Automatic Raspberry Pi Network Configuration and Discovery

Gamble, Zach McDonnell, Brendan Radley, Ian

for Dr. Phillip Romig III

June 16, 2021

Computer Science Department

Contents

List of Figures

Introduction

Python-based Computing: Building a Sensor System (CSCI250) is a class that uses Raspberry Pi's (RPi), a small computer running Linux, in order to learn about interfacing with electronic sensors. Each student purchases a Raspberry Pi to use throughout the semester. The client, Phil Romig, is an instructor for CSCI250 and is in need of a new method for effectively utilizing the RPis. Presently, the RPi must be connected to a monitor, keyboard, and mouse at the start of class, taking valuable class time and requiring a classroom with HDMI monitors — of which there is only one on campus large enough to accommodate a full class.

A preferable solution would be a headless setup: one that does not require the RPi to be connected to a monitor, keyboard, or mouse. A headless setup allows students to remotely access and control the RPi through SSH or VNC from a separate computer. There are two problems preventing a completely headless setup, however. First, the RPis would need to be initially setup to automatically connect to the network, including the Mines network where they need to register their MAC address with the school; and second, the students need some way of determining the IP address of their RPi to connect to it.

The Problem: Automatic Network Connection

The RPi needs to be able to connect to any wireless network the student needs to access it on. This is not currently possible. One issue is that the Mines wireless network requires every device (identified by MAC address) that accesses it to be registered to an individual. Until this registration takes place, the device cannot use the network. The other issue is that the RPi must be manually configured with the name and credentials of a network before it can automatically connect to it. At the moment, there is not method to perform these actions with a headless setup.

The Problem: IP Address Discovery

There is not a simple and reliable way for a student to determine the IP address of a headless RPi, which is necessary for connecting remotely. Presently, the RPi sends its IP address to the student in an email on boot. The email uses an insecure protocol the Colorado School

of Mines (Mines) is no longer permitting in future semesters. This method is unreliable, limited, and will not continue to function.

The goal for this product is to solve both of these issues to enable a completely headless RPi setup. To this end, the product has the following functional and non-functional requirements.

Requirements

Functional Requirements

- Initial Configuration
  - A method for a RPi to automatically connect to CSMwireless (Mines' network).
  - The student must also be able to register the MAC address of the RPi with the school so they have internet access on CSMwireless.
  - Provide instruction and functionality for students to add additional networks for automatic connectivity.
- RPi Information Discovery
  - Provide students with IP address of their RPi and network information (i.e network SSID).
  - Central server with Mines' authentication methods is preferred.
  - All communication must use robust, secure protocols.
  - Additional device information, such as the status of SSH and VNC services on the RPi, should also be included.

Non-functional Requirements

- Has to be a secure implementation such that no student's private information can be identified or stolen.
- Must be effective for students with no prior coding experience.
- Minimal steps to effectively operate.
- No complex command usage.
- Documentation must be readable, explicit, and understandable.
- Any central server system must be able to be easily moved to different server setups without loss of functionality or existing data.
- Methods, research, and end product must be well documented.
  - Added functionality must be well documented for future development or non-standard use cases.

  - End user documentation must be readable, explicit, and understandable for instructors to use in class. Documentation must be provided for student users and for instructors.
  - Methods used and concepts found along the way and any significant technical challenges should be described.

- Shibboleth, a third-party multifactor-secured system used by Mines, provides authentication.
- All traffic to and from the server must be encrypted with standard protocols, e.g. TLS/HTTPS.
- The central server is exposed to the network under a known domain and must be secure such that doxxing and Mines access is impossible.
- The bare minimum requirements of what must be exposed for server function must be documented.
- Exposure points must be secured.

Definition of Done

There must be a way for a RPi to automatically connect to a WiFi network (school or home), password protected or not. The RPi must detect when the IP address and network configuration changes, and must provide the student the IP address and connected network name of the RPi, so a headless setup can be achieved. The client does not have specific tests, but it must be robust in demonstrations under many conditions.

The deliverable for this project is a GitHub repository with all scripts, Docker server images, and their corresponding documentation, and instructions for setting up the Raspberry Pi and server. The repository should also contain documentation about what functionality is implemented, how it is implemented, and why it is implemented that way. Lastly, documentation for usage and potential further improvements.

System Architecture

There are two fundamental components to the design: the web server and the RPI. The RPI can automatically connect once it has been registered by the user (both MAC address and device ID). Once connected, it sends its IP and status information to the web server, as illustrated in Figure 1.

The web server can be accessed by the students, secured behind MultiPass (indicated by the IDP block in the same Figure). Students can then obtain network and status information for their RPIs. If the user is an administrator (instructors), the RPIs for all students are displayed. The server itself comprises a set of micro-services in Docker containers handled via Docker Compose, a multi-container management system. The three container services that make up the server are shown at the top of Figure 2: Apache, FastAPI, and PostgreSQL. Apache acts as a reverse proxy, handles HTTPS encryption, and interfaces with Shibboleth

