



Who Is Home?: A Network Scanner

Raspberry Pi Network Monitor With AWS Cloud Integration To Determine Who Is Home Based On Mobile Device Network Connections

Marc Geggan

CMP408: IoT and Cloud Secure Development

2022/23

**Note that Information contained in this document is for educational purposes.*

+Contents

1	Introduction	1
1.1	Background.....	1
1.2	Aims.....	1
2	Procedure.....	2
2.1	Methodology	2
2.1.1	IoT Device Procedure Overview	2
2.1.2	Script – Frontend.py	3
2.1.3	LKM – LED (Flashing Alert)	3
2.1.4	MQTT	3
2.1.5	AWS – EC2 (Backend Script).....	4
2.1.6	Security	4
3	Conclusion	6
3.1	General Discussion.....	6
3.2	Future Work.....	6
	References	7
	Appendices.....	8
	Appendix 1 – Frontend.py.....	8
	Appendix 2 – Backend.py.....	10
	Appendix 3 – led_flash.c (Linux Kernel Module).....	12

1 INTRODUCTION

1.1 BACKGROUND

This project is a proof of concept for access control that utilizes a Raspberry Pi Zero to monitor a home network for specific IP addresses in order to detect connected devices and individuals. The Pi runs NMAP scans locally and transmits the data to an AWS EC2 instance via MQTT for display on a static web page. The network scanner is capable of detecting the connection and disconnection of specific devices, with a focus on mobile device IP addresses, and logging times, dates, and the number of individuals present in the home. This concept can be utilized to track the comings and goings of individuals through their connection of devices to the network.

1.2 AIMS

For this network scanning IoT project to be successful, a few aims/objectives must be met:

- Software
 - Python script hosted locally to handle network scans, MQTT messages, and loading/removing lkm02.ko file depending on logic:
 - If new device detected
 - Send JSON string MQTT message to EC2 of NMAP scan results.
 - Load LKM, wait 6 seconds, remove LKM.
 - Loop back, scan again.
 - If device already on network
 - Update MQTT message but change no data.
 - If device no longer on network
 - Send JSON string MQTT message to EC2 of updated scan results.
 - Load LKM, wait 6 seconds, remove LKM.
 - Loop back, scan again.
- Hardware
 - Utilizing GPIO pins and a Linux Kernel Module (LKM) to flash an alerting LED light.
 - When a device joins or leaves the network, the LKM02.ko script is called, and an LED attached to the Pi's GPIO pins will flash 5 times.
- Cloud
 - MQTT will be used to constantly pass messages from the Pi's NMAP scan results to the EC2 instance.
 - Backend python script on EC2 converts MQTT messages to HTML file.
 - Index.html file saved to var/www/html
 - Index.html viewed over the internet by visiting the static page hosted on the EC2.

2 PROCEDURE

2.1 METHODOLOGY

2.1.1 IoT Device Procedure Overview

For the IoT portion of the project, a python script is looped every 20 seconds. This script initializes an NMAP scan that scans the desired network for pre-defined targets. If the target(s) have been identified as joining the network, an MQTT message will be sent to the EC2 instance, and the information will be stored on a static web page. If the device(s) have already been identified, and are still on the network, an MQTT message will be sent to the EC2, however no information will be updated. If any target device(s) are no longer on the network, an MQTT message will be sent to the EC2 instance and the html static web page will be updated with the NMAP scan information. Lastly, if a new device is detected or existing device is no longer detected, the LKM02.ko file is called by the python script, which will initiate a flashing LED light attached to the RPI's GPIO pins. After 5 flashes, the LKM will be removed, and the script will continue looping.

