MACTracker: Data Collection and Analytics

David Bedoya

Senior Capstone Project CSC400

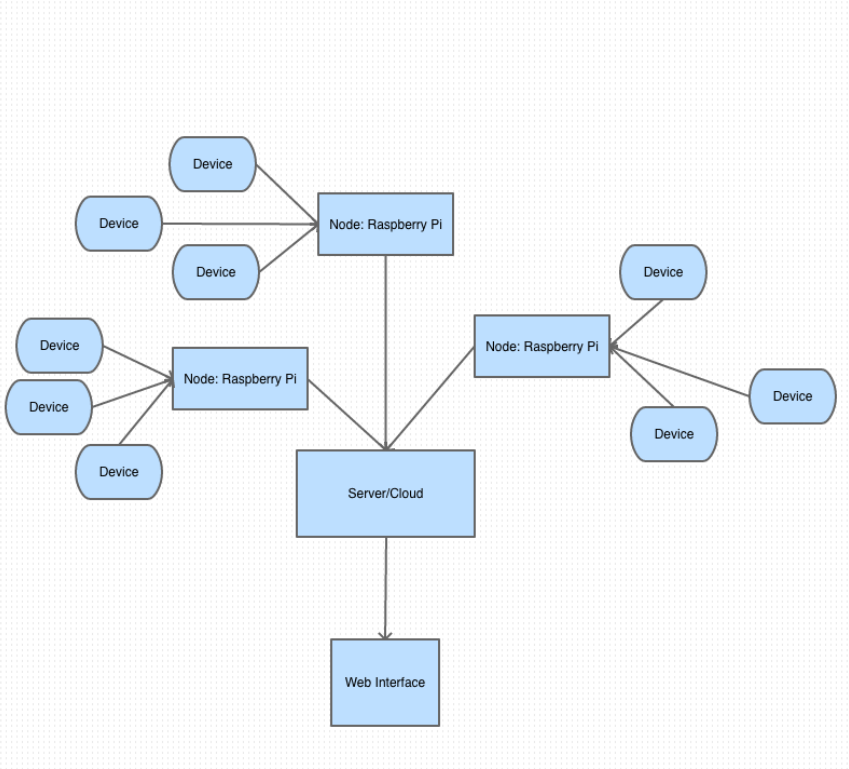
GitHub Repository:
https://github.com/hoy89/MACTracker-CSC400-Final.git

Introduction

This tool serves as an alternative method of determining foot traffic for businesses or individuals, and to serve as a way to collect metrics. Instead of relying on indirect methods of obtaining traffic data, such as sales, this tool will provide a way to have a better understanding of the actual number of people in a specified area. The advantage that this brings is that it can also measure persons who do not interact directly with the services being provided by the business. For example, a business may only have 35% of the people entering its venue make a purchase – but the sales figures only show metrics for that 35%. A business can use this tool to increase the level of interaction with the products and services that it offers. The most important feature that this system offers is data collection and visual representation of that data.

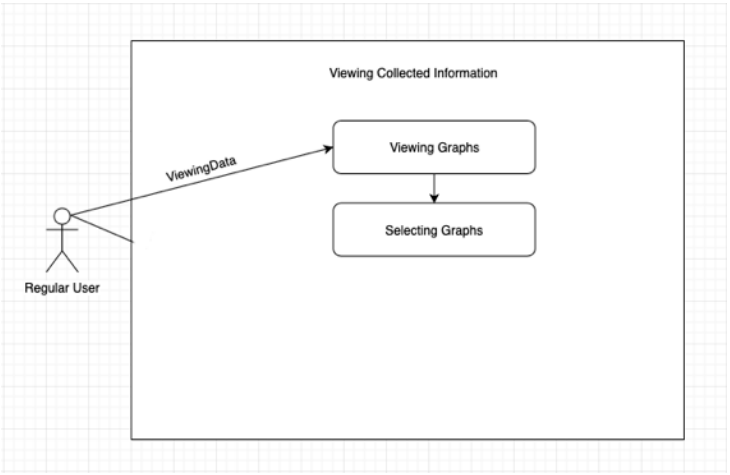
Architecture

The type of system used is a server + sensors configuration. That means that there are sensors (Raspberry Pi's) that the server connects to and retrieves the information from. The server can either be hosted locally, or in the cloud, depending on the specific requirements of the deployment. The only task that the sensors must do is collect data in real time, and then the server will fetch the data through an SFTP connection. At the server level, the data will then be processed as soon as it is detected, in real time. The web management page will then process the metrics that are available, and display the data at the time of opening it. Ideally, the system will only be accessed and configured through a web management interface for all major functions past the initial setup. The main inputs are from the sensors (nodes), which are MAC addresses in range of the data collector. The main output is the processed data that has been collected.


Device
Device
Device
Node: Raspberry Pi
Device
Device
Node: Raspberry Pi
Device
Device
Device
Node: Raspberry Pi
Device
Device
Server/Cloud
Web Interface

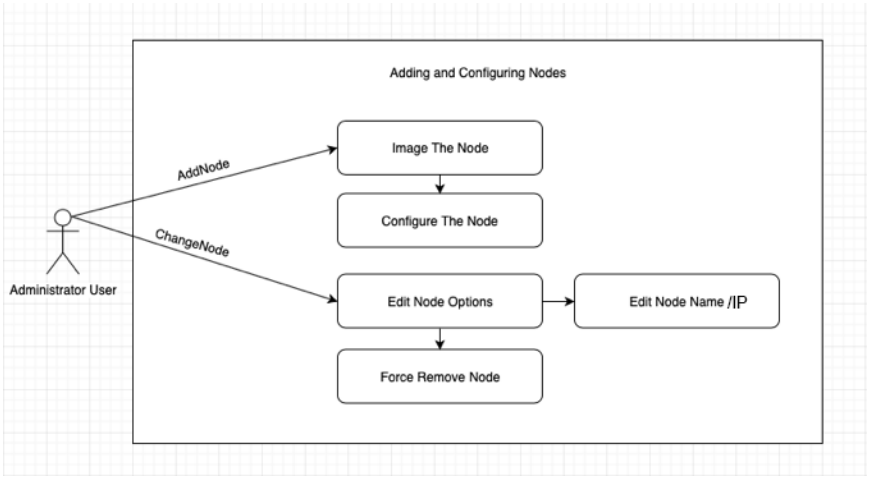
Use Cases

Use Case 1 – Viewing collected information



1.1 - The user navigates to the web management home screen. Here, all graphs available will be visible. There is a general section where an aggregate of the information collected is visible, as well as a section to view a breakdown of each area and nodes within that area.

