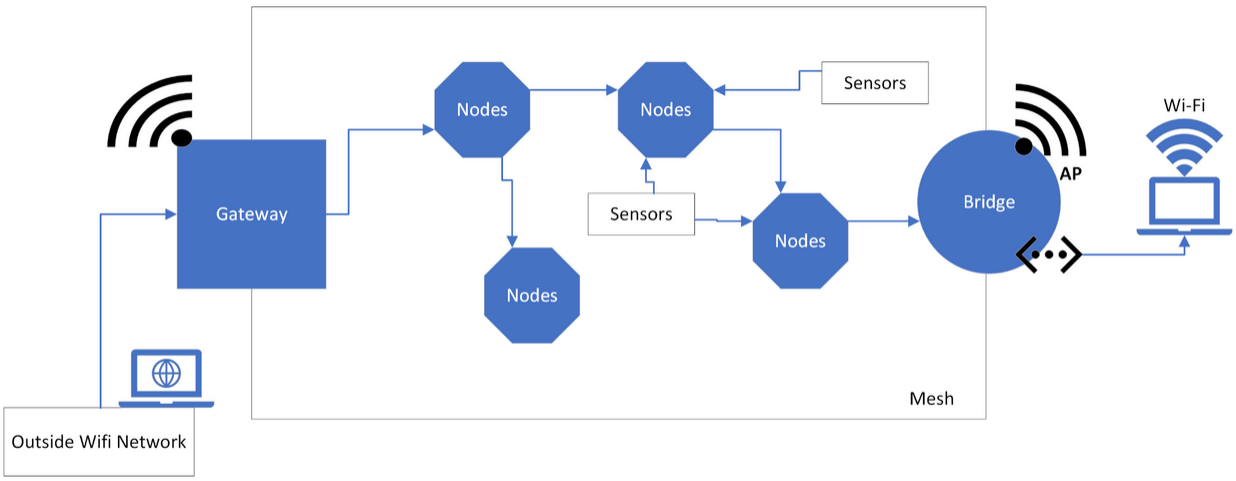

Figure 2.1: Mesh Network Implementation

2.1 Implementation

To create the mesh network it is necessary to convert the Raspberry Pi's into gateway and bridge nodes. We achieve this by combining several software libraries designed for networking. Namely we utilize: Batman ADV, Hostapd, and Iptables:

- **Batman-adv** is a routing protocol that is used in wireless mesh networks to enable efficient communication between nodes. It works by creating a virtual mesh network, where each node acts as a router and is able to forward data packets to other nodes in the network. One of the key features of batman-adv is its ability to handle multiple paths and choose the most efficient route for data transmission, which helps to improve network performance and reliability. It also supports automatic mesh network configuration and dynamic topology changes, making it suitable for dynamic and changing environments. In addition, batman-adv provides support for Quality of Service (QoS) features, allowing for the prioritization of

certain types of traffic. It also includes security features, such as encryption and authentication, to help ensure the privacy and integrity of network data. We use Batman-adv to setup the mesh protocol and have all the nodes communicate with each other.

- **Hostapd** Hostapd is a software package that is used to turn a computer or device with a wireless network interface into a wireless access point (AP). It allows the device to create a Wi-Fi network that other devices can connect to, similar to the network provided by a typical wireless router. The main function of hostapd is to manage the wireless network interface and handle authentication and encryption of network traffic. It supports a variety of authentication and encryption methods, including WPA and WPA2, which help to ensure the security and privacy of network communications. Hostapd is used to create a Wi-Fi and ethernet access point on the bridge node which enables external computers and networks to access and connect to the mesh network.

- **Iptables** Iptables is a software tool used in Linux-based operating systems to manage and control network traffic. It works by filtering and manipulating packets of data as they move through the system, allowing administrators to control the flow of network traffic based on a set of rules and policies. The main function of iptables is to provide firewall protection for the system. It allows administrators to create rules that determine which traffic is allowed to pass through the system and which traffic is blocked, based on criteria such as the source and destination IP addresses, port numbers, and protocol types. Iptables also supports network address translation (NAT), which allows multiple devices to share a single IP address, and can be used to implement port forwarding, which allows incoming network traffic to be redirected to a specific device or service on the network. We utilize Iptables on the gateway node to assign IP addresses to all the nodes in the mesh, and to route external internet traffic to every node in the mesh.