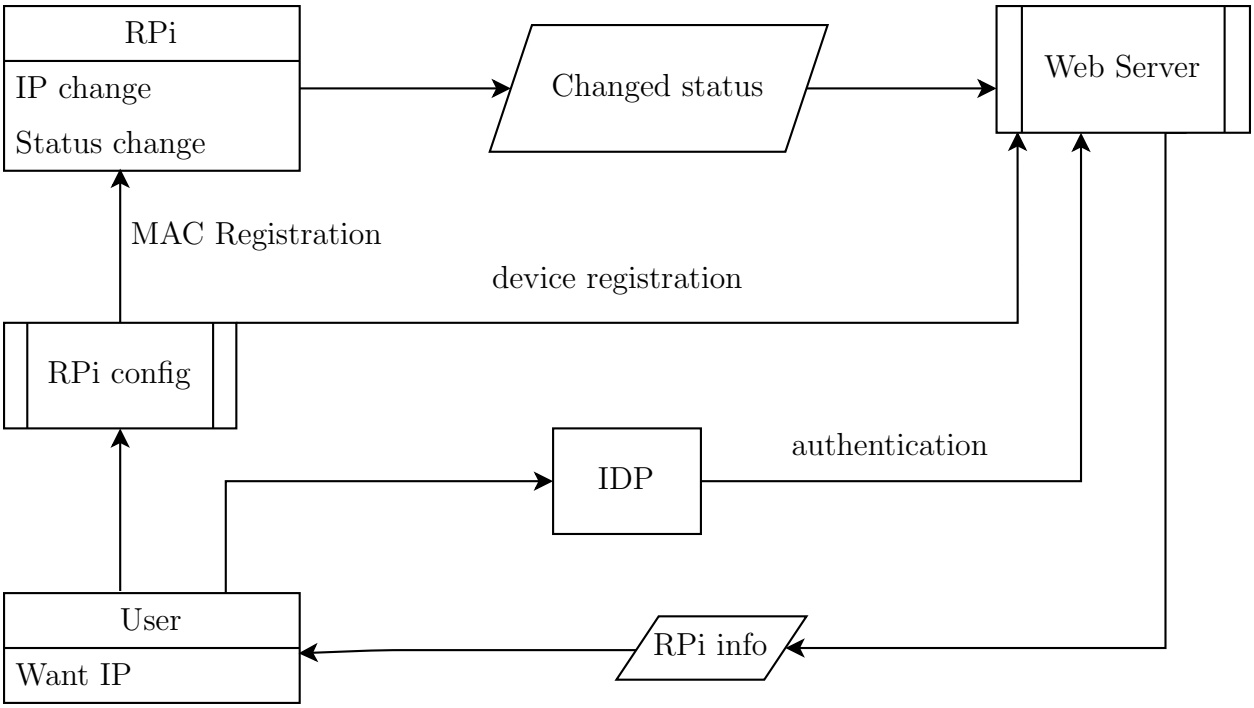

Figure 1: Overall system architecture[ab]

[a]RPI config is Figure 5
[b]Web Server is Figure 2

(the system responsible for Mines MultiPass) providing authentication. Apache routes all traffic after login to FastAPI. FastAPI provides the logic of the server, handling RPI status updates and generating user pages. The data used by FastAPI is stored in a PostgreSQL relational database.

The RPI side has two components: IP/status discovery and automatic configuration. IP/status discovery comprises a set of triggers that fire on specific system events to send IP/status updates to the server and the actual logic of sending the updates. Automatic configuration consists chiefly in providing the MAC address to the student via the boot drive and providing utilities for adding additional WiFi networks.

Technical Design

Web Server

The web server has one main role: it must receive information from the RPIs and provide that information to authenticated users. The server performs other smaller tasks that are integral to this main role, such as generating device ID information to enable RPI registration.

The web server comprises a set of docker images connected together with Docker compose. When the server is started, each container is placed in an isolated virtual network, able