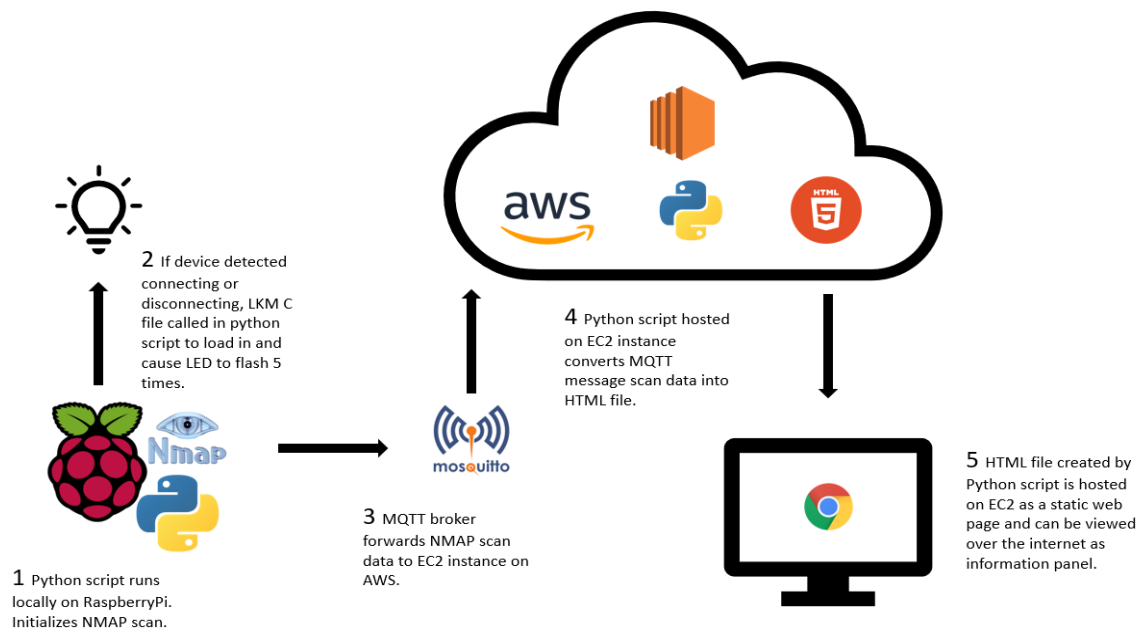


Figure 1 – Overview of IoT project architecture.

2.1.2 Script – Frontend.py

For the software aspect of the project, python scripts were implemented. A python script named frontend.py was the main driving force behind the project and handled multiple processes depending on certain outcomes and logic. This script can be viewed at *Appendix 1 – Frontend.py*

The python script scans the home network for pre-determined devices every 20 seconds using NMAP. The script then iterates through the list of devices found on the network. If the scan returns with a new detected device on the network, multiple things will happen. Firstly, the NMAP scan saves the date and time of when the device was detected on the network to a dictionary, updates a list with the names of peoples devices detected, and converts this into a message variable to be sent over MQTT. This message is then converted into a JSON string and using the line '*client.publish(topic, message)*', the MQTT message is sent to an EC2 instance. After this has been sent, the LKM is loaded using '*os.system("sudo insmod lkm02.ko")*'. The LKM c file can be viewed at *Appendix 3 – led_flash.c*.

If the network scan runs a second time and detects that a previously connected device has disconnected from the network, the IP address will be removed from the list of connected devices, and the date and time of when the device has disconnected is saved in the 'out' section of the log. This information is then converted into a message variable, which is then converted into a JSON string, and sent over MQTT. This change in device status will also load the Linux Kernel Module to flash the alerting LED light. Also, if no change in the connection of the device is detected, the connection date and time, and connected device name is not changed and viewable on the EC2 instance. After the scan has been completed, the script will wait x number of seconds and loop again until an interrupt signal from keyboard is detected.

2.1.3 LKM – LED (Flashing Alert)

For the hardware aspect of the project, a red LED light was connected to the RaspberryPi. This was done by connecting an LED to a resistor on a breadboard, and connecting from the RaspberryPi's GPIO pins to the breadboard. Next, a C file was created. This file was called led_flash.c and was adapted from the lkm02 file provided by the CMP408 module. A for loop function was implemented that allowed for the LED to flash 5 times. A kernel object file was then built and compiled out of the led_flash.c file using a make file. When the Frontend python script meets certain logic, the lkm02.ko file is loaded in using '*os.system("sudo insmod lkm02.ko")*' and removed using '*os.system("sudo rmmod lkm02.ko")*'. The led_flash.c file can be viewed at *Appendix 3 – led_flash.c*

