Internet of Things Network Dashboard with Automatic Device Detection

Carleton University - COMP 4905
Honours Project Final Report

AJ Ricketts - 101084146
Supervised By: David Barrera

December 16th, 2022

Abstract

Given the rise in Internet of Things (IoT) devices in recent years, the number of ways a network can become compromised has increased considerably due to the less secure nature of these devices. This project proposes an Internet of Things network dashboard that aims to address some of the concerns consumers have with how their data is being handled by these devices and wish to gain more insight on the security of their home network. This project is broken up into three main parts, the automatic detection of IoT devices on a network, the analysis of network data, and a front facing dashboard for users to easily view information about the network in an easily understandable, digestible format. Although automatically detecting IoT devices on the network proves to be a challenging and complex task, this report investigates various approaches as to how to achieve this task. In addition, the design and implementation of the dashboard UI is also explored.

Table of Contents

List of Figures

1. Introduction

The main objective of this honours project was to provide ordinary (not directly in the tech field) people with the ability to gain more information on what is happening on their network regarding Internet of Things (IoT) devices. Many families now have smart home devices in their house to control various aspects of their home, whether that be their lights, blinds, coffee machine, thermostat, etc. Using all of these devices in conjunction usually comes with the help of a voice assistant so users don't have to reach for a device every time they want to interact with their IoT products. With all these devices connected to the web, users may begin to worry about how all this data regarding what they do, or even potentially what they say in their own homes is being used and protected.

1.1 Problem

Between those that are not as experienced with technology being concerned that their devices are constantly listening to them, to more experienced users that want to know what is happening to their data, the same problem with IoT devices emerges. Currently, there are minimal regulations on the development of these devices, and at an increasing rate, these kinds of devices are being targeted by hackers. With the cost of making smart objects becoming negligible, these problems will only become more widespread and intractable.

1.2 Goals

My goals for this project included two parts. First, to create a process to automatically detect IoT devices on a user's network and track the data going to and from these devices. Second, analyze that data, and display various findings on a Dashboard. Some of these findings would include; incoming and outgoing packets; where these devices were sending or receiving data from; information on the sender/receiver; etc.

1.3 Motivation

My motivation for this project came from my own experiences with IoT devices. Having members in my family that are less experienced with technology being afraid of using these kinds of devices for fear that they are being monitored, as well as my own curiosity regarding how these devices go about making sure user data is used for only what is necessary and is well protected. After brainstorming a general idea, my supervisor helped me formulate a more specific and functional plan for this project.

1.4 Objectives

The initial objective of this project was to first create an application that would be able to detect and gather all the necessary data from IoT devices on the network. Due to complications during development and time constraints, this part of the project was not completed. The next objective was to develop a dashboard where all of this data and the findings gathered could be displayed to a user in an easily digestible format.

2. Background

2.1 Background Information

This section will outline all the background information needed to understand the approach, complications, development, and results of this project. It will also cover related works and various other systems designed by others, that attempted to achieve a similar goal and strategies they used.

2.1.1 Wireshark, Tshark, and Pyshark

Wireshark is a free and open-source packet analyzer used for network troubleshooting and analysis. Wireshark can help users view network packets and display them at a granular level. This is a powerful tool that allows for a detailed view of the activity of the network. Users can view network packets from the previous packet captures using tools such as tcpdump, capture packets live directly from

Wireshark, as well as filter or search for packets using many criteria. For those who are looking to incorporate Wireshark functionality programmatically. Tshark is the terminal-oriented version of Wireshark used when a graphical user interface isn't necessary. On the same theme, Pyshark is the Python wrapper for Tshark that allows for packet capture and parsing from within Python.

2.1.2 MAC Address and OUI

A media access control (MAC) address is a unique identifier [1] assigned to a network interface controller that is used to identify individual devices on a network. This is often referred to as the hardware or physical address and is primarily assigned by device manufacturers. MAC addresses are typically represented as six groups of two hexadecimal digits, separated by a colon. The Organizational Unique Identifier (OUI) is the part of the MAC address that identifies the vendor of the network adapter. The OUI is the first three groups of the six-group field. Although MAC addresses are assigned to network controllers by the manufacturer, often these controllers come with the capability to change their MAC address.

2.1.3 Local IP vs Public IP

Both public and local IP addresses serve the same purpose, however, the difference is scope [2]. Internet Service Providers (ISPs) assign each customer a public IP address. When sending requests across the internet, this is the address others on the internet will see. However, each device on a private network will have a local IP address. This address is what a router will use to determine which device on the network packets should be delivered to.

2.1.4 *ARP*

Address Resolution Protocol (ARP) is a protocol used to resolve IP addresses to MAC addresses. Although each device on a local network is given a local IP address, for actual communication with some other device on a local network, devices will use ARP to determine the MAC addresses associated with those IP addresses. As local IP addresses are not static most of the time, this is an important part of communication over the network.

2.1.5 Nmap

Nmap is a network scanner used to discover hosts and services on a network. Some features of Nmap include host discovery, port scanning, TCP/IP stack fingerprinting, and more. One of the most commonly used features, host discovery, is achieved by sending TCP and/or ICMP requests across the network and seeing which hosts respond. ICMP (Internet Control Message Protocol) is a supporting protocol used by network devices typically used for diagnostic or control purposes.

2.1.6 nslookup

nslookup (name server lookup) is a command line tool for querying the domain name system to acquire the mapping between domain names, IP addresses, or other DNS records. Although, it is most commonly used to find the IP address that corresponds to a host, or the domain name that corresponds to an IP address.

2.2 Related Work

This section will cover other projects related to automatic IoT detection and the approach they took toward the problem. Each project attacked the problem differently and provided insight and inspiration into the possible ways this project could be approached.

2.2.1 NetScanIoT & Web IoT Detection (WID) Tool

The NetScanIoT and WID tools were developed by researchers at the European Organization for Nuclear Research (CERN). The goals of these tools were to aid in the automatic detection, and classification of IoT devices on the CERN network to test them for vulnerabilities. The motivation for the development of these tools remains consistent with that of this project, researchers were concerned about the data these devices collected and how access to the data collected and stored by these devices can aid criminals in gathering sensitive information, like patients' healthcare data or video footage of home security cameras [3].

NetScanIoT is a tool written in Python which pings the devices within the network and checks the ICMP message if the target is reachable. If the target is available, an nslookup is performed to determine the hostname. Devices were filtered by port scanning and then non-IoT devices were manually removed from the list of devices connected to the network. Figure 1 shows a graphical representation of the NetScanIoT tool.

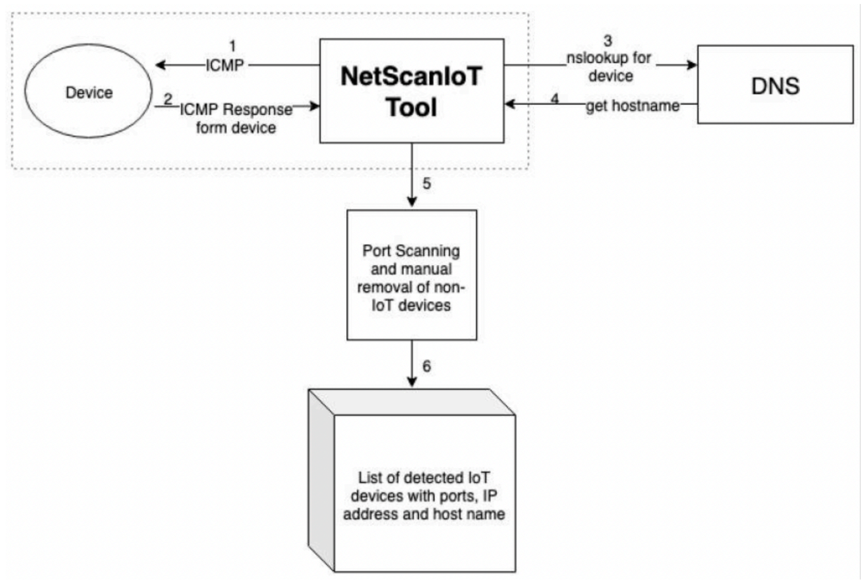


Figure 1: Overview of NetScanIoT tool

WID is another tool written in Python using Selenium in headless mode for web scraping. WID traverses the IoT device's web pages to determine information such as the model and the firmware version running on the device. Although the results of these tools were promising (identifying 92.45% of IoT device models and 100% of IoT devices that have a web user interface), this implementation is not a good option for a general use case, which will be discussed later in this report. Figure 2 shows a graphical representation of the WID tool.