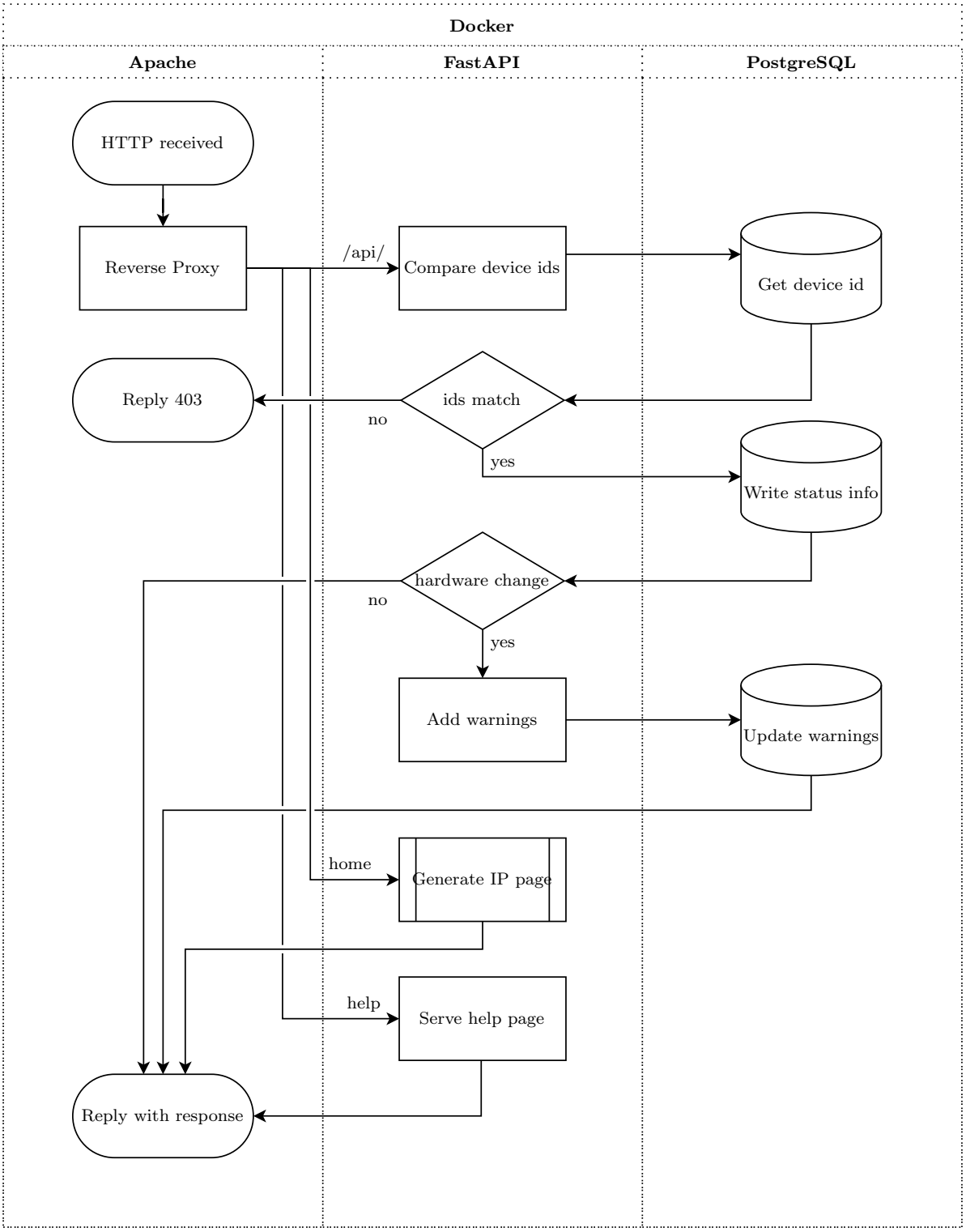

Figure 2: Server architecture swimlane flowchart[a]

[a]Generate IP page is Figure 6

to communicate only with the services it must interact with directly. The networks are configured to allow the following: database and API can communicate, API and proxy can communicate, and the proxy can communicate with the Mines network. To be clear, the proxy is the only service that can communicate directly with the Mines network. Each of the following subsections describe a single docker image.

Database

The PostgreSQL (PSQL) database stores all persistent data for the web server. The database derivesfrom the ERD in Figure 3. There are three tables: user, raspi, and raspi_warning. The first table ('user') stores data about each user registered with the system. This table consists of three fields: username, last_login, is_admin. The last_login field identifies users for removal if they have not logged in for a year. This removal is performed with a trigger in PSQL on additions to the user table. The second table ('raspi') stores information about all RPIs that have been added to the server. The second table stores IP address, device ID, hardware ID, service status, and power status information, as well as fields for last contact and creation time. The table also has a generated column that identifies a device as having been registered. Registration for a RPI is defined as the RPI having contacted the server with an existing device ID. The device IDs are generated whenever a new row is inserted into the table; the creation time is generated in the same way. The 'last_contact' is updated automatically whenever values in the row are updated. The rest of the fields are specified by the RPI on status update. The last table ('raspi_warning') stores a list of warnings mapped to RPIs along with the time the warning was added. Only one warning of a particular type may exist in the table for a given RPI. The database is administered using scripts requiring administrative access to the server itself.

API Logic

FastAPI provides the essential logic for interfacing with the database and generating the pages presented to the user. FastAPI is a lightweight Python framework. It interfaces with the database using Pyscopg2. There are four primary endpoints specified in FastAPI: home, registration page, help, and API. The three user facing pages are implemented as requiring GET requests to obtain. The homepage is the standard student web page, and an example is shown in Figure 4. This is the web page a students sees once they have at least one registered RPI. As shown on the example page, there may be a warning indicating that the device ID has been associated with a different device. If the student did not swap their SD card to a new device, the student should contact the instructor, as this may be an indication that the ID was obtained by some malicious actor, and the IP may be pointing to a malicious server. The warning is automatically cleared after the first time it is viewed. Instructors see all the students' RPIs in addition to their own.

Proxy

Apache handles TLS encryption and serves as a proxy, redirecting traffic to the API logic. It also integrates Shibboleth for providing Mines MultiPass authentication. When a user makes a request to the server, Shibboleth redirects the user to Mines' Identity Provider (IDP) server