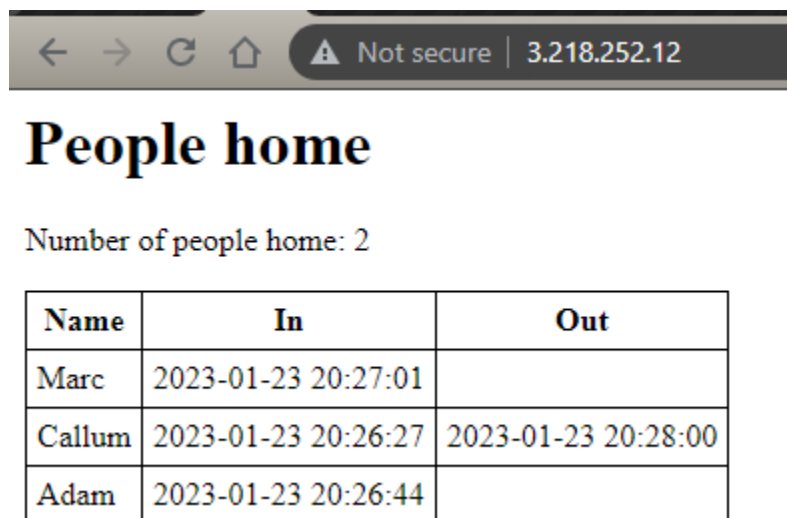
2.1.4 MQTT

To enable the cloud section of the project, it was decided that MQTT would be used to pass messages from the Pi to the cloud services. MQTT is becoming a popular protocol for IoT data transfer and was designed for connecting remote devices to a central service (*AutomationWorld, 2022*). To allow for connection using MQTT, a broker must be set up. After connecting to the EC2 instance over SSH, a broker called 'Mosquitto' was installed that will allow the Pi to send and the EC2 to receive MQTT messages. To view the messages being sent between the Pi and EC2 instance, the software 'MQTT Explorer' was installed. Entering the IP address of the EC2 instance to this software will connect to Mosquitto and the messages being sent can be viewed. This allows for debugging any issues that arise. The python script

running locally on the Pi will handle the conversion of scan data to messages, connect to the broker, and send the MQTT message.

2.1.5 AWS – EC2 (Script – Backend.py)

For the cloud aspect of the project, the Amazon Web Services EC2 instance was implemented. It was decided that the EC2 instance was appropriate as it has the ability to host static web pages, and run python scripts. Within the EC2 instance, a python script was created called 'backend.py'. This can be viewed at Appendix 2 – Backend.py. The python script receives the JSON message sent via MQTT. It then parses the message and prints the data, converting it into an HTML file. This HTML file is designed to display the data in an easily viewable table. The HTML file is then written in the var/www/html directory and saved as index.html. This script is looped until a keyboard interrupt is detected to ensure that the connection doesn't unexpectedly time out. As the frontend python script loops every 10 seconds or so, MQTT messages are constantly being sent. This means the index.html file is being updated frequently and live results can be viewed by visiting the web page 'www. {EC2 Instance IP} /index.html' (See figure 2).



Name	In	Out
Marc	2023-01-23 20:27:01	
Callum	2023-01-23 20:26:27	2023-01-23 20:28:00
Adam	2023-01-23 20:26:44	

Figure 2: Index.html static page hosted on EC2 instance, outputting scan results from nmap scan.

2.1.6 Security

Some security measures were also considered during the implementation of this project:

- Raspberry Pi SSH Connection
 - The password for creating SSH connections from Host to RaspberryPi was modified from the generic password to a more complex password.
- Private keys for ssh into ec2
 - When setting up the EC2, a security group was added.
 - This requires a private key file (.pem) keypair to be created.
 - This must be used when connecting from host machine to EC2.

- MQTT broker username and password
 - A username and Password can be set up to connect to the broker.
 - Sudo nano /etc/mosquitto/mosquitto.conf
 - Allow_anonymous false
 - Listener 1883
 - Password_file /etc/mosquitto/passwd
 - Set username and password with sudo mosquitto_passwd /etc/mosquitto/passwd {name} {passwd}

3 CONCLUSION

3.1 GENERAL DISCUSSION

In conclusion, the network scanning IoT project was successful in meeting all the objectives set out. The project incorporated software, hardware, and cloud aspects to build a working IoT device with web page integration. Using technologies such as MQTT, the RaspberryPi was able to successfully communicate with Amazon cloud services to transfer data from a locally hosted device to the cloud, which in turn cost very little to run. Security has been considered and implemented throughout the project, with at least one security measure implemented on the hardware device, data transfer service, and cloud service. This project successfully imitates a proof-of-concept access control type device that, when set up with IP addresses and names, can successfully show who is in a home by detecting pre-defined devices that are connected to a home network.

3.2 FUTURE WORK

This project, although meeting all objectives, has room for future work such as:

- Improving the style and design of index.html
- Enabling the use of static MAC addresses of devices will allow for more stable results.
- Utilize databases to store times and dates of devices interacting with the network.
- Use more information from NMAP scan results to display further metrics on the devices attached to the network.