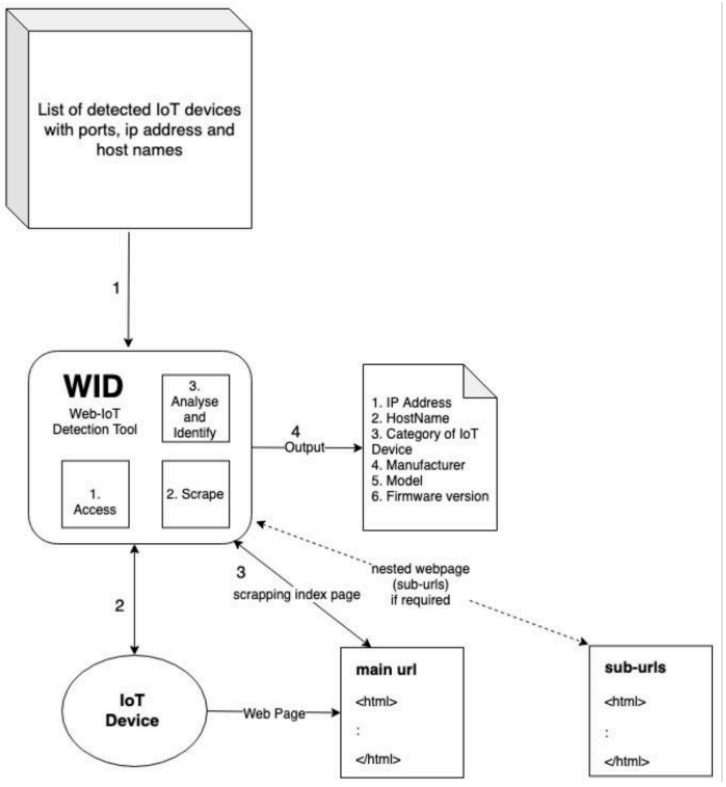


Figure 2: Overview of the WID tool.

2.2.2 *IoT Inspector Client*

IoT Inspector Client is a standalone desktop app that allows users to analyze their home IoT devices. IoT Inspector automatically sends out ARP packets to all IP addresses in the local subnet to discover devices. Once the discovery of all connected devices is complete, users must explicitly indicate which of the listed devices IoT Inspector is allowed to monitor (i.e., collect traffic). To help users choose what devices to monitor, IoT Inspector also displays the likely identities of individual devices, using external data sources such as the IEEE Organizationally Unique Identifier (OUI) database [4]. To capture data, display information and provide statistics to the user, IoT Inspector utilizes ARP spoofing. ARP spoofing allows the device IoT Inspector is running on to intercept all packets going to and from the user's router, and the device currently being monitored. Therefore, packets for all devices the user allows IoT Inspector to monitor would first be intercepted by the inspector, before being passed along to their destination.

2.2.3 Machine Learning for IoT Detection and Identification

Researchers have studied the use of machine learning to identify IoT devices with rogue device detection to secure the IoT ecosystem. One such study [5] looked at various technologies and methods for this task. As made clear in the paper, even though machine learning (ML) and Deep Learning (DL) have the potential to automatically discover distinctive latent features for accurate device identification, state-of-art algorithms require intensive modifications to be utilized in IoT. As IoT devices communicate with remote service providers through the REST API, researchers have highlighted that using only port numbers, domain names, and cipher suites, a Naive Bayesian classifier can reach high accuracy in classifying several commercial IoT devices [6]. However, this may not be able to be used for all devices as some interact with anonymous service providers. In this case, authors had to take various approaches using complex algorithms and statistical models to achieve a high rate of accuracy.

3. Approach

The following section will outline the approach I took to develop the automatic device detection system, device monitoring, and dashboard. This will cover my thought process along the way, the complications and complexities of automatic device detection and device monitoring that led to those sections of the project not being implemented in the final version, as well as the final implementation of the dashboard.

3.1 Attempts

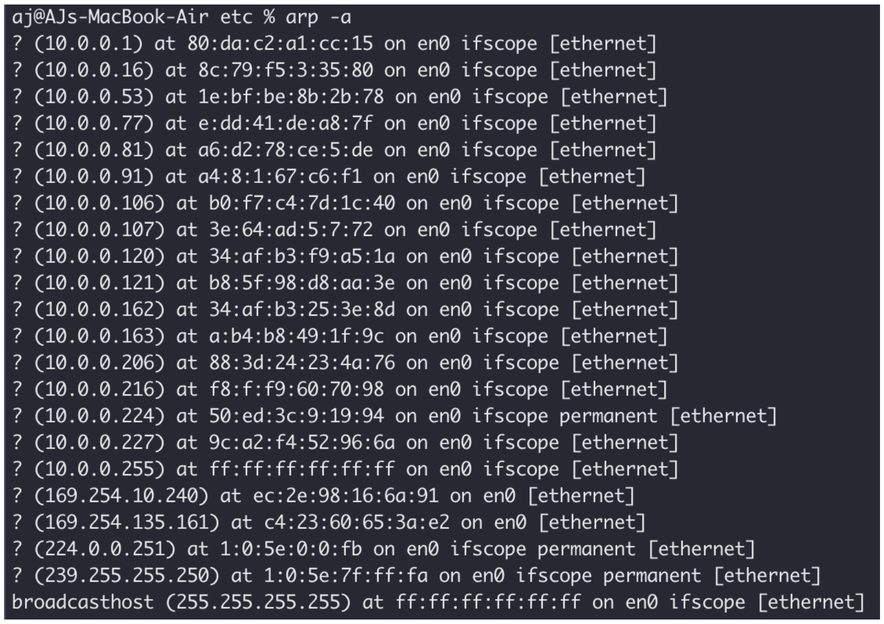
Between automatic IoT device detection, and device activity monitoring, there have been many approaches that were considered. This section aims to highlight some of those ideas.

3.1.1 Overall Device Discovery

The first step to automatically identifying IoT devices on the network was to produce a list of all devices connected to the network. Users can normally see a list of all connected devices by simply logging into their router admin page. Doing this can give users access to information such as information on connected devices, their local network configuration, and firewall settings, as well as advanced features such as port forwarding and more.

Utilizing the information on these pages would undoubtedly allow for an easier process of filtering out which devices on the network should be classified as IoT. However, accessing any of the information on this admin page would require a program to use some sort of automation (whether that be Selenium, Puppeteer, etc.) to pretend to be the user on the web, log in to the admin page, and then scrape the information needed. Providing admin access to their router to a 3rd party program is likely something most users are not comfortable with (if they're even aware of how to log in). In addition, taking this route would introduce other security requirements for ensuring users' router login information remained secure. Because of these reasons, this approach was not taken.

Although, a simpler method to gaining similar information is running the arp -a command through the command line. This command simply displays the current ARP table stored on that device. Each device that's connected to a network has its own ARP table, responsible for storing the address (MAC & IP) pairs that a specific device has communicated with. Figure 3 displays the output of the arp -a command which provides both the local IP address of the device, as well as the MAC address.



```
aj@AJs-MacBook-Air etc % arp -a
? (10.0.0.1) at 80:da:c2:a1:cc:15 on en0 ifscope [ethernet]
? (10.0.0.16) at 8c:79:f5:3:35:80 on en0 ifscope [ethernet]
? (10.0.0.53) at 1e:bf:be:8b:2b:78 on en0 ifscope [ethernet]
? (10.0.0.77) at e:dd:41:de:a8:7f on en0 ifscope [ethernet]
? (10.0.0.81) at a6:d2:78:ce:5:de on en0 ifscope [ethernet]
? (10.0.0.91) at a4:8:1:67:c6:f1 on en0 ifscope [ethernet]
? (10.0.0.106) at b0:f7:c4:7d:1c:40 on en0 ifscope [ethernet]
? (10.0.0.107) at 3e:64:ad:5:7:72 on en0 ifscope [ethernet]
? (10.0.0.120) at 34:af:b3:f9:a5:1a on en0 ifscope [ethernet]
? (10.0.0.121) at b8:5f:98:d8:aa:3e on en0 ifscope [ethernet]
? (10.0.0.162) at 34:af:b3:25:3e:8d on en0 ifscope [ethernet]
? (10.0.0.163) at a:b4:b8:49:1f:9c on en0 ifscope [ethernet]
? (10.0.0.206) at 88:3d:24:23:4a:76 on en0 ifscope [ethernet]
? (10.0.0.216) at f8:f:f9:60:70:98 on en0 ifscope [ethernet]
? (10.0.0.224) at 50:ed:3c:9:19:94 on en0 ifscope permanent [ethernet]
? (10.0.0.227) at 9c:a2:f4:52:96:6a on en0 ifscope [ethernet]
? (10.0.0.255) at ff:ff:ff:ff:ff:ff on en0 ifscope [ethernet]
? (169.254.10.240) at ec:2e:98:16:6a:91 on en0 [ethernet]
? (169.254.135.161) at c4:23:60:65:3a:e2 on en0 [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:7f:ff:fa on en0 ifscope permanent [ethernet]
broadcasthost (255.255.255.255) at ff:ff:ff:ff:ff:ff on en0 ifscope [ethernet]
```

Figure 3: Output of arp -a command in terminal

As this information needs to be accessed programmatically, another approach was briefly considered. As mentioned earlier in the report, Nmap is a network scanner used to discover hosts and services on a network. Not only is Nmap used on the command line, but it is also easily used within Python using libraries such as python3-nmap. Despite nmap being a powerful tool, there were consistency issues with generating a complete list of devices. Running a network scan would occasionally return missing between 1-3 connected devices. This could be for various reasons, such as a device not responding to a ping during the network scan, therefore reporting as down, network configuration issues on different systems or other bugs that a user could run into that are specific to their network/device setup.