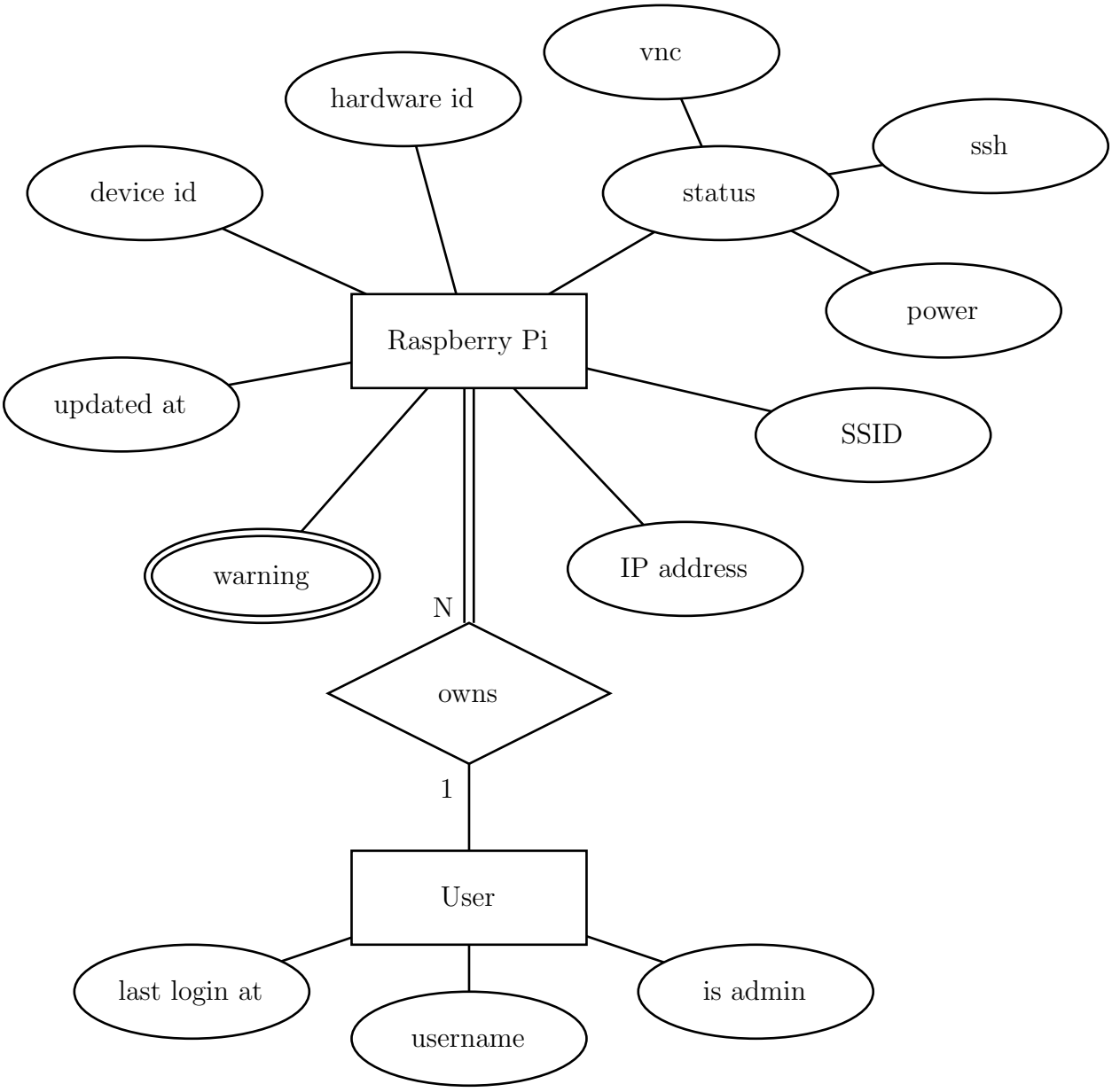

Figure 3: Database Entity Relationship Diagram

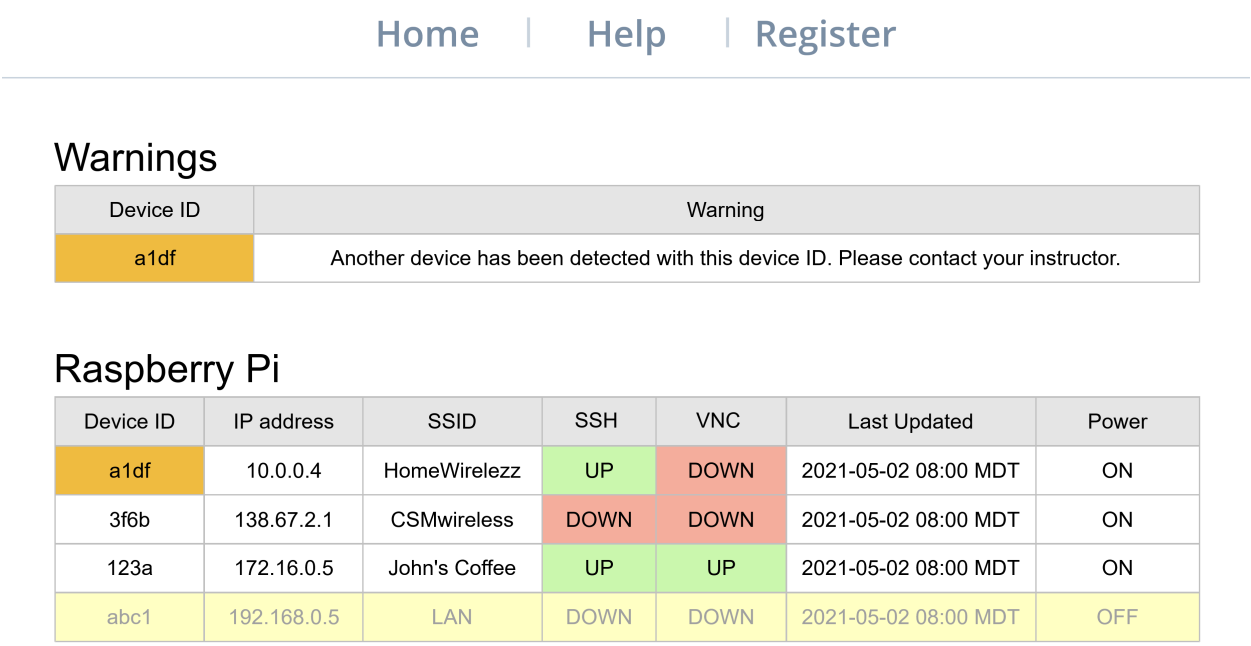

Figure 4: Example home page for a student user.

to authenticate. The IDP server then provides a session token and user information to the Shibboleth service provider (this proxy is the service provider). While Mines' IDP server can provide many different types of information, it is configured to only provide the student's username to the service provider. Shibboleth then takes this username and places it in an HTTP header before routing the request to the API logic.

Raspberry Pi

The RPI needs to be able to accomplish two primary tasks: automatic configuration and connection to the network, and automatically sending information to the server for a user to view.