Use Case 2 – Adding and configuring a tracking “Node”



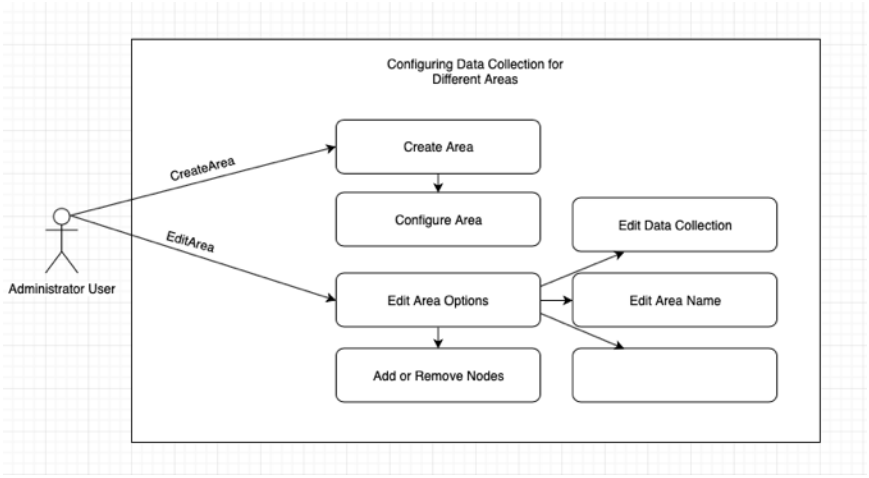
2.1 - An administrator user wishes to add another tracking node. First, the node itself must be configured before it can be added. The user must first image the node with the base image provided. The configuration file is then updated with the proper Wi-Fi network connection information that will be used in the physical location where the node is placed, as well as assigning a name and IP address. The node name and IP address must then be added in the “Node Configuration” page.

2.2 - A user wishes to view the identities of all nodes. The user clicks on the "Node Configuration", then clicks on "Select a Node". From here, it is possible to view the information on each of the nodes available. The name of the node, as well as the IP address of the node is visible in this section

2.3 - To be Implemented: A user wishes to change the identities of al node. The user clicks on the "Node Configuration", then clicks on "Select a Node". From here, it is possible to view the information on each of the nodes available. The name of the node, as well as the IP address of the node are configurable in this section.

2.3 - To be Implemented: It may be desired that a node be removed. The user clicks on the main menu, then clicks on the "Node Configuration" selection. The user then clicks on the name of the node, and once on the page for that node, can select "Remove Node". The server will then remove the node from the list of recognized nodes. The user will then have to manually reconfigure the node.

Use Case 3 – Configuring data collection for different areas

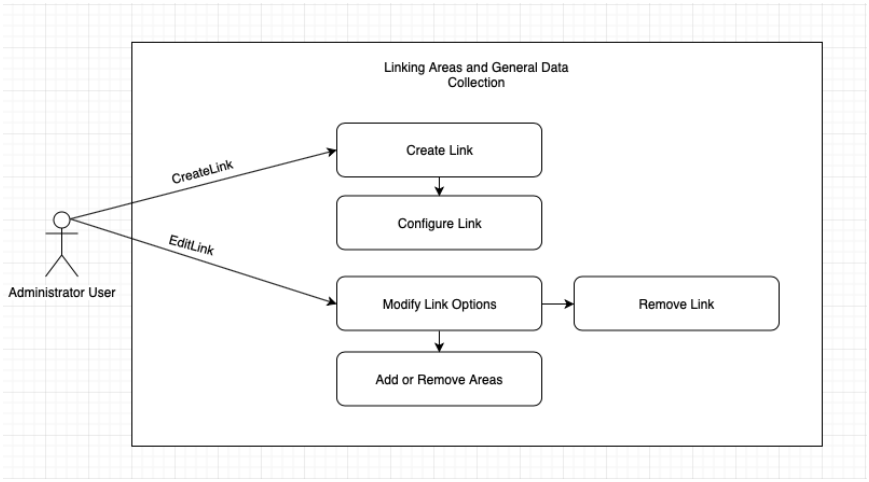


3.1 - By default all nodes will have their data collection merged together and represented as one big area. A user decides that different nodes physically located in different areas should have their data represented separately from each other. The user clicks on the main menu, then clicks on the "Area Settings" selection, then on "Create Area". The user is then prompted to enter a name for the area, and can select any node. If a node is already assigned, it will be updated to be under the new area. Once the entry is saved, the main management page will be updated to show statistics for the new area.

3.2 - A user would like to remove or edit an area's name or nodes. The user clicks on the main menu, then clicks on the "Area Settings" selection, then on the area they would like to edit. There is the option to edit the name of the area, add or remove nodes, or to delete the area. Once the changes are saved, if the area is deleted or nodes are removed, the nodes will be moved back into the default area. The changes will then be reflected on the main management page.

3.3 - To Be Implemented: Beyond the default data collection (Number of people detected over an interval of time); the user can configure other data collections for an area. The user selects the main menu, clicks on "Area Settings", then on the area they would like to edit. The user can then select different types of data collection (e.g. Average length of time a person is detected in that area), and save their selection. The changes will then be reflected on the main management page.