Because of this, the decision was made to transition back to using ARP tables, this time through python. There are multiple ways to run command line commands through python, but the simplest is using the subprocess module with the check_output method. This method returns a bytes object in Python3 which can be converted to ASCII with output.decode("ascii"). Using this method we can achieve the same result as shown in Figure 1.

3.1.2 IoT Device Filtering

Once all devices connected to the network are found, the next step is to filter out the non-IoT devices. This filtering process is where major complications are found, and the solution proved to be quite complex. The following section will explore various options that were considered, in the order they were investigated.

3.1.2.1 2.4 vs 5GHz Frequencies

In 2022, routers that consumers purchase can operate on both the 2.4 and 5GHz wifi bands, in addition, nearly all products consumers buy can operate on either frequency. IoT devices are an exception to this with most manufacturers choosing to force users to connect these devices to the 2.4GHz band. Because of this, detecting which devices were connected to this frequency was the first attempt at filtering out non-IoT devices. Most devices that are capable of connecting to both, opt to connect to the 5GHz frequency before 2.4. Of course, there may be non-IoT devices still connected to the 2.4GHz band, but this would considerably narrow down the list. With a shorter list, another method could be employed to further filter the list, or users could simply remove the few (if any) non-IoT devices remaining in the list.

Looking more into separating devices based on what frequency they were connected to proved not to be a great option for multiple reasons. First, there is no stored list or any way of scanning for which devices are connected to which band like there is for simply discovering devices on the network. Because of this two options are left, attempting to discover what band devices are using through packet captures, or utilizing the router admin page as previously mentioned. Considering the fact that the majority of modern computers only come equipped with one wireless adapter, and therefore one radio, capturing packets on two frequency channels at the same time is not possible. Attempting to take this route would either require additional hardware (another computer) or specifically a Linux machine (as you can use many wifi adapters at once) and an additional wifi adapter (USB for example).

As requiring additional hardware to run the program shouldn't be a requirement (as it brings in extra complexity for everyday users) and gaining access to the user's router admin page is also not an option, for reasons outlined above, this did not seem like a viable approach.

3.1.2.2 OUI Lookup

After becoming familiar with the IoT Inspector program and their approach to discovering IoT devices, using the IEEE Organizationally Unique Identifier (OUI) database was considered. Although this technique seemed promising, it was soon clear that using the OUI database alone could not identify who devices were manufactured by. It seemed increasingly likely that many devices connected to a network would return a valid result from an OUI lookup. This is due to one of three reasons, the address wasn't registered with IEEE (this can be the case with cheaper products made by small brands), the device has a feature enabled that masks the MAC address (which is normally enabled by default), or the user manually changed the MAC address of the device in question. Throughout testing, approximately one-third of devices were not identifiable using an OUI lookup. Additionally, many vendors produce IoT devices in conjunction with other types of devices. Between the returned vendor from the OUI lookup producing multiple kinds of devices, and not receiving a vendor from the lookup at all, it is clear this method must be used to supplement another method rather than used by itself.

3.1.2.3 NetScanIoT & nslookup

As mentioned in the related works section, the NetScanIoT utilized nslookup to query a DNS to receive the hostname for devices connected to the network. This was followed up by their WID tool that would visit the manufactures website and scrape web pages to discover the device's name, model, and firmware version. This approach was briefly considered but proved to not be a viable option due to the differences in the networks being tested on. Due to the program being developed for this project being run within normal household networks, IP addresses received from preliminary device detection would be using local IP addresses. Because of this, performing a nslookup would not return hostnames.

3.1.2.4 Machine Learning for IoT Identification

As mentioned in the related works section, using machine learning for the classification and identification of IoT devices has proven to be an accurate approach. Despite this appearing to be the most consistent and promising approach, the actual implementation of these types of systems remains to be rather complex. Accurate machine learning methods typically include large data sets of network packets (33,468 data instances in one study [7], and 50,378 labelled instances in another [8]) and require feature engineering [9], i.e., the process of selecting, manipulating, and transforming raw data into features that can be used in supervised learning [10]. This process usually requires manual input from IoT and machine / deep learning domain experts and is a time-consuming process. Additionally, multiple-stage models can be needed, which means the architecture of these methods can be complex. Because of the complexities of implementing these types of systems, and the time and equipment required, this approach was not taken. However, it does appear to be the most promising in terms of maintaining accuracy in a general use case.

3.1.3 Packet Analysis

3.1.3.1 Gathering Information on Packet Source / Destination

One of the smaller goals of this project was to identify useable information on where devices were sending or receiving packets from. After studying packet captures of IoT devices, this proved to be a more challenging objective than originally anticipated. These challenges stem from the decision to capture packets in monitor mode on a single device. Using this approach, packets that come in will be Layer 2 802.11 packets. Normally capturing packets (when you're only concerned about packets going to and from the machine you're capturing from, i.e., not in monitor/ promiscuous mode) brings in layer 3. These layer 3 (the network layer) packets include source and destination IP addresses which could be used to trace the path packets took using the **traceroute** command. Traceroute requires IP addresses as it exploits IP's TTL (Time to Live) feature, however, 802.11 adapters often transform 802.11 data packets into fake Ethernet packets before supplying them to the host, and, even if they don't, the drivers for the adapters often do so before supplying the packets to the operating system's networking stack and packet capture mechanism [11]. Because of this, accessing the layer 3 packet information is not possible, and therefore running a traceroute on these packets does not seem achievable. The most obvious solution to this seemed to be the approach the developers of IoT Inspector Client took. As described in the related works section, IoT Inspector Client utilized ARP spoofing to intercept packets rather than utilizing a feature such as monitor / promiscuous mode. Even though using this technique would have allowed for the capture of layer 3 packets, this approach did not seem like the best option for the reasons previously mentioned. Although there may be other methods of exploring where packets are being sent to, or where they are being received from, this was unfortunately not explored in this project.

3.2 Design

The dashboard UI element of this project was developed using React JS with the file and component structure organized with industry standards in mind. Some of the main packages used in this application include Material UI, nivo, and react-pro-sidebar. Material UI was used for various components, layouts, theming, and icons. Nivo is the package that provides the bar chart for the network statistics page, although many of their charts were imported for potential future use. Lastly, react-pro-sidebar was used to implement the sidebar and its functionality. The file structure is broken down into four main folders, assets, components, data, and pages and can be seen in Figure 4 below (Each folder in pages has an index.js file, which is only shown below in one folder for readability).

In addition, there are four standalone files, App.jsx, index.css, main.jsx, and theme.js that provides styles to the React app as well as attach the React App to the DOM. More specifically, App.jsx handles placing the side and top bars on the page, ensures the Material UI theme is applied to the whole app, as well as specifies the routes for each page. Index.css provides general styling to use for the entire app, including backup fonts to use, and scrollbar styling. Main.jsx is only used for attaching the React app and routes to the DOM. Finally, theme.js defines the colours (mostly generated using the tailwind shades extension with some manual configuration) used for the applications themes. The themes available are light and dark mode and can be toggled by clicking the sun or moon icon (depending on which mode you currently are in). This icon is housed in the Topbar and can be seen in the multiple screenshots in the Implementation section below. In terms of layout and styling, each page utilizes one or more Material UI box components as a wrapper for most of the CSS utility needs. In addition, the Typography component is used to present and style text content efficiently. More information on how information is populated on each page can be seen in the next section.