The RPI configuration is split up into three parts, as shown in Figure 5. These parts must be completed in the presented order, but each sub-process can be repeated independently. Part 1 describes the process of registering the RPI's MAC address in order to connect to CSMwireless. This is done by utilizing the boot directory of the RPI so users can access the MAC address from a Windows machine. Part 2 outlines the steps to register a RPI on the web server. Again, the boot directory of the RPI is used in order to keep the RPI setup headless. The last section of Figure 5 is optional and allows users to add additional networks to the RPI. Both methods allow for the RPI to be in a headless configuration. One method uses a text file in the boot directory to deliver the network's credentials, which are automatically read, added, and configured upon RPI reboot. The second method must be completed while connected to the RPI through SSH. SSH allows the user to run a script that asks for the network's credentials.

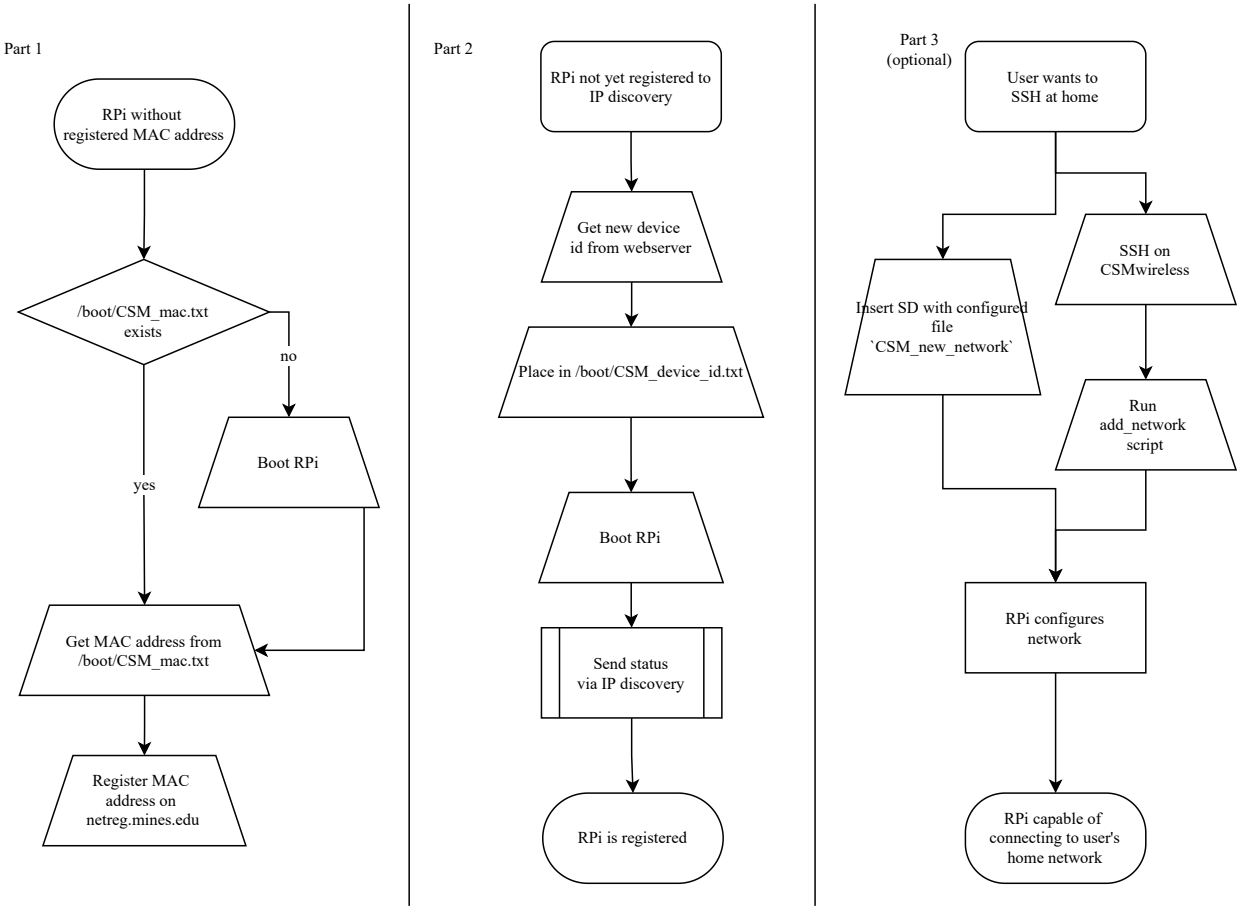

Figure 5: RPI config flowchart[a]