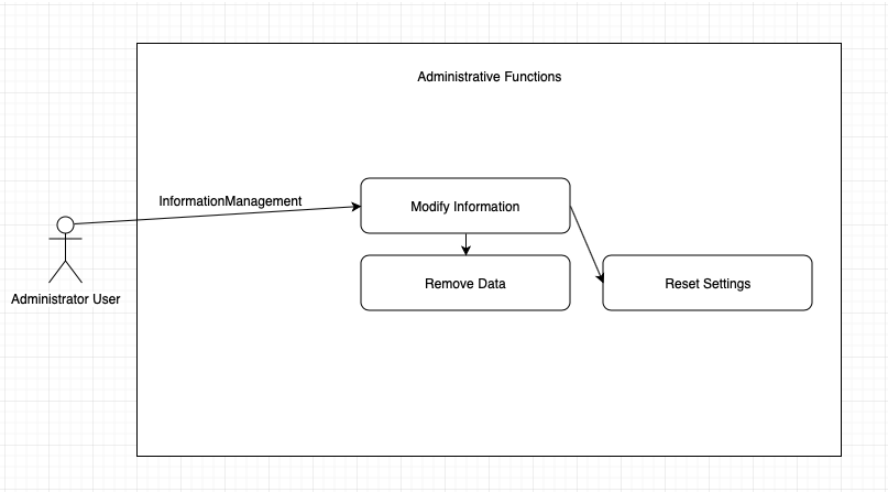
Use Case 4 – To Be Implemented: Linking areas



4.1 - To Be Implemented: A user may wish to compare different areas. They may do so by "linking" two or more areas. The user navigates to the main menu and selects "Area Linking". From here, the user selects "New Link" and can choose from a list of unlinked areas. By default, only the common statistics that both areas are already collecting will be compared. The user may also have the option to select additional statistics such as percentage of people found in one area that are also found in the other area. Once the link is saved, there will be a new option on the main management page to show the new statistics.

4.2 - To Be Implemented: A user may also wish to "un-link" areas, or modify a link. They may do so by navigating to the main menu, selecting "Area Linking", then "Edit Link". Then the user may select the link they wish to modify, and can select either "Add or Remove Areas", or "Remove Link". If a user selects "Add or Remove Areas", they will then have the option to add an available area or remove an existing area. If the user selects "Remove Link", the link will be removed. Once the changes are saved, the changes will then be reflected on the main management page.

Use Case 5 – Administrative Functions

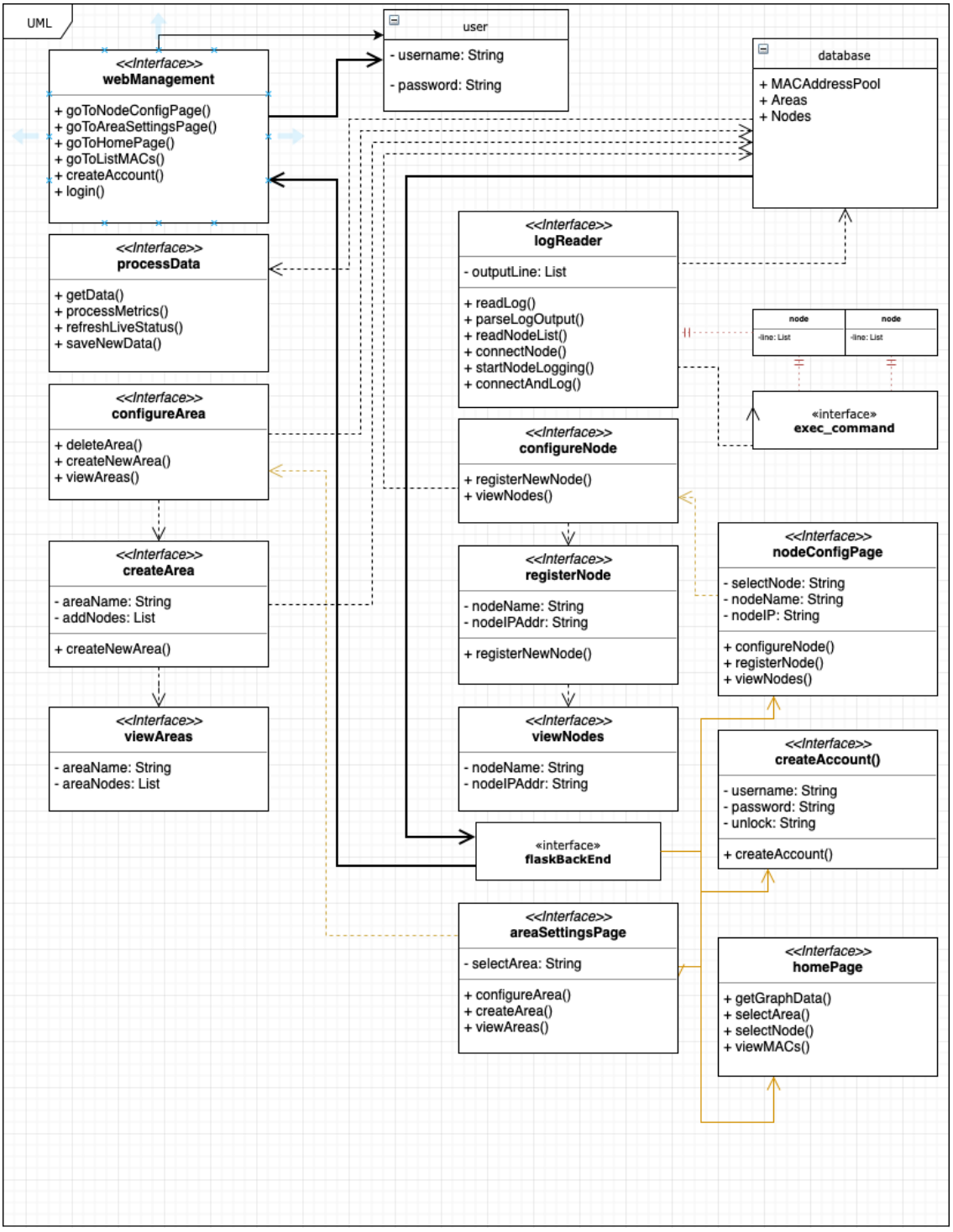


5.1 - To Be Implemented: Deleting or reset collected data. The administrator account has the ability to delete old data, or completely reset all data collected. The user clicks on the main menu, clicks on "Settings", "Delete Data". They then have the option to either click "Delete All Data". Once the option is selected, the user will be prompted with a "Are you sure you want to continue?" warning, then once accepting the risk, the data will be deleted permanently.