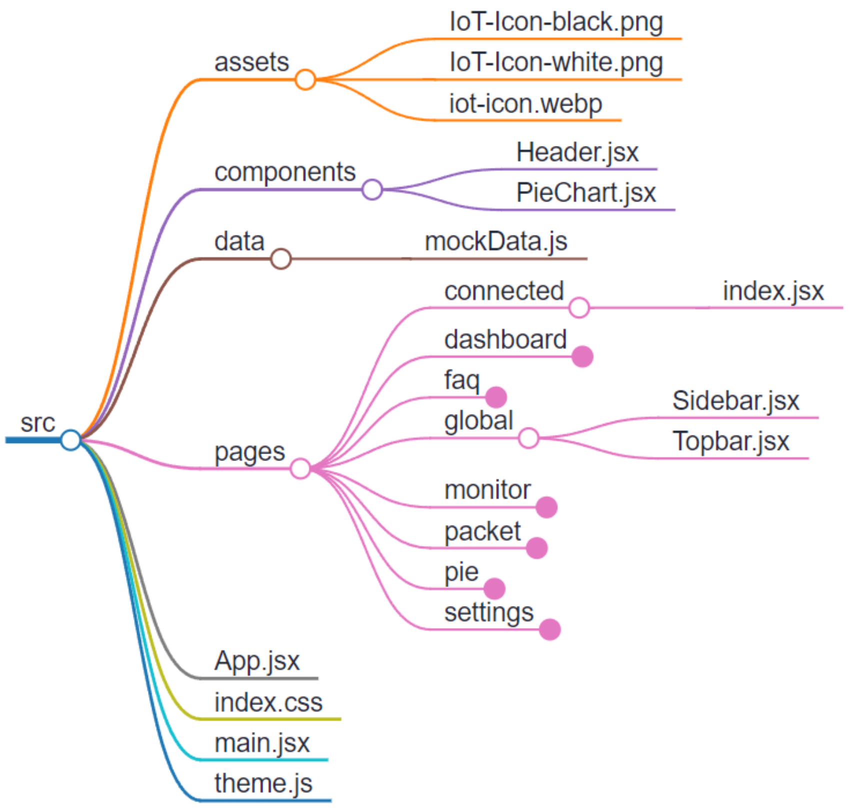


Figure 4: Dashboard File Structure

3.2.1 Sidebar

As discussed above, the sidebar was developed using react-pro-sidebar. This is a sidebar component that includes all necessary sidebar functionality including, collapsibility, nested menus, and styling. In the collapsed state the sidebar displays only the various icons for the menu headings, and in the expanded state displays the title and icon for the dashboard, and all menu headings and section labels. The sidebar is always visible and maintains its collapsed or expanded state when switching between pages.

3.2.2 Topbar

The Topbar is the small section at the top of the React app, visible on every page, that includes the light/dark mode theme icon, a settings icon that takes you to the settings page, and a user icon (this icon currently does not have any functionality, but is a placeholder for the future implementation of user accounts/login capabilities).

3.2.3 Header

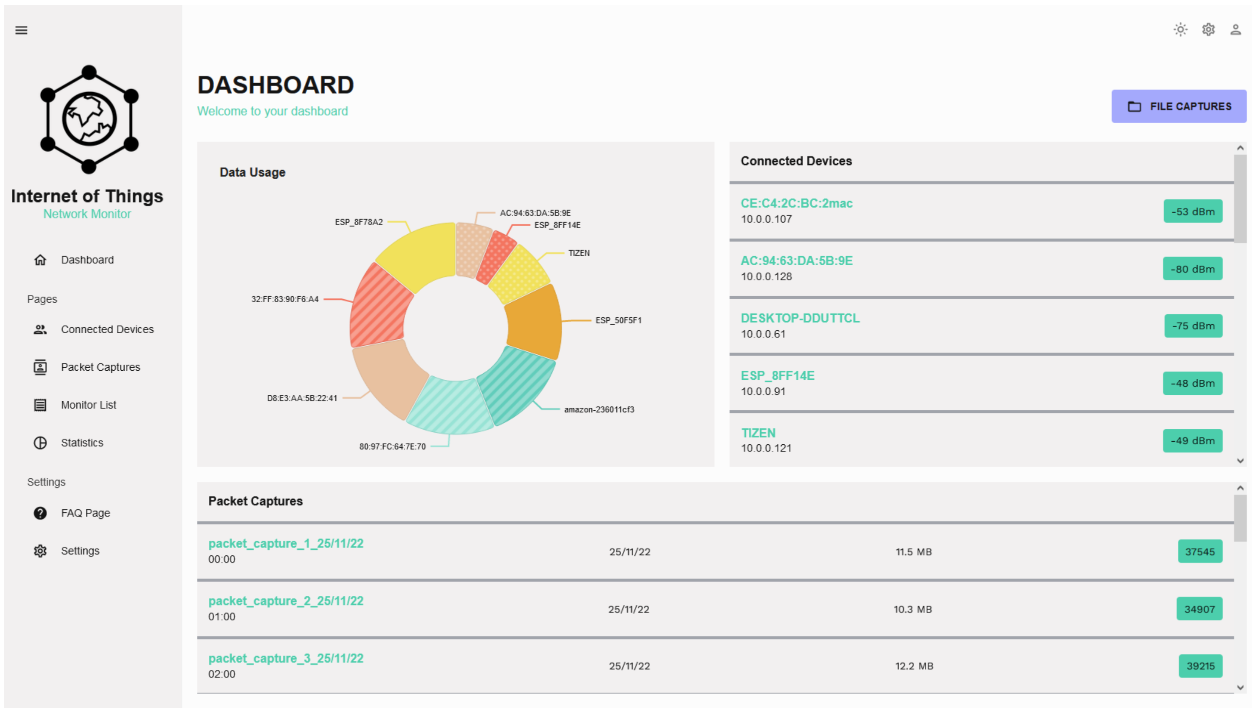
The Header is a short component that displays the title and subtitle on each page. As each page was going to have both a title and subtitle, this component was made to reduce repeated code.

3.3 Implementation

The dashboard has various pages, each with their own specific function. This section will explore each page, and explain the intended use of each page, and the data that should be made available to the user. Although it is currently populated with mock data, this should serve as a clear layout for future implementation using live data.

3.3.1 Main Screen

The main screen of the local webpage serves as an overview of the information the user is most likely to be interested in. This information is a miniaturized version of information that is also available on other pages. This includes a graphical representation of the calculated statistics, currently, this is a visualization of the share of the total data recorded used by each device. However, this could be represented in another format or be replaced by a different statistic. In addition, two lists are also populated on the main page, a list of connected devices, as well as a list of packet captures that have been collected so far that day. Both lists will be broken down in more detail below. The entire page is laid out using a grid with 12 columns and 8 rows. The Data Usage and Connected Devices sections both use 6 columns and 3 rows each with a 20-pixel gap in between them, the Packet Capture section uses all 12 columns and 2 rows to fill up the bottom of the screen. The Data usage section simply uses a smaller version of the PieChart component, while the Connected Devices and Packet Capture sections use data from the mockData.js file, and the map function to iterate through the mockDataConnected and mockDataPacket arrays to populate the lists respectively.