NOTE: Wi-Fi dongle setup will vary based on the Wi-Fi dongle that is selected. Based on the Wi-Fi dongle chosen for this project the following guide was used. Make sure to install these utilities except network manager before starting the next steps.

2.1.1 Node Setup

The following implementation will be completed on the latest version of Debian (Linux) that is available. This base setup should be completed on every node in the network as it will setup the Batman-adv service which will allow the implementation of the network. The following steps will implement the use of batctl which will allow for the management of the mesh network utilizing the Batman-adv service:

1. To manage the mesh network and monitor its functioning, the batctl utility needs to be downloaded with the following command:

```
sudo apt-get install -y batctl
```

2. Create a start up script file to run the necessary commands ~/start-batman-adv.sh:

```
nano ~/start-batman-adv.sh
```

It should contain the following:

```
#!/bin/bash
# batman-adv interface to use
sudo batctl if add wlan0
sudo ifconfig bat0 mtu 1468

# Tell batman-adv this is a gateway client
sudo batctl gw_mode client

# Activates batman-adv interfaces
sudo ifconfig wlan0 up
sudo ifconfig bat0 up
```

3. The script is made executable with the following command:

```
chmod +x ~/start-batman-adv.sh
```

4. Now that the mesh network commands are ready it is imperative to ensure that the OS start up doesn't take over the mesh network interface. To prevent this a network interface was created for the wlan0 (main) interface:

```
sudo nano /etc/network/interfaces.d/wlan0
```

And then inside the file, the wireless name and channel number can be changed to user liking and depending on the available bands in the user's country:

```
auto wlan0
iface wlan0 inet manual
wireless-channel 3
wireless-essid call-code-mesh
wireless-mode ad-hoc
```

Make sure these settings are consistent throughout all the nodes in the network. You can refer to this link to find the appropiate Wi-Fi bands in your country.

5. Stop the DHCP service from managing the wireless network by adding the following line to the end of the file /etc/dhcpcd.conf:

```
denyinterfaces wlan0
```

6. The startup script can be called on boot by editing /etc/rc.local as a root user:

```
sudo nano /etc/rc.local
```

```
/home/pi/start-batman-adv.sh &
```

Make sure that the last line has the following line: exit 0.

These are all the steps necessary to setup a node in the mesh network. The following sections will contain setup information on the two special nodes which provide connectivity to the rest of the network: the gateway node and the bridge node.

2.1.2 Gateway Setup

The following involves the procedure for converting one of the Raspberry Pi devices into a gateway node between the existing internet network and the Mesh network. We will enable connectivity to the internet through a Wi-Fi connection via the Wi-Fi USB dongle.

As the gateway uses IP routing to selectively allow traffic to pass between the Mesh and local internet networks the Mesh network needs to have a different address range. The instructions use the following network details:

- Network 192.168.199.x
- netmask 255.255.255.0
- gateway address 192.168.199.1

The DHCP server within the gateway will assume the role of providing network configuration for the mesh network. DHCP, which stands for Dynamic Host Configuration Protocol, is the service responsible for furnishing network settings to devices connected to a network. Since the mesh network functions independently of the home/office network, the DHCP service will supply devices with network configuration tailored specifically for the TCP network operating over the mesh.

On the selected Pi that will be the gateway complete the following steps on the command line:

1. Make sure to firstly install the DHCP software with the command:

   ```
   sudo apt-get install -y dnsmasq
   ```

2. You must configure the DHCP server to service the correct address range so as to not interfere with the local network. We achieve this by editing dnsmasq.conf as a root user:

   ```
   sudo nano /etc/dnsmasq.conf
   ```

   Add the following lines to the end of the file:

   ```
   interface=bat0
   dhcp-range=192.168.199.2,192.168.199.99,255.255.255.0,12h
   ```

3. To connect to a WiFi network the raspberry pi uses a configuration file /etc/wpa_supplicant/wpa_supplicant.conf. In order to connect properly on startup we have to setup the correct network name in this file to provide internet connection to the mesh network:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
        ssid="network_name"
        psk="password"
        key_mgmt=WPA-PSK

}
```

Make sure to change the ssid to the name of your internet network and the psk to the passcode of your network.

4. We now need to update traffic using the Iptables library to successfully forward the correct IP adresses to every node on the mesh. Firstly make sure to install Iptables using the following command:

```
sudo apt-get install iptables
```

Once that is installed change your start-batman-adv to forward the IP adresses from wlan0 to wlan1:

```
#!/bin/bash
# batman-adv interface to use
sudo batctl if add wlan0
sudo ifconfig bat0 mtu 1468

# Tell batman-adv this is an internet gateway
sudo batctl gw_mode server

# Enable port forwarding between eth0 and bat0
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o wlan1 -j MASQUERADE
sudo iptables -A FORWARD -i wlan1 -o bat0 -m conntrack --ctstate RELATED,ESTAI
sudo iptables -A FORWARD -i bat0 -o wlan0 -j ACCEPT

# Activates the interfaces for batman-adv
sudo ifconfig wlan0 up
sudo ifconfig bat0 up # bat0 is created via the first command
sudo ifconfig bat0 192.168.199.1/24
```

With these commands the gateway node should be all setup and ready to run when the Raspberry Pi is restarted.

2.1.3 Bridge Setup

As previously described the bridge node will provide connectivity to non-mesh devices through a Wi-Fi hotspot and ethernet connection. The gateway node provides the DHCP server that will also serve bridged devices, as DHCP requests flow over a bridge. The following steps are necessary to setup the bridge:

1. Install the bridge utilities command using the following command:

   ```
   sudo apt-get install -y bridge-utils
   ```

2. Connectivity for the ethernet interface can be completed by creating file /etc/network/interfaces.d/eth0 as root user. The interface will be set to hotplug so that that the ethernet can be disconnected and reconnected:

   ```
   sudo nano /etc/network/interfaces.d/eth0
   ```

   The contents should be set to:

   ```
   auto eth0
   allow-hotplug eth0
   iface eth0 inet manual
   ```

3. To create the Wi-Fi access point hostapd needs to be installed on the Pi:

   ```
   sudo apt-get install -y hostapd
   ```

4. To enable the Wi-Fi access point we will need to modify the contents of the hostapd start up script. Edit file /etc/hostapd/hostapd.conf as root user and set the content to:

   ```
   interface=wlan1
   driver=nl80211
   bridge=br0
   hw_mode=g
   channel=7
   wmm_enabled=0
   macaddr_acl=0
   auth_algs=1
   ignore_broadcast_ssid=0
   wpa=2
   wpa_key_mgmt=WPA-PSK
   wpa_pairwise=TKIP
   rsn_pairwise=CCMP
   ssid=raspimesh
   wpa_passphrase=passw0rd
   ```

   This will get the Wi-Fi access point ready for use on startup.

5. Modify the /etc/dhcpcd.conf file as root:

```
sudo nano /etc/dhcpcd.conf
```

   Change the last line to:

```
denyinterfaces wlan0 eth0 bat0
```

6. Update start-batman-adv to contain the following lines:

```
#!/bin/bash

# Tell batman-adv which interface to use
sudo batctl if add wlan0
sudo ifconfig bat0 mtu 1468

sudo brctl addbr br0
sudo brctl addif br0 bat0 eth0

# Tell batman-adv this is a gateway client
sudo batctl gw_mode client

# Activates the interfaces for batman-adv
sudo ifconfig wlan0 up
sudo ifconfig bat0 up

# Restart DHCP now bridge and mesh network are up
sudo dhclient -r br0
sudo dhclient br0
```

7. Enable the access point service on wlan1 interface using command:

```
sudo systemctl enable hostapd@hostapd.service
```

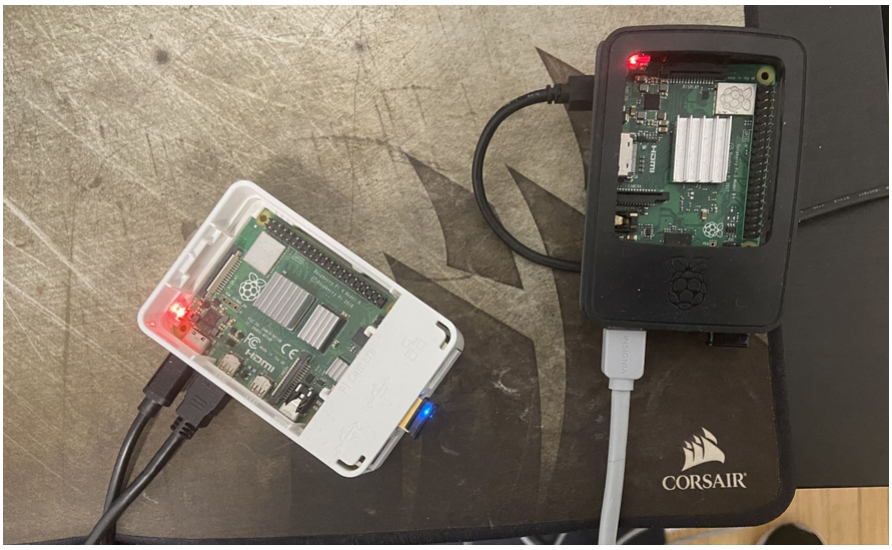

Figure 2.2: Physical Mesh Network Implementation

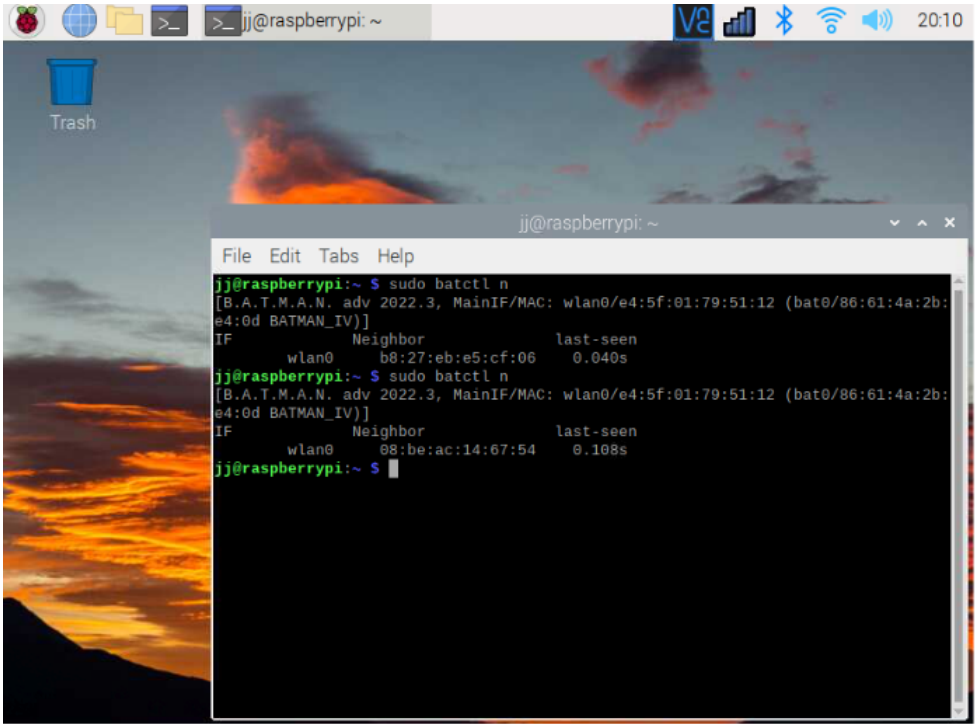

Figure 2.3: Working Gateway and Bridge Node

Chapter 3

Creating an IOT app and interface

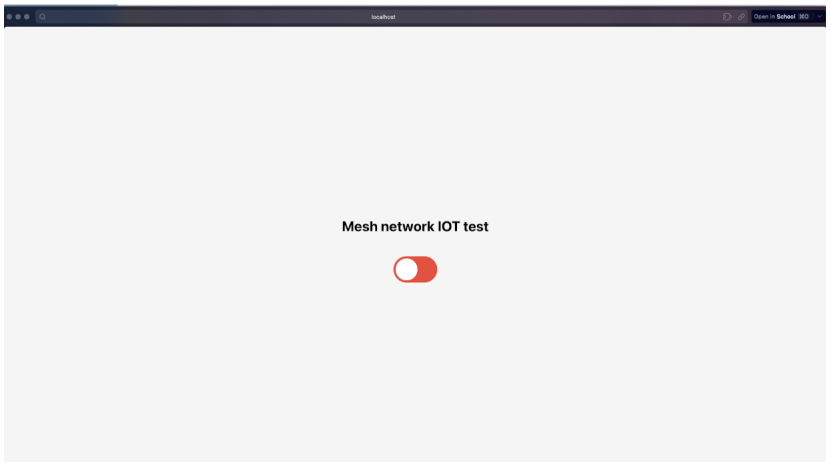

Figure 3.1: Website Screenshot

Building upon the foundation established in the previous chapter, we will now explore the implementation of an IoT application using a Raspberry Pi within the ad-hoc mesh network. The IoT application will be responsible for controlling a diode connected to the Raspberry Pi. The goal is to demonstrate the ease of integrating IoT devices into the mesh network and facilitate seamless communication and control of these devices.

3.1 IoT Application Implementation

To develop the IoT application, we utilize a Node.js Express server running on a Raspberry Pi. The Express server provides a RESTful API that exposes endpoints for controlling the diode, allowing clients to turn it on, turn it off, and retrieve its current status. In this section, we will discuss the key components of this application, as well as how it integrates with the mesh network.

As mentioned earlier, the IoT application is built using Node.js and Express, a popular web application framework for Node.js. Additionally, the onoff library is used to access and control the Raspberry Pi's General Purpose Input/Output (GPIO) pins, which in turn control the diode.

3.2 API endpoints

The Express server is configured to listen on port 4000, and exposes three API endpoints:

- **POST /on** This route turns on the diode by writing a '1' value to the GPIO pin. It then sends a success response back to the client with the message "Diode turned on."
- **POST /off** Similarly, this route turns off the diode by writing a '0' value to the GPIO pin. It responds with the message "Diode turned off."
- **GET /status** This route returns the current status of the diode. It reads the value from the GPIO pin and sends a response containing the status as "on" or "off.""

The IoT application is designed to work seamlessly within the mesh network. When deployed on a Raspberry Pi within the network, clients can access the diode control API by sending requests through the bridge node or gateway node. This allows clients to communicate with the IoT application even if they are not directly connected to it, taking advantage of the decentralized and resilient nature of the ad-hoc mesh network.

To ensure a graceful shutdown of the server and proper release of resources, a signal handler is included for the SIGINT event. When triggered, the handler turns off the diode, unexports the GPIO pin to free resources, and closes the server with a message indicating that the GPIO resources have been released.A screenshot of the code can be viewed at 3.2.

3.3 Website Implementation

To facilitate user interaction with the IoT application, a web interface was developed using React, a popular JavaScript library for building user interfaces. The web application calls the REST API exposed through the ad-hoc network to control the diode. In this section, we will discuss the structure and functionality of the web application, as well as how it interacts with the IoT application running on the Raspberry Pi within the mesh network.

The website has a toggle switch that represents the diode's status. When the toggle switch is clicked, the diode's status changes and an HTTP request is sent to the appropriate API endpoint to control the diode. A screenshot of the website can be viewed at 3.1. A screenshot of the code can be viewed at 3.3.