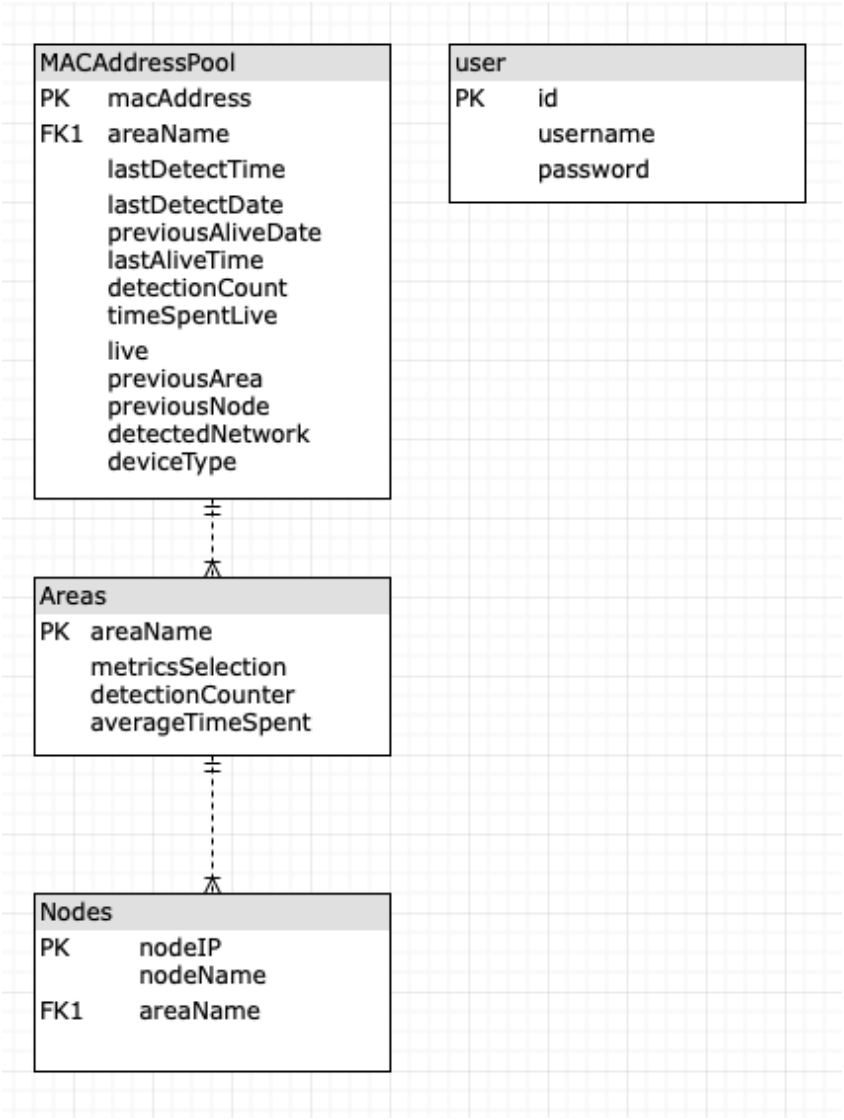
5.2 - To Be Implemented: Resetting settings to default. If desired, all settings can be reset to the default state without deleting any information. The user clicks on the main menu, clicks on "Settings", "Reset Settings to Default". Once the option is selected, the user will be prompted with a "Are you sure you want to continue?" warning, then once accepting the risk, the settings will be returned to default.

5.3 - Creating a new user. A new user may be created at any time, as long as the unlock key is known. The user clicks on register at the top, while signed out, then enters a unique username, password, as well as the unlock key. The user can then sign in.

Structural Design

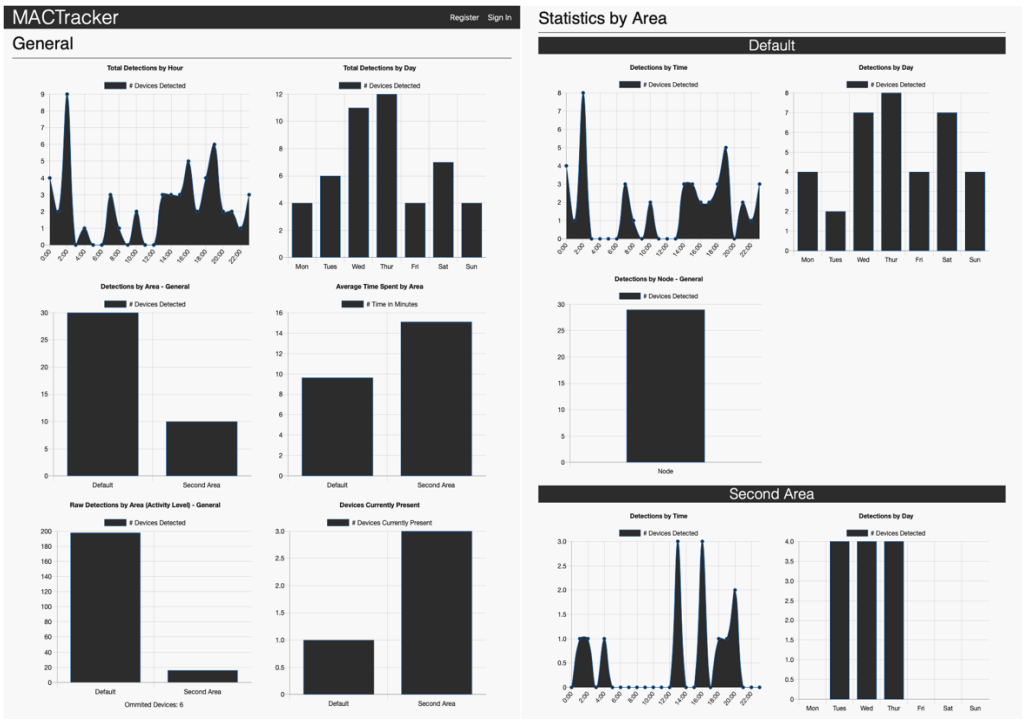


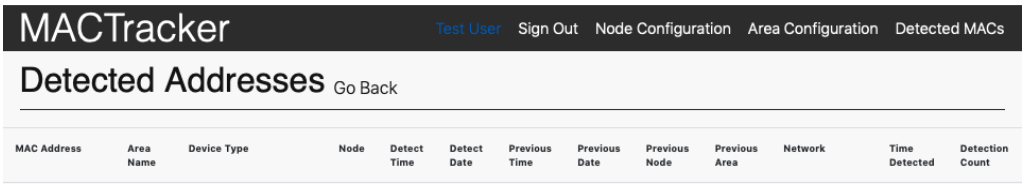
Data


MACAddressPool
PK macAddress
FK1 areaName
lastDetectTime
lastDetectDate
previousAliveDate
lastAliveTime
detectionCount
timeSpentLive
live
previousArea
previousNode
detectedNetwork
deviceType
user
PK id
username
password
Areas
PK areaName
metricsSelection
detectionCounter
averageTimeSpent
Nodes
PK nodeIP
nodeName
FK1 areaName