[a]Send status via IP discovery is Figure 6

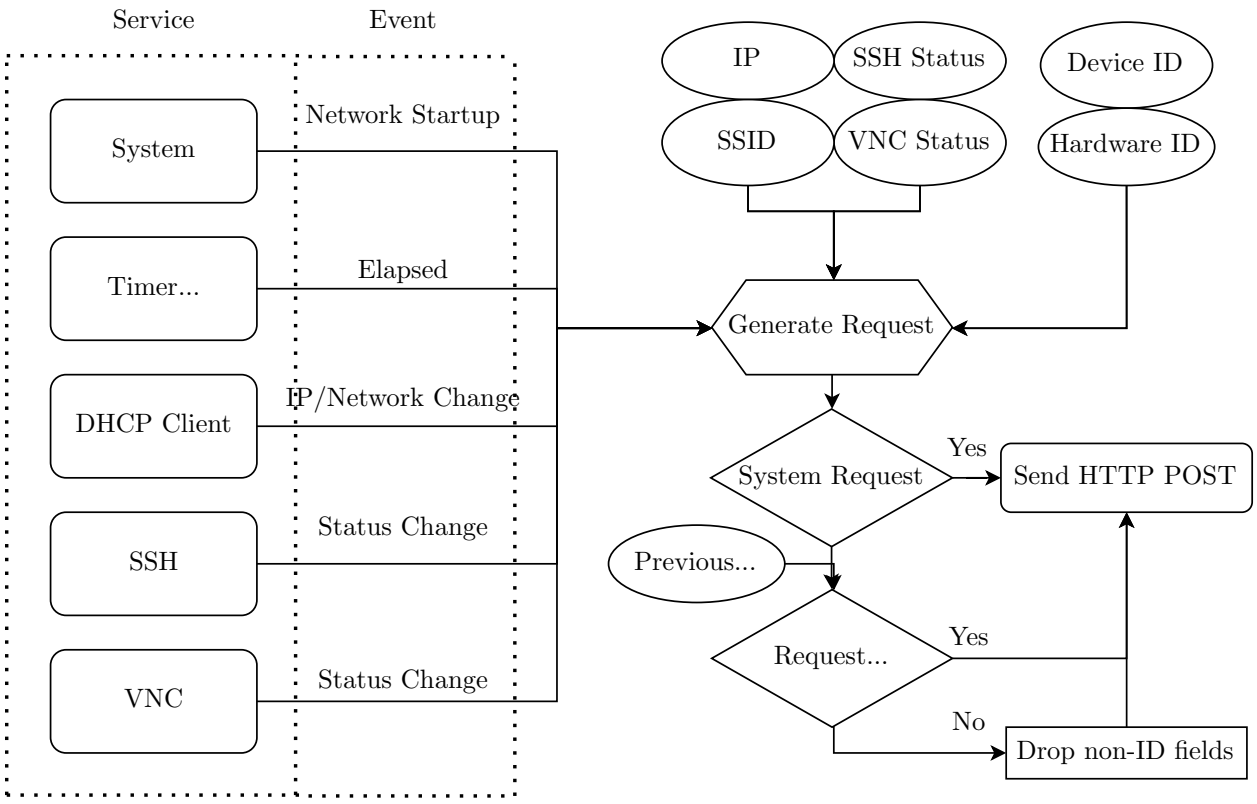

Figure 6: IP Discovery

Once configuration is complete, the RPI can send information to the server. As can be seen on the left in Figure 6, there are a set of services that can trigger IP discovery requests on certain events. The request gathers information about the network configuration (IP address(es), SSID), service/system status (SSH/VNC/system up/down), and ID information (hardware and device IDs). If the request is a startup event, the entire request is always sent. Otherwise, it is compared to the previous request. If the request is the same as the previous request, only the ID fields and event type are sent to tell the server that the RPI is still available. If the request is different from the previous request, all fields are sent.

Figure 7 describes the user interaction with the web server while retrieving their RPI's IP address. Upon accessing the homepage, the user is redirected to Mines' IDP server to obtain a session token for authentication. After authenticating, the user is redirected to the homepage, which displays their RPIs with any applicable warnings.

Quality Assurance

The following procedures were used to ensure the quality of the project, both in terms of readability and validity.