Connected Devices

CE:C4:2C:BC:2mac 10.0.0.107	-53 dBm
AC:94:63:DA:5B:9E 10.0.0.128	-80 dBm
DESKTOP-DDUTTCL 10.0.0.61	-75 dBm
ESP_8FF14E 10.0.0.91	-48 dBm
TIZEN 10.0.0.121	-49 dBm

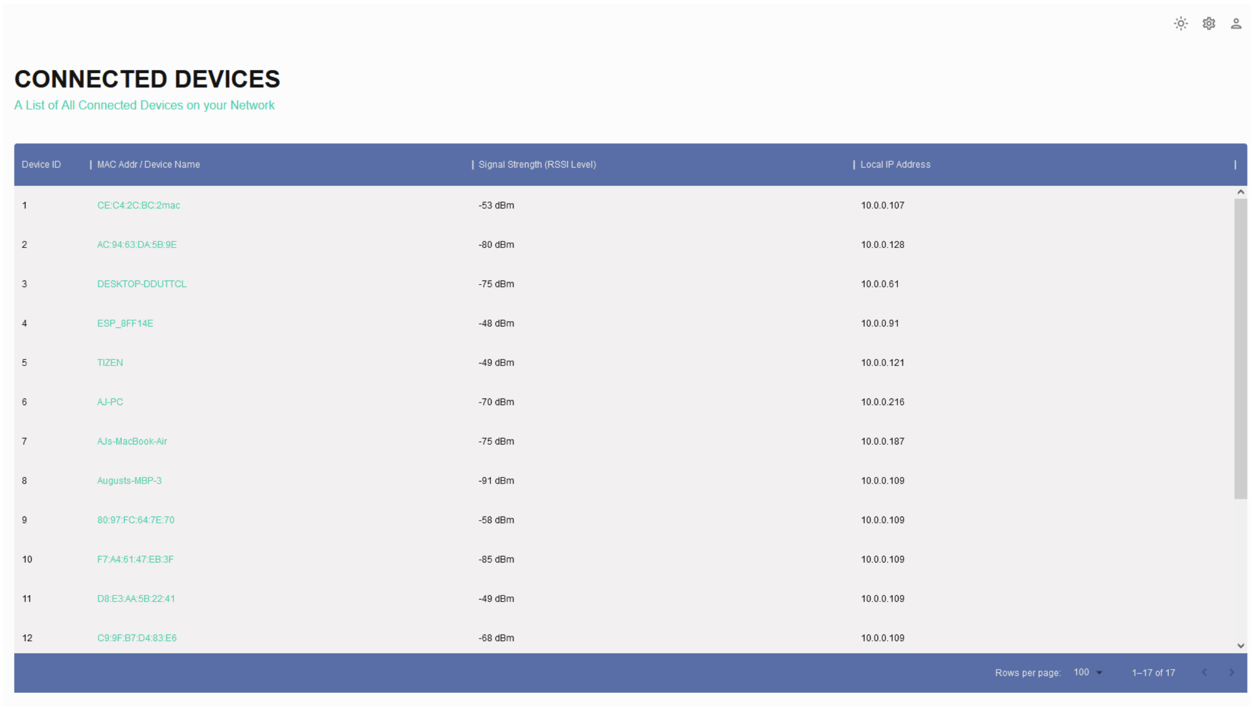
Packet Captures

packet_capture_1_25/11/22 00:00	25/11/22	11.5 MB	37545
packet_capture_2_25/11/22 01:00	25/11/22	10.3 MB	34907
packet_capture_3_25/11/22 02:00	25/11/22	12.2 MB	39215

Figure 5: Default Dashboard Page

3.3.2 Connected Devices Page

The connected devices page will display a list of all connected devices on the network. As a map of the network should have been made before filtering out non-IoT devices, a list of all connected devices can easily be stored to populate the above list. This list also reports a few facts about the devices in this list including RSSI level and the local IP addresses. More information about these devices can easily be added here in additional columns if desired. The connected devices' data is presented using a Material UI DataGrid which acts as a container for the device information and provides features such as scrolling, paging, and selecting the number of rows per page. The data for the data grid is pulled from the mockConnectedData array from the mockData.js file.



CONNECTED DEVICES

A List of All Connected Devices on your Network

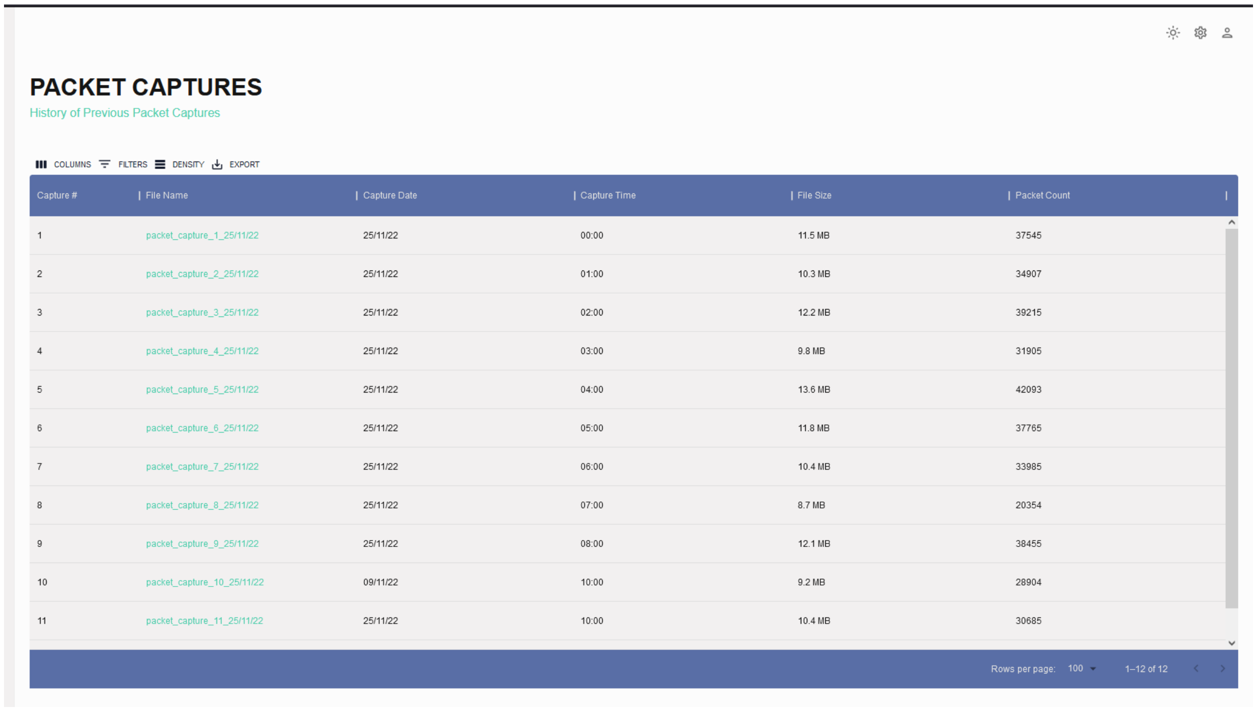
Device ID	MAC Addr / Device Name	Signal Strength (RSSI Level)	Local IP Address
1	CE:C4:2C:BC:2mac	-53 dBm	10.0.0.107
2	AC:94:63:DA:5B:9E	-80 dBm	10.0.0.128
3	DESKTOP-DDUTTCL	-75 dBm	10.0.0.61
4	ESP_8FF14E	-48 dBm	10.0.0.91
5	TIZEN	-49 dBm	10.0.0.121
6	AJ-PC	-70 dBm	10.0.0.216
7	AJs-MacBook-Air	-75 dBm	10.0.0.187
8	Augusts-MBP-3	-91 dBm	10.0.0.109
9	80:97:FC:64:7E:70	-58 dBm	10.0.0.109
10	F7:A4:61:47:EB:3F	-85 dBm	10.0.0.109
11	D8:E3:AA:5B:22:41	-49 dBm	10.0.0.109
12	C9:9F:B7:D4:83:E6	-68 dBm	10.0.0.109

Rows per page: 100 1–17 of 17

Figure 6: Connected Devices Page

3.3.3 Packet Capture Page

The packet capture page (Figure 7) displays a list of all packet captures generated so far that day. Ideally, multiple packet captures should occur daily to get a general overview of the network throughout the day. These packet captures should be listed here so the user is aware of what the data being presented on the dashboard is based on. Each entry in this list includes the name of the packet capture, the date, time and file size of the capture, and the number of packets collected. All of this information is presented using a Material UI DataGrid component, similar to the connected devices page. On top of the Data Grid is a Material UI GridToolbar to provide the user with the ability to sort the list by column title, use various filters, as well as exporting the packet captures. As the dashboard will be run locally, the export option should take them to the location in their filesystem where the packet captures are stored. The data for the packet captures presented in the data grid is once again pulled from the mockData.js file, more specifically, the mockDataPacket array.