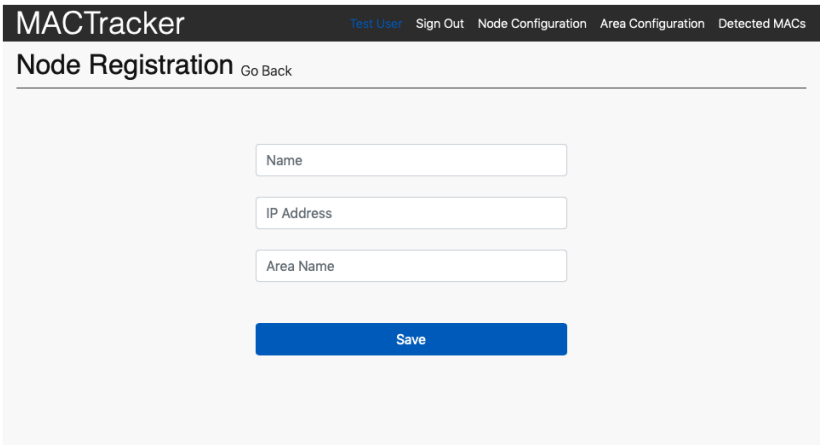
User Interface Design

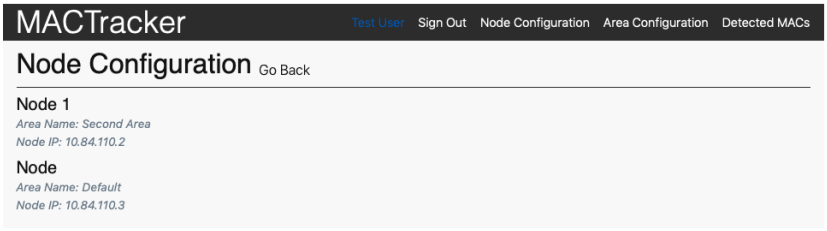
Viewing Information

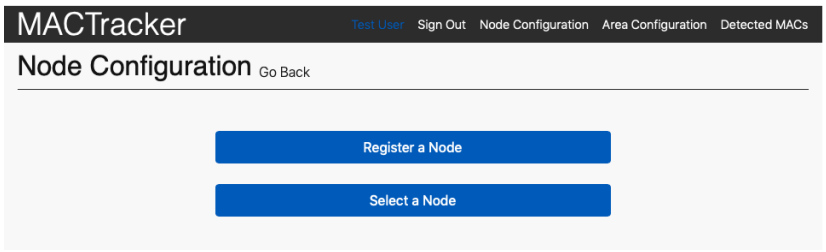




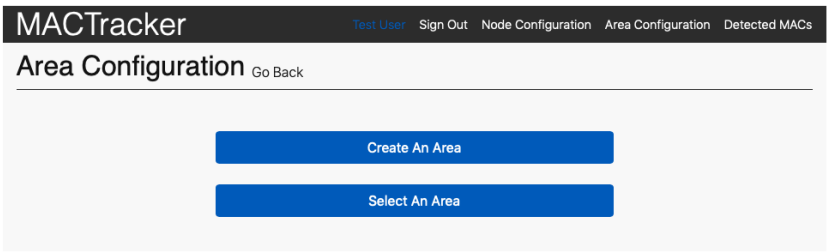
Configuring Nodes

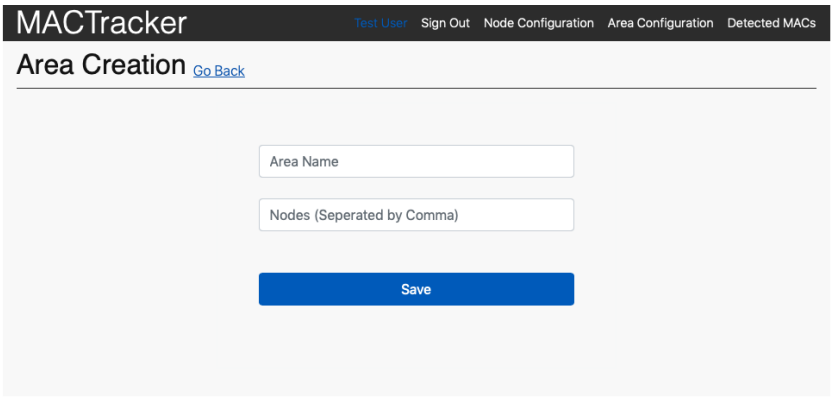


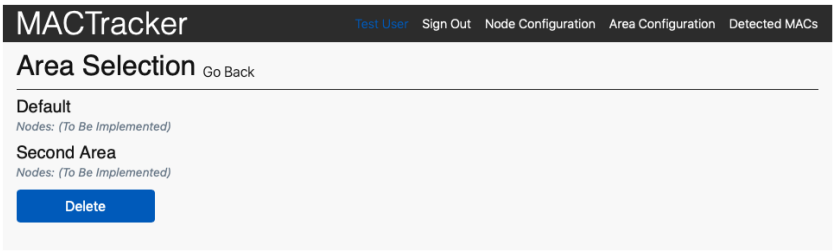




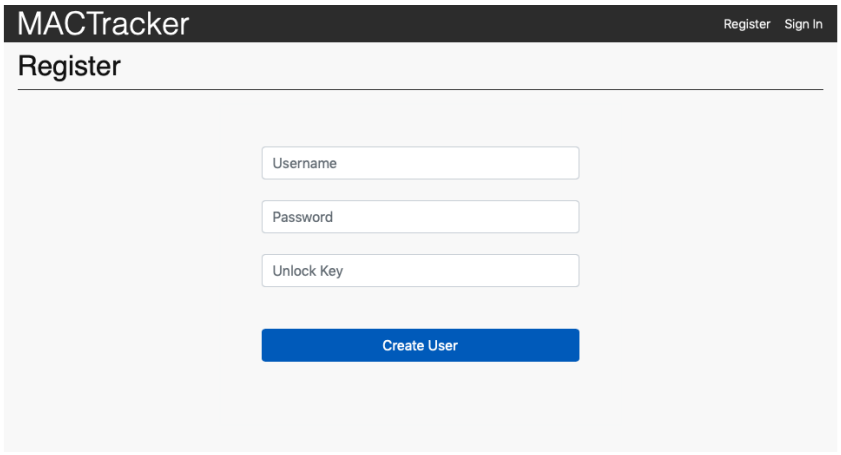
Configuring Areas

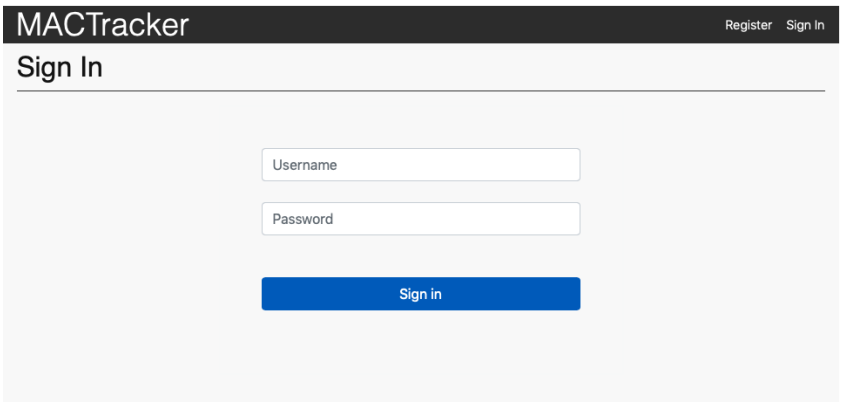






Administrative Functions





Documentation of the Implementation

The base of this project is flask, which controls all of the major functions. Originally, the core of the project was a separate Python program which controlled and stored the information gathered by the Raspberry Pi's. This has been integrated into flask, where flask runs this program and also collects the output from it. Outside of this main application, is the Python program Probemon run on the Raspberry Pi's that ties together the hardware, and data collection, aspect with logging – modified to suit the project. Probemon is a small program that uses Scapy to process the packets that come from the USB wifi dongle set to monitoring mode with Aircrack.

To read and process the resulting log file generated by Probemon, a separate (the original, logreader.py) Python program is used. First, it must read the list of nodes configured in the web application from the database. From here, it starts a new thread for each node as it must be constantly running. The sequence is as follows:

1.) There is a connection established over SSH and an SFTP connection is made as well

```
def connectNode(client_name, user, passw):
    print("Connecting...")
    connectAttempts = 3
    errorFlag = 0
    sftpClient = None
    sshClient = None

    while connectAttempts > 0:
        try:
            connectAttempts = connectAttempts - 1
            sshClient = paramiko.SSHClient()
            sshClient.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            sshClient.connect(client_name, username=user, password=passw)
            sftpClient = sshClient.open_sftp()
            return sftpClient, sshClient, errorFlag
        except:
            print("Connection failed. Trying again...")