Code review for each pull request

- All code reviewed for quality prior to merging into the main branch

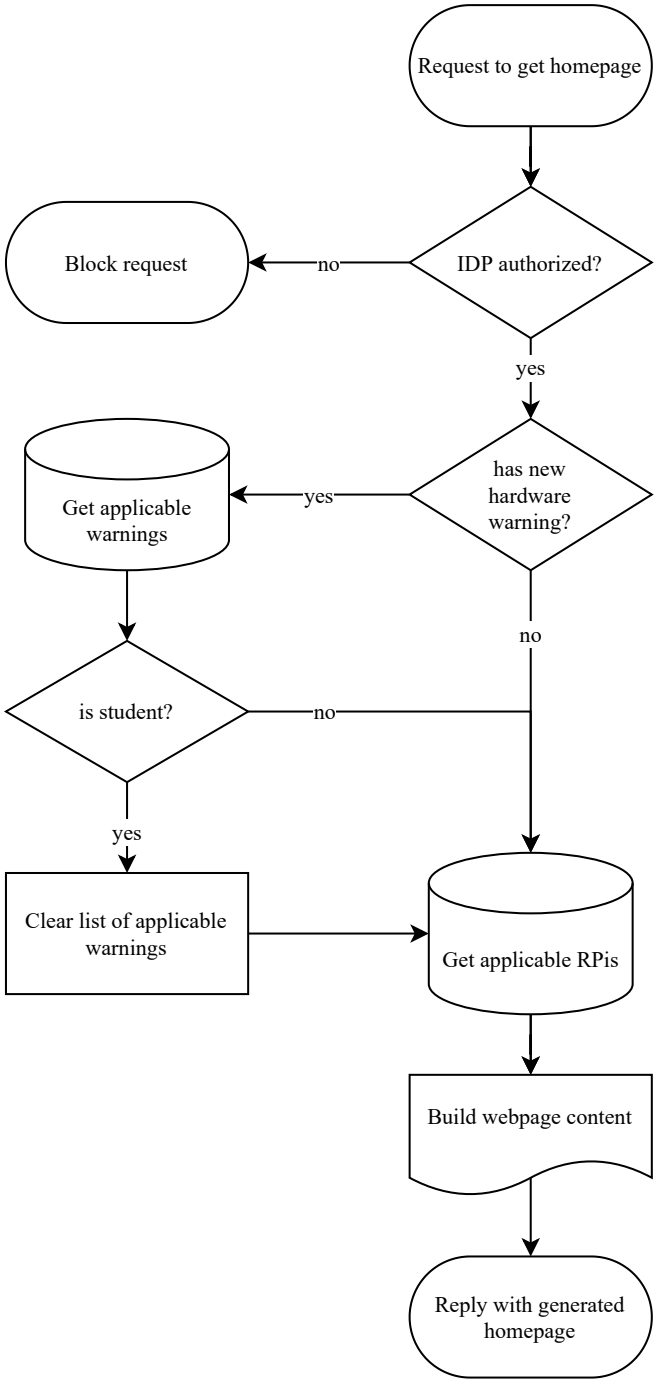

Figure 7: IP Discovery Process Flowchart

- Done by all members of Team John
- All code modifications are done via pull requests

Automated testing via RPI Docker image

- Allows for running external automated tests.
- Runs internal automated tests.
- Customized attributes like IP address.

Refactoring

- Standardized code and reviewed again by team members
- Modularization
- Ensures readability and

High coding standards with pre-commit standards tests

- Doc strings used for increased readability
- Code must pass coding standard tests on each commit and push. This is verified using the following technologies:
  - Black
    - Automatically formats code
  - isort
    - Automatically sorts Python imports
  - Flake8
    - Linter that checks for Python code complexity and code style; specifically, issues that can't be automatically fixed
  - Pydocstyle
    - Performs some rudimentary checks on Python docstrings, such as: existence and verb mood
- Future: Usability testing
- Focus group testing for usability on the end user side.
  - Provides more insight on ease of use but also potential bugs.

Results

The goal of this project was to create a method for efficiently creating a headless setup of a RPI. The RPI needs to connect to the university network, CSMwireless, automatically. The student also needs a way to know their RPI's IP address for SSH or VNC. The proposed solution correctly configures the RPI and provides a server system, allowing headless RPI configuration and network/status information discovery, meeting the client's specifications.

Much of the basic functionality like locally retrieving the MAC address, hardware serial number, and IP address were able to be tested via the allotted RPIs. However, testing on a large scale proves to be difficult, as some information, like MAC addresses, do not typically change. In response, a Docker container running Raspberry Pi OS is used for more robust testing, as device information can be customized.

Lessons Learned

- Windows and Docker do not interact well with each other. While this may not be widespread, calling `docker-compose up` on Windows would randomly fail with strange errors while the exact same command and configuration worked fine on Linux.
- PostgreSQL is a very powerful database with a lot of convenient features. For example, it simplified generating a device ID by providing a built-in way of generating it for a new RPI entry.

  It can also automatically update a last update time column when a row is updated, so that it does not have to be manually updated. It can also provide generated columns that automatically change on conditions; this allows the table to automatically identify RPI's that have been registered by checking whether a specific field is blank, without requiring the backend to set a registered flag.
- Refactoring is truly valuable. Refactoring has allowed us to delete whole scripts, simplify install processes, and condense and simplify documentation. As this software may be handed off to another team, the better modularity, extensibility, and modifiability refactoring improves the quality of the product significantly.

Non-Implemented Ideas

- It would have been preferable to automatically elevate HTTP to HTTPS for non-API endpoints but not API endpoints. However, the server system used (Apache) did not have any apparent way to do this. HTTP was disabled instead.
- It would be more convenient for the students if the MAC address for the RPI could be registered automatically; however, this turns out to be infeasible. MAC registration requires student credentials and information and is thus impossible to do automatically without storing credentials in plaintext.
- Warnings cannot be manually dismissed. There is a concern that if warnings are dismissed automatically upon viewing, if a network failure occurs or the student's home page is closed, they may miss the warning.

Future Ideas

- Enable the IP discovery system to account for multiple IP addresses per interface on the RPI. This would allow more complex network setups to work with the system if the user wanted to do this.

- Provide a way for students to customize aliases for their RPIs, allowing them to choose a more memorable name to identify the particular RPI they are using.
- Add service hooks to RPI IP discovery, allowing the user to send arbitrary service information to the service. This may be useful if the user wants to use some special network service to access the RPI and would like to see the service's status without logging into it.
- The current solution requires students to use another computer to view text files on the RPI's SD card which could be streamlined.
- Encrypt the device ID at rest. This would likely require a runtime-specific key that the server does not know and thus some form of zero-knowledge proof system.

Appendix

Installation instructions

Build image

1. `cd /path/to/autopi/`
2. `chmod +x install`
3. `./install`
4. Remove autopi project: `cd ..; rm -rf autopi`

Setting up the web server

The server is managed by Docker Compose. While most of the server setup is internal to the container system and does not require setup on the host side, there are some important exceptions.

Database password

Setting the database password requires two steps: creating the password file and updating `docker-compose.yaml` to the password file.

1. Add a strong unique password to a password file. This is, by default, `.db_password.secret`. For a random password, the following command can be run:

```
cat /dev/urandom | head --bytes 64 | sha256sum - | cut -d ' ' -f 1 |
↪  tee .db_password.secret
```

2. If you used a password file other than `.db_password.secret`, that must be changed in `docker-compose.yaml`. If the respository is managed by Git, add the file to `.gitignore` (`.gitignore` already contains `*.secret`):

```
  yaml
secrets:
  db_password:
    file: your_file_here
```

An alternative strategy is to use Docker Swarm instead of Docker Compose, so that secrets are stored in a vault, can be rotated, etc., but this is not how this project was developed.