PACKET CAPTURES

History of Previous Packet Captures

COLUMNS FILTERS DENSITY EXPORT

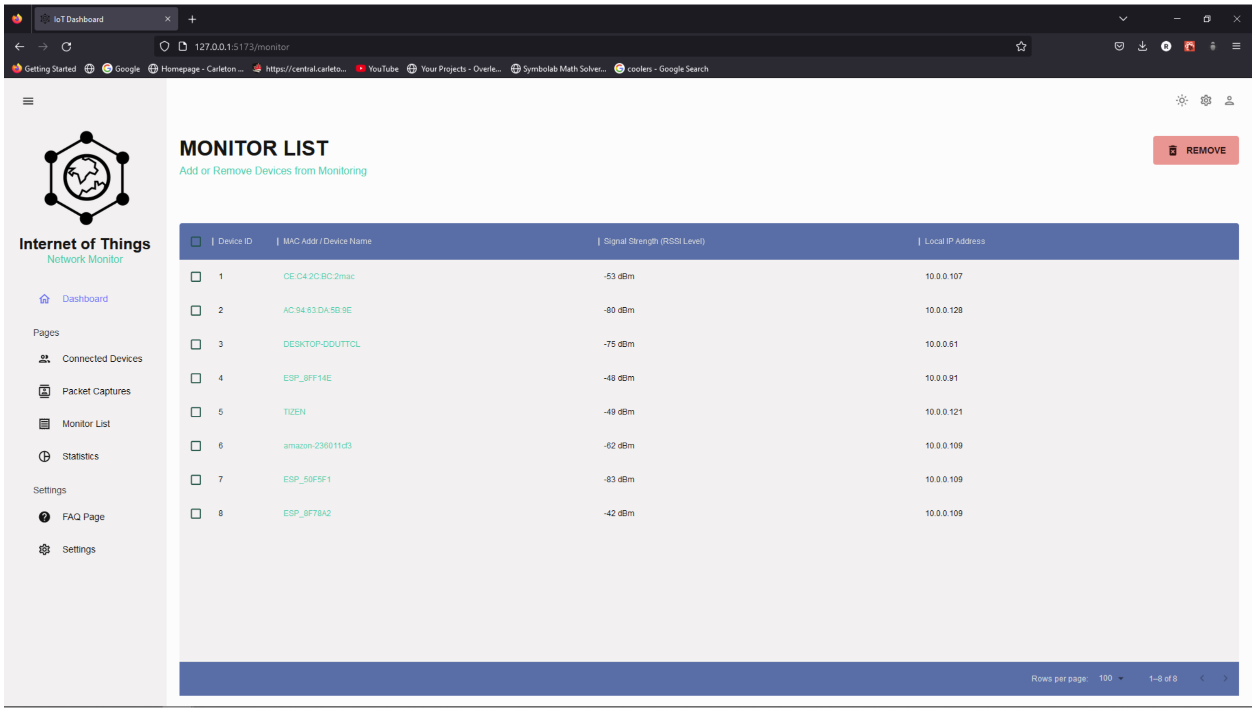
Capture #	File Name	Capture Date	Capture Time	File Size	Packet Count
1	packet_capture_1_25/11/22	25/11/22	00:00	11.5 MB	37545
2	packet_capture_2_25/11/22	25/11/22	01:00	10.3 MB	34907
3	packet_capture_3_25/11/22	25/11/22	02:00	12.2 MB	39215
4	packet_capture_4_25/11/22	25/11/22	03:00	9.8 MB	31905
5	packet_capture_5_25/11/22	25/11/22	04:00	13.6 MB	42093
6	packet_capture_6_25/11/22	25/11/22	05:00	11.8 MB	37765
7	packet_capture_7_25/11/22	25/11/22	06:00	10.4 MB	33985
8	packet_capture_8_25/11/22	25/11/22	07:00	8.7 MB	20354
9	packet_capture_9_25/11/22	25/11/22	08:00	12.1 MB	38455
10	packet_capture_10_25/11/22	09/11/22	10:00	9.2 MB	28904
11	packet_capture_11_25/11/22	25/11/22	10:00	10.4 MB	30685

Rows per page: 100 1–12 of 12

Figure 7: Packet Capture Page

3.3.4 Monitor List Page

The Monitor List page shows the user a list of the current devices that are being monitored. This page is very similar to the connected devices page as it displays some of the same information about devices; device name or MAC address, RSSI level, and local IP address. The devices in this list should all be devices that were identified to be IoT devices. This page also gives users the ability to remove devices from this list. Removing devices from being monitored would be accomplished by selecting the desired device using the checkbox in the far left column, and then clicking on the remove button. This could occur because the user identifies the device to not be an IoT device, or simply because they don't want the device to be tracked for personal preference. In a similar fashion to the previous pages, the Monitor List page utilizes the DataGrid component, however this time with the checkboxSelection attribute to enable the user to select an element from the list as previously mentioned. The data is pulled from the mockDataMonitor array in the mockData.js file, similar to the previous two pages. A screen capture of this device can be seen below in figure 8.



MONITOR LIST

Add or Remove Devices from Monitoring

Device ID	MAC Addr / Device Name	Signal Strength (RSSI Level)	Local IP Address
1	CE:C4:2C:BC:2mac	-53 dBm	10.0.0.107
2	AC:94:63:DA:5B:9E	-80 dBm	10.0.0.128
3	DESKTOP-DDUTTCL	-75 dBm	10.0.0.61
4	ESP_8FF14E	-48 dBm	10.0.0.91
5	TIZEN	-49 dBm	10.0.0.121
6	amazon-236011cf3	-62 dBm	10.0.0.109
7	ESP_50F5F1	-83 dBm	10.0.0.109
8	ESP_8F78A2	-42 dBm	10.0.0.109

Figure 8: Monitor List Page

3.3.5 Statistics Page

The Statistics page includes a graphical representation of statistics to display to the user in an easily digestible format. Currently, this is a singular pie chart with mock data outlining the share of the total data recorded used by each device. However, this page could include different / more graphs or information for the user to take note of.

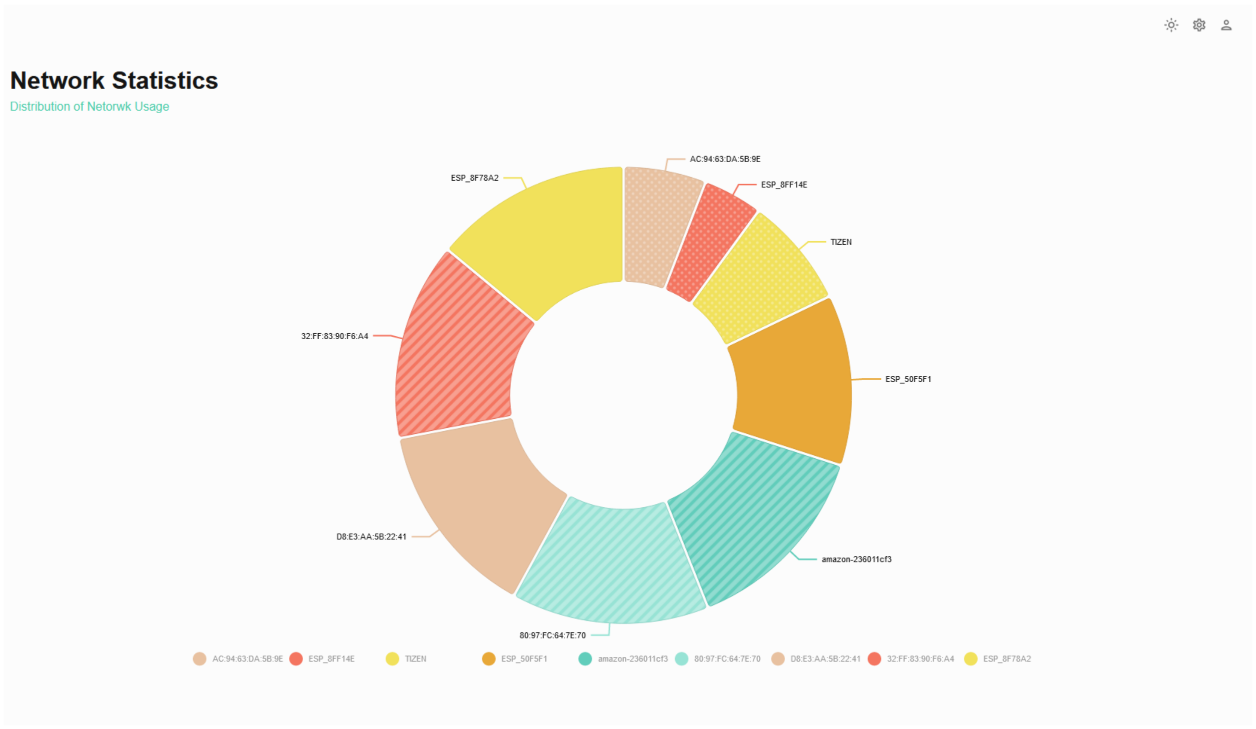
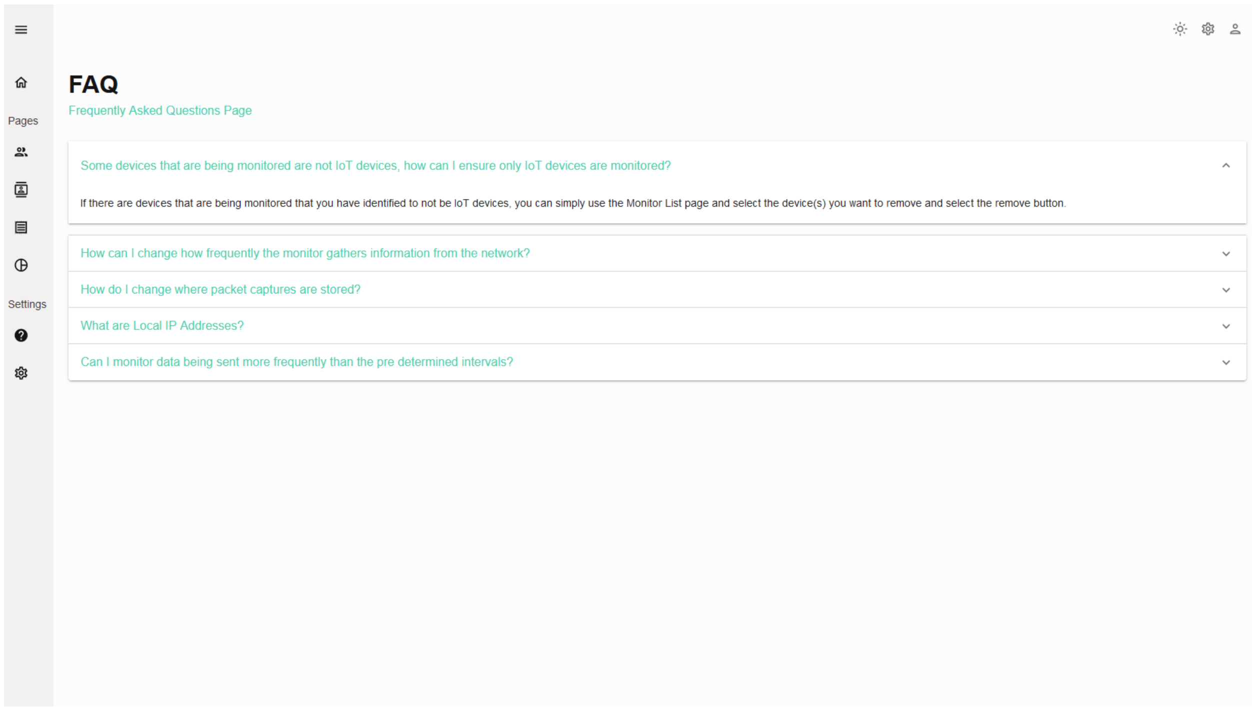