    errorFlag = 1
    return sftpClient, sshClient, errorFlag
```

2.) Commands are sent remotely to start the logging process

```
def startNodeLogging(sshClient):
    print("Starting node logging...")
    stdin_, stdout_, stderr_ = sshClient.exec_command("sudo pkill -9 -f probemon.py") #kill any previous script first
    stdin_, stdout_, stderr_ = sshClient.exec_command("sudo rm /home/pi/probemon.log") #remove old log before starting up logging
    stdin_, stdout_, stderr_ = sshClient.exec_command("sudo touch /home/pi/probemon.log") #create new log file
    stdin_, stdout_, stderr_ = sshClient.exec_command("sudo airmon-ng start wlan1") #enable monitor mode on wlan1
    stdout_.channel.recv_exit_status() #wait until previous command finishes
    stdin_, stdout_, stderr_ = sshClient.exec_command("sudo python /home/pi/python/probemon/probemon.py -i wlan1mon -f -s") #start monitoring
    time.sleep(5)
```

3.) The resulting log is then read line by line and the output is written to the database

```
def readLog(sftpClient, nodeName, areaName, liveInterval, app):
    print("Logging started on " + nodeName)
    remoteFile = sftpClient.open('/home/pi/probemon.log')
    while True:
        where = remoteFile.tell()
        line = remoteFile.readline()
        if not line:
            time.sleep(0.100)
            remoteFile.seek(where)
        else:
            line = parseLogOutput(line)
            writeDatabase(line, nodeName, areaName, liveInterval, app)
```

3.b) Before writing to the database, the data is processed (excerpt)

```
# misc
detectionCount = detectionCount + 1

# cosmetics
if detectedNetwork == '':
    detectedNetwork = None

if live == False:
    # update previous values
    previousAliveDate = lastDetectDateOld
    lastAliveTime = lastDetectTimeOld
    previousArea = newPreviousArea
    previousNode = newPreviousNode

if live == True:
    if lastAliveTime is None:
        timeSpentLive = liveInterval
    else:
        currentTime = line[3]
        previousTime = lastAliveTime

        currentTime = int(currentTime[-2:])
        previousTime = int(previousTime[-2:])
        timeSpentLive = abs(currentTime - previousTime)
```