Database backups

Database backups are managed by the prodrigestivill/postgres-backup-local image.

Unlike other Docker data, the database backups are not kept in a named Docker volume, but instead in a bind mount. This prevents accidental backup deletion during Docker volume

management (e.g. `docker-compose down --volume`) and can simplify remote backups if necessary.

The bind must be configured:

1. Create the directory. By default, `docker-compose.yaml` assumes `/var/opt/pgbackups` is used:

   ```
   sudo mkdir -p /var/opt/pgbackups && sudo chown -R 999:999 /var/opt/pgbackups
   ```

2. If you used a directory other than `/var/opt/pgbackups,` update `docker-compose.yaml` :

   ```
     yaml
   db_backup:
     volumes:
       - your_dir_here:/backups
   ```

Backup timings are set in the `db_backup.env` file. Currently, standard defaults are used:

- backups are made daily
- daily backups are kept 7 days
- weekly backups are kept 4 weeks
- monthly backups are kept 6 months

TLS certificates

There are two sets of certificates needed. First, the actual autopi.mines.edu certificates are expected to be in a folder called `tls,` this folder should be in the same directory as `docker-compose.yaml,` that is the project root. There should be three files here:

```
tls/
|
| - autopi_chain.pem
| - autopi_server.cer
| - autopi_server.key
```

The chain file consists of the intermediate certificates. The cer file is a pem certificate with the actual `autopi.mines.edu` certificate. The key is the private key. These are not stored in the repository and must be supplied.

Additionally, Shibboleth requires a key and certificate as well. These are very specific files, but they are not provided in the repo. When obtained, they should be placed in the folder

`src/web/shib/` and should be called `sp-cert.pem` and `sp-key.pem` for the certificate and key respectively. Without these, MultiPass cannot function; the image will also fail to build.

Firewall rules

The only port that must be exposed is port `:443`. Only the `proxy` service exposes external ports, and it only accepts `https` traffic; all other traffic are in segmented internally managed networks that cannot be accessed from outside the Docker service stack.

Migrating the web server

Migration is simple, as the Docker containers are ephemeral. Migrating each volume ensures the server state remains constant.

For each named volume (which can be found under the global `volumes` directive in `docker-compose.yaml`), follow the official Docker documentation on backing up, migrating, and restoring with named volumes.

For the database backups, it is as simple as copying the contents of the backup directory from the old host machine to the new host machine

Setup

There are two required setup steps. First, the RPI must have its MAC address registered with the school so it can connect to the internet on `CSMwireless`. Second, the RPI must be registered with the IP/Status discovery system. This registration can only occur once the RPI is connected to the network (i.e. has performed network registration). Custom networks can also be added to the RPI.

Device and MAC Registration

1. Insert the SD card into the RPI.
2. Plug in the RPI, lights will turn on.
3. Wait for 2 minutes.
4. Unplug the RPI.
5. Remove the SD card from RPI.
6. Insert SD card into your computer.
7. Open the SD card drive on your file explorer (named `boot`).
8. Open the file `CSM\_mac\_address.txt`.
9. Go to `netreg.mines.edu` while connected to `CSMwireless`.
10. Agree to the terms and conditions.
11. On the next page, enter the MAC address from the file in step 7.

12. Enter the rest of your information and click register.
13. The RPI's MAC address is now registered.
14. The RPI will be able to access internet in up to 5 minutes, meanwhile, continue steps.
15. Reopen the SD card drive on your file explorer (named `boot`).
16. Open the file `CSM\_device\_id.txt` .
17. Go to `https://autopi.mines.edu/` .
18. Login with your Mines MultiPass.
19. Click `Register` at the top of the page.
20. Copy the `ID` into the file opened in step 16.
21. Save and close the file.
22. Eject the SD card and plug it into the RPI.
23. Plug in the RPI, lights will turn on.
24. See *Get IP/status information* to confirm successful registration.
25. To add additional networks, see *Home Network Registration* procedure.

Home Network Registration

There are two methods for adding a Home network to the RPI. -

- If you *can* SSH into RPI:
    1. Type `CSM\_add\_home\_network` into a terminal window on the RPI.
    2. Follow the prompts and enter network information.
    3. Once completed, network settings will reconfigure.
- If you *cannot* SSH into the RPI:
    1. Insert SD card into your computer.
    2. Open the SD card drive on your file explorer (named `boot`).
    3. Find file `CSM\_new\_network.txt` .
    4. Read comments and fill out listed fields.
    5. Eject SD card from your computer.
    6. Insert SD card into RPI.
    7. Turn on RPI, and network settings will reconfigure.
    8. `CSM\_new\_network.txt` will be reset to blank parameters for adding additional networks.

Get IP/status information

In order to see the IP and status information for the RPI, do the following:

1. Go to `https://autopi.mines.edu/` .
2. Login with your Mines MultiPass.
3. Your registered RPI will be displayed with all corresponding IP, network, and status information. If you have multiple RPI's, they will all be displayed.
4. Refresh page for status changes.
5. Note that RPIs highlighted in yellow are off or cannot be contacted for some other reason. If this occurs, try restarting the RPI.

Development instructions

Run:

- `pip3 install -r requirements.txt`
- `pip3 install -r requirements-dev.txt`
- `pre-commit install`

When `pre-commit` fails (and a hook states that files have changed), run `git add -{}-update` to update the git staging area.

Glossary

Headless	Not requiring a mouse, keyboard, or monitor to be physically connected..
RPI	Raspberry Pi; a miniature Linux computer..
SSH	Secure SHell; provides secure access to a device over a network.
VNC	Provides remote access to a desktop interface over a network.