Network Statistics
Distribution of Netorwk Usage
AC:94:63:DA:5B:9E
ESP_8FF14E
TIZEN
ESP_50F5F1
amazon-236011cf3
80:97:FC:64:7E:70
D8:E3:AA:5B:22:41
32:FF:83:90:F6:A4
ESP_8F78A2
● AC:94:63:DA:5B:9E ● ESP_8FF14E ● TIZEN ● ESP_50F5F1 ● amazon-236011cf3 ● 80:97:FC:64:7E:70 ● D8:E3:AA:5B:22:41 ● 32:FF:83:90:F6:A4 ● ESP_8F78A2

Figure 9: Statistics Page

Some of the ideas of what could be on this page originally included the security status of each device (secured, potentially vulnerable, vulnerable, compromised), and possibly a geographical map of locations where data is being sent (if that information was acquired). The chart on this page is generated using Nivo charts with the chart data populated from mockPieData in mockData.js.

3.3.6 FAQ Page

The FAQ Page includes some answers to questions that the user may ask. This page can be especially useful to those who are not familiar with networking or have very limited knowledge. Currently, there are five placeholder questions and answers, this page could always be expanded to include more questions about the dashboard, or even general networking questions. Another idea for this page originally included setup walkthroughs for any advanced features that could be implemented in the dashboard in the future. The questions on this page are placed in Material UI Accordion components. This provides the drop-down functionality seen in figure 10 below.



FAQ

Frequently Asked Questions Page

Some devices that are being monitored are not IoT devices, how can I ensure only IoT devices are monitored?

If there are devices that are being monitored that you have identified to not be IoT devices, you can simply use the Monitor List page and select the device(s) you want to remove and select the remove button.

How can I change how frequently the monitor gathers information from the network?

How do I change where packet captures are stored?

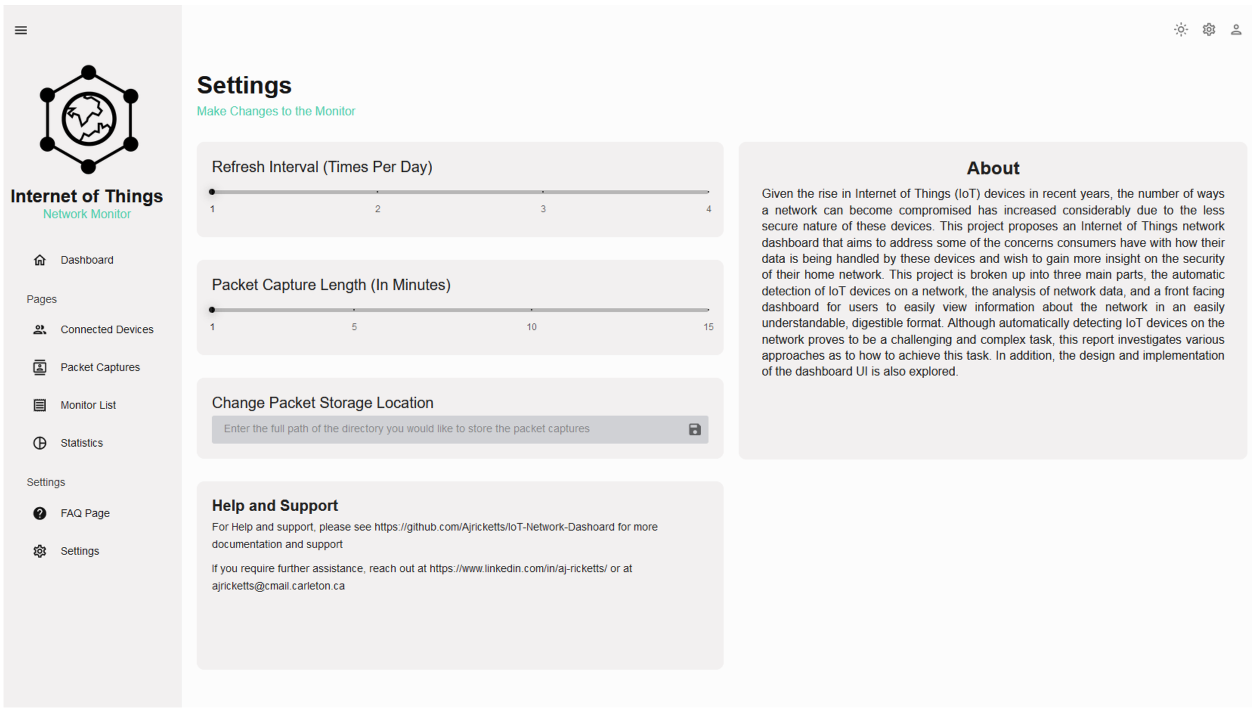
What are Local IP Addresses?

Can I monitor data being sent more frequently than the pre determined intervals?

Figure 10: Frequently Asked Questions Page

3.3.7 Settings Page

The Settings page is the final page of the dashboard. The settings page includes various options for the user to change the functionality on how many times per day information on the network is gathered (Refresh Interval), how long the packet captures should last (Packet Capture Length), as well as the location where packet captures should be stored. The original design for the system was to have packets captured for 10 minutes at the beginning of every hour. The current placeholders for the refresh interval range from one to four times a day, however, this can easily be changed to include a greater or smaller range. Similarly, the packet capture length option has placeholders beginning at 1 minute and increasing to 15 minutes. The change packet storage location option allows the user to input a full directory path of where they would like to change where the packet captures are stored. Finally, there are the about and help and support sections. The help and support section directs the user to get into contact with the developer for more help



Settings

Make Changes to the Monitor

Refresh Interval (Times Per Day)

1 2 3 4

Packet Capture Length (In Minutes)

1 5 10 15

Change Packet Storage Location

Enter the full path of the directory you would like to store the packet captures

Help and Support

For Help and support, please see https://github.com/Ajricketts/IoT-Network-Dashoard for more documentation and support

If you require further assistance, reach out at https://www.linkedin.com/in/aj-ricketts/ or at ajricketts@cmail.carleton.ca

About

Given the rise in Internet of Things (IoT) devices in recent years, the number of ways a network can become compromised has increased considerably due to the less secure nature of these devices. This project proposes an Internet of Things network dashboard that aims to address some of the concerns consumers have with how their data is being handled by these devices and wish to gain more insight on the security of their home network. This project is broken up into three main parts, the automatic detection of IoT devices on a network, the analysis of network data, and a front facing dashboard for users to easily view information about the network in an easily understandable, digestible format. Although automatically detecting IoT devices on the network proves to be a challenging and complex task, this report investigates various approaches as to how to achieve this task. In addition, the design and implementation of the dashboard UI is also explored.