In addition to this constantly running program, there is another program (Scheduler.py) that is also run separately that is continually keeping track of time that a MAC address is detected.

```
    def updateLive(app, liveInterval, refreshSpeed):
        with app.app_context():
            db = get_db()
            print("Starting live status update.")
            while True:
                macAddressPool = db.execute(
                    'SELECT macAddress, lastDetectTime, live, lastAliveTime, timeSpentLive FROM macAddressPool'
                ).fetchall()

                for macAddress in macAddressPool:
                    mac = macAddress[0]
                    live = macAddress[2]

                    if live == True:
                        currentTime = macAddress[1]
                        currentTime = (int(currentTime[:2]) * 60) + int(currentTime[-2:])

                        # Get current system time in minutes starting from 0
                        date = dateutil.parser.parse(datetime.datetime.now().isoformat())
                        systemTime = str(date.time())[0:5]
                        systemTime = (int(systemTime[:2]) * 60) + int(systemTime[-2:])

                        if abs((currentTime - systemTime)) > liveInterval:
                            db.execute('UPDATE macAddressPool SET live=? WHERE macAddress=?', [False, mac])
                db.commit()
                time.sleep(refreshSpeed)
```

The rest of the application is in the main flask Python script, entitled “Main.py”. This application loads each function as needed when the appropriate URL is loaded. For example, the index page will automatically generate all of the statistics that are being tracked, from the information stored earlier in the database. These statistics are stored in lists that are read with Jinja and then ChartJS/Javascript creates a graph that is then arranged within the HTML.

An example of one statistic, devices detected by hour and by day:

```
for macAddress in macAddressPool:
    # get times for all mac addresses

    currentTime = macAddress[4]
    currentTime = int(currentTime[:2])
    timebreakdown[currentTime] = timebreakdown[currentTime] + 1
    if macAddress[7] != None:
        previousTime = macAddress[7]
        previousTime = int(previousTime[:2])
        timebreakdown[previousTime] = timebreakdown[previousTime] + 1

    # get dates for all mac addresses
    currentDate = macAddress[5]
    year, month, day = (int(x) for x in currentDate.split('-'))
    result = datetime.date(year, month, day)
    result = result.strftime("%A")
    if result == 'Monday':
        datebreakdown[0] = datebreakdown[0] + 1
    elif result == 'Tuesday':
        datebreakdown[1] = datebreakdown[1] + 1
    elif result == 'Wednesday':
        datebreakdown[2] = datebreakdown[2] + 1
    elif result == 'Thursday':
        datebreakdown[3] = datebreakdown[3] + 1
    elif result == 'Friday':
        datebreakdown[4] = datebreakdown[4] + 1
    elif result == 'Saturday':
        datebreakdown[5] = datebreakdown[5] + 1
    elif result == 'Sunday':
        datebreakdown[6] = datebreakdown[6] + 1

    if (macAddress[6] != None) and (macAddress[6] != macAddress[5]):
        currentDate = macAddress[5]
        year, month, day = (int(x) for x in currentDate.split('-'))
        result = datetime.date(year, month, day)
        result = result.strftime("%A")
        if result == 'Monday':
            datebreakdown[0] = datebreakdown[0] + 1
        elif result == 'Tuesday':
            datebreakdown[1] = datebreakdown[1] + 1
        elif result == 'Wednesday':
            datebreakdown[2] = datebreakdown[2] + 1
        elif result == 'Thursday':
            datebreakdown[3] = datebreakdown[3] + 1
        elif result == 'Friday':
            datebreakdown[4] = datebreakdown[4] + 1
        elif result == 'Saturday':
            datebreakdown[5] = datebreakdown[5] + 1
        elif result == 'Sunday':
            datebreakdown[6] = datebreakdown[6] + 1
```

An example of one graph:

```
  <div align="center" style="display: inline-block; padding-left: 3.5%">
    <canvas id="totalByDay" width="400" height="400"></canvas>
    <script>
      var ctx = document.getElementById('totalByDay').getContext('2d');
      var totalByDay = new Chart(ctx, {
          type: 'bar',
          data: {
              labels: ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun'],
              datasets: [{
                  label: '# Devices Detected',
                  data: {{ datebreakdown|safe }},
                  backgroundColor: '#323232',
                  borderColor: '#0065c2',
                  borderWidth: 1
              }]
          },
          options: {
              responsive: false,
              title: {
                display: true,
                text: 'Total Detections by Day'
              },
              scales: {
                  yAxes: [{
                      ticks: {
                          beginAtZero: true
                      }
                  }]
              }
          }
      });
    </script>
  </div>
```

Other functions of the application include a rudimentary user system that has been sourced from the flask documentation, but modified to include a basic verification function. IE, an addition ‘master’ password. Functions such as deleting an area will propagate the changes to all of the nodes that are listed for it, and replace the area that they are listed under as ‘Default’. Conversely, creating an area will propagate the changes to all of the listed nodes to reflect the new area. Changes made here will affect the information shown on the main page, and all of the data collected will then be processed to reflect the changes as well.

Shown is the code that deletes an area:

```
@bp.route('/selectarea', methods=('GET', 'POST'))
@login_required
def select_area():
    db = get_db()
    areas = db.execute(
        'SELECT areaName FROM areas'
    ).fetchall()

    if request.method == 'POST':
        if request.form['delete'] == "Delete":
            areaName = request.form['areaName']
            areaNameDefault = 'Default'

            nodeList = db.execute(
                'SELECT nodeName FROM nodes WHERE areaName=?',
                [areaName,]).fetchall()

            db.execute(
                'DELETE FROM areas WHERE areaName=?',
                 [areaName])

            for node in nodeList:
                    db.execute(
                        'UPDATE nodes SET areaName=? WHERE nodeName=?',
                         [areaNameDefault, node[0]])
                    db.execute(
                        'UPDATE macAddressPool SET areaName=? WHERE nodeName=?',
                         [areaNameDefault, node[0]])
            db.commit()
            return render_template('main/areaconfig.html', areas=areas)

    return render_template('main/selectarea.html', areas=areas)
```

Registering nodes use a similar process as the above. Database is read, then written to.

Installation

Raspberry Pi Portion:

1.) Flash the image onto an SD card 8GB or larger

2.) Edit /etc/dhcpcd.conf under section “Example static IP configuration” in order to set a static IP address for the Pi.