```
const express = require("express");
const Gpio = require("onoff").Gpio;

const app = express();
const port = 3000;

// Set the Raspberry Pi GPIO pin that the diode is connected to
const diodePin = 17;
const diode = new Gpio(diodePin, "out");

app.use(express.json());

app.post("/on", (req, res) => {
  diode.writeSync(1);
  res.status(200).send("Diode turned on");
});

app.post("/off", (req, res) => {
  diode.writeSync(0);
  res.status(200).send("Diode turned off");
});

app.get("/status", (req, res) => {
  const diodeStatus = diode.readSync() ? "on" : "off";
  res.status(200).send(`Diode is ${diodeStatus}`);
});

const server = app.listen(port, () => {
  console.log(
    `Raspberry Pi diode control server listening at http://localhost:${port}`
  );
});

process.on("SIGINT", () => {
  diode.writeSync(0); // Turn off the diode
  diode.unexport(); // Unexport GPIO and free resources
  server.close(() => {
    console.log("Server closed, GPIO resources released");
    process.exit(0);
  });
});
```

Figure 3.2: Server Code

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import styled from "@emotion/styled";
import "./App.css";

function App() {
  const [isOn, setIsOn] = useState(false);

  let deviceIp = "http://172.20.10.79:5000";
  const toggleSwitch = () => {
    setIsOn(!isOn);
  };

  async function turnOn() {
    await axios.post(deviceIp + "/" + "on");
  }

  async function turnOff() {
    await axios.post(deviceIp + "/" + "off");
  }
  async function getStatus() {
    await axios.get(deviceIp + "/" + "status");
  }

  useEffect(() => {
    if (isOn) {
      turnOn();
      console.log("turned on!");
    } else {
      turnOff();
      console.log("turn off!");
    }
  }, [isOn]);

  return (
    <Container>
      <Title>Mesh network IOT test</Title>
      <LightSwitch isOn={isOn} onClick={toggleSwitch}>
        <Switch isOn={isOn} />
      </LightSwitch>
    </Container>
  );
}
```

Figure 3.3: React Code

Index