Figure 11: Settings Page

with using the dashboard, currently, this includes the GitHub repo for the dashboard code so they can post in the issues section, as well as my LinkedIn profile. This section would ideally include links to a more professional support email if the application was open to the public. Lastly, the about section should contain a short description of the dashboard, as well as the motivation behind its development, currently, the abstract of this report can be seen in the about section. Each section on this page uses a custom "Widget" component to serve as a housing for each setting. The Refresh Interval and Packet Capture Length sections both use Material UI Slider components with adjusted styling to fit this specific use case. The Change Packet Storage Location section uses an InputBase component to allow the user to input text and the Help and About sections simply use Typography components to display text.

4. Results

Currently, the frontend UI, the area in which the user will interact with, is complete. This section will explain future work that can be done to improve and complete the project, as well as outline the results of the dashboard implementation.

4.1 Future Improvements

As previously mentioned, due to complications in development only the dashboard UI is completed. In the future, functionality for automatic device detection, discovering information on the source/destination of packets, and pushing this information to the dashboard could be implemented.

Regarding device detection, the simplest approach would be to use an OUI lookup for each device to potentially get a general idea of what each device is, and force the user to manually select which devices they want to be monitored. For automatic IoT device detection, machine / deep learning would not only be the most accurate but also user-friendly as the user would not have to identify each device in the list (either by name or by matching the MAC addresses). Although, this solution is the most complex and would likely require extensive testing with various devices and networks.

Looking at gathering information on the source or destination of packets, a suggested implementation would be to utilize ARP spoofing similar to IoT Inspector Client. Although there are security implications to this approach, utilizing ARP spoofing would be the simplest approach to implement. By having packets intended for IoT devices pass through the computer running the dashboard before going on to their destination, layer 3 packets should be able to be captured, and therefore IP headers. Utilizing this method should allow for the use of the traceroute command which is key for seeing how a packet travelled to the user's network.

Once all desired information is gathered, pushing information to the dashboard should be relatively straightforward. There are various arrays (filled with objects) that need to be updated. Each object in these arrays is required to have a key titled “id”. Other than that mandatory key, all other keys in these objects can be up to the developer (as long as the corresponding pages are updated accordingly).

4.2 Dashboard UI Results

After realizing the development of this project could be picked up in the future by someone else, the design decisions in the way the dashboard UI was developed changed. In an attempt to make the code easier to read and debug, the languages, frameworks, and libraries used in the development of the UI were changed. I originally intended to use tailwind to speed up styling but despite it typically being faster, it tends to make the code less readable to those who are not familiar with it. Opting to use Material UI for various components and styling achieved the same development speed goals while maintaining code readability. As previously mentioned, the code was also designed to ensure anyone collaborating or taking over development would easily be able to alter the code to make the dashboard live. Because of this, all information to populate the components is housed in one file so all data can be updated in one place and then propagate to the rest of the dashboard.

In terms of the UI, the majority of elements I intended to include were implemented with mock data inserted to simulate the dashboard being live. One of the main goals of the UI was to ensure that it was intuitive and simple to use which I feel was achieved. Each page is clearly labelled and has a specific function, designed as such to not overwhelm the user with too much information.

5. Conclusion

As Internet of Things devices will undoubtedly continue to become more popular in the future, it remains clear that more thought and attention need to be placed on how to better protect local and enterprise networks (LANs & EPNs) with the introduction of these devices. Due to the low importance placed on security when developing these devices, manufacturers have inadvertently introduced network security liabilities in countless networks around the world. It is clear from the findings of this project that there are various approaches to identifying and monitoring these devices. Real-world implementations of automatic IoT device detection and packet analysis proved to be more complex than originally anticipated, however, the various topics covered throughout this report are a good place to start in tackling this problem. Even with the omission of these features, the network dashboard remains a beneficial user interface that can aid in the future addition of automatic device detection and packet analysis of Internet of Things devices.

6. References

[1] *How to find your device mac address - computing services - office of the CIO - Carnegie Mellon University*. How to Find Your Device MAC Address - Computing Services - Office of the CIO - Carnegie Mellon University. (n.d.). Retrieved December 16, 2022, from https://www.cmu.edu/computing/services/endpoint/network-access/mac-address.html#:~:text=A%20MAC%20(Media%20Access%20Control,%2D63%2DC2%2D26.

[2] *What's the difference between external and local IP addresses?* H3XED. (n.d.). Retrieved December 16, 2022, from https://www.h3xed.com/web-and-internet/whats-the-difference-between-external-and-local-ip-addresses#:~:text=An%20external%20or%20public%20IP,and%20devices%20connected%20to%20it.

[3] Agarwal, S., Oser, P., & Lueders, S. (2019). Detecting IoT Devices and How They Put Large Heterogeneous Networks at Security Risk. *Sensors (Basel, Switzerland)*, *19*(19), 4107. https://doi.org/10.3390/s19194107

[4] HUANG, DANNY YUXING ,.APTHORPE, NOAH,. LI, FRANK,. ACAR, GUNES, & FEAMSTER, NICK. (n.d.). *IOT Inspector: Crowdsourcing labeled network traffic from Smart Home ...* Retrieved December 16, 2022, from https://iotinspector.org/papers/ubicomp-20.pdf

[5] Y. Liu, J. Wang, J. Li, S. Niu and H. Song, "Machine Learning for the Detection and Identification of Internet of Things Devices: A Survey," in IEEE Internet of Things Journal, vol. 9, no. 1, pp. 298-320, 1 Jan.1, 2022, doi: 10.1109/JIOT.2021.3099028.

[6] A. Sivanathan, "Iot behavioral monitoring via network traffic analysis," arXiv preprint arXiv:2001.10632, 2020.

[7] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. 2017. ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. In Proceedings of the Symposium on Applied Computing (SAC '17). Association for Computing Machinery, New York, NY, USA, 506–509. https://doi.org/10.1145/3019612.3019878

[8] A. Sivanathan et al., "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," in IEEE Transactions on Mobile Computing, vol. 18, no. 8, pp. 1745-1759, 1 Aug. 2019, doi: 10.1109/TMC.2018.2866249.

[9] Kotak, Jaidip & Elovici, Yuval. (2020). IoT Device Identification Using Deep Learning. https://www.researchgate.net/publication/339526701_IoT_Device_Identification_Using_Deep_Learning

[10] Patel, H. (2021, September 2). *What is feature engineering - importance, tools and techniques for machine learning*. Medium. Retrieved December 16, 2022, from https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10

[11] https://wiki.wireshark.org/CaptureSetup/WLAN#packet-types

[12] *Complaint regarding terms of use and privacy policies for smartwatches / GPS watches for children*. (n.d.). Retrieved December 16, 2022, from https:fil.forbrukerradet.no/wp-content/uploads/2017/10/20171013-klage-smartklokker-dt-fo-eng.pdf

[13] https://github.com/nyu-mlab/iot-inspector-client/issues?page=1&q=is%3Aissue+is%3Aopen

[14] CAPANO , D. A. N. I. E. L. E. (2022, November 3). *Wi-Fi and the OSI model*. Control Engineering. Retrieved December 16, 2022, from https://www.controleng.com/articles/wi-fi-and-the-osi-model/

[15] Roy Chowdhury, R., Aneja, S., Aneja, N., & Abas, P. E. (2021). Packet-level and IEEE 802.11 MAC frame-level network traffic traces data of the D-Link IoT devices. *Data in brief*, *37*, 107208. https://doi.org/10.1016/j.dib.2021.107208

[16] https://networkengineering.stackexchange.com/questions/56750/is-it-possible-to-perform-a-layer-2-mac-address-traceroute

[17] Andrea, H. (2022, January 21). *Comparison of IP Layer 3 packet vs Layer 2 frame in networking*. Networks Training. Retrieved December 16, 2022, from https://www.networkstraining.com/comparison-of-packet-vs-frame/