3.) Done. Pi can be accessed at pi@[configured IP] with password ‘password’, using SSH

Web Application Portion:

1.) Navigate to folder containing flask application (cd ~/Location/MACTracker)

2.) Run these commands:

```
python3 -m venv venv
. venv/bin/activate
pip install -e .
export FLASK_APP=flaskr
export FLASK_ENV=development
export FLASK_DEBUG=0
pip install paramiko
pip install python-dateutil
flask init-db
flask run --host [IP address]
```

NOTE: Log Reader and Scheduler run after executing init-db. Kill the process with control + C, it is safe

3.) Create new log in, default Unlock Key is ‘password’

4.) Register a node

5.) Done. Logging will start immediately

State of the Implementation

Features:

- Logging MAC addresses (Raspberry Pi portion): Done, no further development needs to be done on the Pi itself

- Parsing MAC address logs: Done, all of the relevant information is able to be extracted from the logs in real time. Further processing can be done with this information collected

- Data Metrics: 60-75% complete. Some change needs to be made in order to separate the data from the actual MAC address entries in the data base, to prevent unwanted changes from occurring as MAC address entries are updated. This feature is definitely the biggest function of the application and further polish and added metrics will always need to be improved upon

- User Log In: Done. Users can log in without issues

- User Creation: Incomplete. Stop gap 'unlock key' is used in lieu of an actual user rights system

- Node (Raspberry Pi) Registration: Done. Registered nodes are automatically picked up and connected to

- Node Editing and Deletion: Not Done. Nodes cannot be removed or changed in the database because the Log Reader application does not support that functionality without needing to be restarted manually

- Area Creation: Done. Areas that are created are automatically reflected in the web application fully, and nodes that are added to it are also updated as well

- Area Editing and Deletion: Partially Complete. Areas that are deleted have changes reflected without any issue. No functionality for editing areas directly has been implemented

- Viewing Raw Data: Done. A table with all information collected is generated when visiting the "Detected MACs" page

- Chart Generation from Data: Done. Charts are created that automatically reflect the latest data collected when refreshing the page. Caveat: in order to create new graphs, they must be manually coded in.

Testing and Evaluation

To test the implementation, several regular devices were used to determine how well the system picked up MAC addresses and processed them. The following methods gave positive result and usable data:

- Leaving Android devices in standby for tens of minutes at a time
- Bringing any phone within range of a 2.4GHz network that it automatically connects to
- Using the device, i.e., not in standby
- Entering the WIFI network selection screen

Further testing needs to be done in a location that is dense with WIFI enabled devices and with several more nodes, to test the system's responsiveness and accuracy when presented with substantial load. Other testing done included how well the system handled unexpected errors, such as nodes not existing or disconnecting, improper input in the web application, etc. Overall, the application does seem to work mostly reliably with the most difficult portion of it is the initial setup due to needing to know exactly how the network it is deployed on is set up. It does meet most of the objectives that I set out to complete, but the biggest hurdle that I have not been able to overcome is the randomization of MAC address problem that iPhones have. Almost every other device can be tracked with a certain amount of reliability (depends on the brand of phone – Samsung phones are very reliable; LG is least but still much better than Apple)

Lessons Learned and Reflection

The part of the project that went fairly smoothly was the initial logging of MAC addresses. Of course, that is to be expected because of the use of pre-made resources. However, the part that was the most difficult by far was making sense of the data collected. Simply logging MAC addresses is neat, but not very useful by itself. It does not give any meaningful information presented as is, without further processing. Initially, I thought that the most difficult portion of the project was collecting the addresses and that the collection and display of said information would be a metaphorical 'walk in the park'. This turned out to be the complete opposite, and it took a significant amount of time to redesign my initial concept (that I had imagined) into real code. So much so, that other smaller features needed to be dropped or not polished. If I were to do the project all over again, what I should have done was realized that when I found that it was not an incredibly demanding task to simply collect MAC addresses, like I had originally imagined, was to immediately switch focus to data analytics as that is where the real value was. To put in perspective, the collection of addresses only took a couple of weeks whereas I believed that it would take at least 1-2 months, meanwhile the remaining portion of the application absorbed the rest of the time that I had to work on the project.

Version 2

With an extra couple of months, the biggest feature I would want to implement is a rudimentary estimation of phones that are using MAC address randomization. Realizing that this would be a separate research project on its own, the goal instead would be to give an additional metric with a certain level of confidence (say 40-60%) on an estimation of the amount of these types of devices present. Other than that, if there was enough time, I would also implement other useful metrics such as directly comparing two different areas or a proper user account system, etc. Beyond features, general polish and code cleanup would also be done.

Professional Development and Lifelong Learning

The process of developing skills is always linked directly towards the actual practice of the skill that one would like to develop. For example, if one would like to learn front end development, the only way to properly learn front end development is to develop front end. Learning and becoming proficient in a skill is a continuous process of 'adding a little bit more' to what one currently is knowledgeable in. In this project, the biggest skill that I have begun to learn, is the integration of many different components to build a singular system. This project is an amalgamation of many smaller components, such as the use of Linux based Raspberry Pi's that intercept packets, that communicate with SSH/SFTP, that are handled by a multithreaded application, that communicates with a database, etc. The important take-away from a project like this is to gain a better understanding of the bigger picture and how the bits and pieces that have been learned previously, actually matter. Specifically, in this project, resources that I have personally used include sites like forums, Stack Exchange, GitHub, and various different sites and articles to pull relevant pieces of information for each small portion of the project. Additionally, the documentation for different libraries, such as the paramiko library (used for using SSH/SFTP within python), has been invaluable as in many cases there is not too much discussion in how exactly to use those libraries. Other tools such as the default Raspberry Pi Linux OS, while definitely useful, have ended up being rather annoying to work with as it is oriented towards 'ease of use' and 'easy to learn' while not really conforming completely to the standard Linux environment. This meant that I had to go out of my way to learn the little niches of Raspbian rather than be ready to go right away, and installing a familiar Linux distribution was not an option as they have to be specifically built for the Raspberry Pi. Situations like these showcase the importance of not relying solely on tools that are designed for ease of use wherever possible for one's own sake, as it masks a better understanding of the underlying system. However, the same could also be said about flask, which does do a fair amount of hand holding, but that would be outside of the scope of this kind of assignment. This project helped give a better perspective of the level of knowledge and work required to create a polished finished product, and while it is unrealistic to assume that one should be able to create a working final product, start to finish and ready to sell right off the bat, it does show the level of skill needed to be successful in a real environment. Overall, I think that this assignment is a good stepping stone into real world development that actually matters, and to continue to work on the skills learned here until they can be considered competent and competitive, and further beyond that.