REFERENCES

- Chandra, M. (2021). Host a static website on AWS EC2. [online] Available at: https://www.linkedin.com/pulse/host-static-website-aws-ec2-chandan-mishra?trk=articles_directory.
- Amazon.com. (2019). Getting Started with Amazon EC2 Linux Instances - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html.
- steve (2019). Paho Python MQTT Client – Publish With Examples. [online] Steves-internet-guide.com. Available at: <http://www.steves-internet-guide.com/publishing-messages-mqtt-client/>.
- Grgic, K., Speh, I. and Hedi, I. (2016). A web-based IoT solution for monitoring data using MQTT protocol. 2016 International Conference on Smart Systems and Technologies (SST). doi:10.1109/sst.2016.7765668.
- Automation World. (2022). *Advantages of a Smart MQTT Broker*. [online] Available at: <https://www.automationworld.com/factory/iiot/article/22235994/advantages-of-a-smart-mqtt-broker>
- Lima, D.B.C., da Silva Lima, R.M.B., de Farias Medeiros, D., Pereira, R.I.S., de Souza, C.P. and Baiocchi, O. (2019). A Performance Evaluation of Raspberry Pi Zero W Based Gateway Running MQTT Broker for IoT. [online] IEEE Xplore. doi:10.1109/IEMCON.2019.8936206.
- Rui, S. (2022). Install Mosquitto Broker Raspberry Pi | Random Nerd Tutorials. [online] Available at: <https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/#:~:text=The%20MQTT%20broker%20is%20responsible>
- iamrachel (n.d.). How to Setup Mosquitto MQTT on AWS. [online] Instructables. Available at: <https://www.instructables.com/How-to-Setup-Mosquitto-MQTT-on-AWS/>
- Ask Ubuntu. (2013). Mount error: 'unknown filesystem type "exfat"'. [online] Available at: <https://askubuntu.com/questions/364270/mount-error-unknown-filesystem-type-exfat>
- Derek (2015). *Writing a Linux Kernel Module — Part 1: Introduction*. [online] derekmolloy.ie. Available at: <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>

APPENDICES

APPENDIX 1 – FRONTEND.PY

```
import nmap
import time
from datetime import datetime
import json
import paho.mqtt.client as mqtt
import os

# MQTT Data
broker_url = "{ENTER IP ADDRESS OF AWS EC2}" # Modify this address
broker_port = 1883
topic = "people_data"

# Specify the IP addresses of the devices you're looking for
target_ips = {
    '{IP ADDRESS}': '{NAME}', #Enter target IP and persons name
    '{IP ADDRESS}': '{NAME}', #Add more lines as desired
    '{IP ADDRESS}': '{NAME}'
}

# Initialize a dictionary to store the status of the devices
device_status = {}
for ip in target_ips:
    device_status[ip] = False

# Initialize a list to store the names of people home
people_home = []

# Initialize a dictionary to store the log of when each person comes
# and goes
people_log = {}
for name in target_ips.values():
    people_log[name] = {'in': None, 'out': None}

# Create an instance of the MQTT client
client = mqtt.Client()

# Connect to the MQTT broker
client.connect(broker_url, broker_port)

while True:
    # Scan the local network for devices
    nm = nmap.PortScanner()
    nm.scan(hosts='192.168.0.1/24', arguments='-sn') #IP to be
    scanned by Nmap

    # Iterate through the list of hosts
```

```

for host in nm.all_hosts():
    if host in target_ips:
        if nm[host]['status']['state'] == 'up':
            if not device_status[host]:
                # Device has just joined the network
                device_status[host] = True
                print('Welcome home:
{}'.format(target_ips[host]))
                people_home.append(target_ips[host])
                people_log[target_ips[host]]['in'] =
datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                # Create a message to send over MQTT
                message = {
                    'people_home': people_home,
                    'people_log': people_log,
                    'num_people_home': len(people_home)
                }
                # Convert message to json string
                message = json.dumps(message)
                # Send the message over MQTT
                client.publish(topic, message)
                os.system("sudo insmod lkm02.ko") #Insert kernel
module

                time.sleep(6)
                os.system("sudo rmmod lkm02.ko") #Remove kernel
module

            else:
                if nm[host]['status']['state'] == 'down':
                    if device_status[host]:
                        device_status[host] = False
                        print('Goodbye:
{}'.format(target_ips[host]))
                        people_home.remove(target_ips[host])
                        people_log[target_ips[host]]['out'] =
datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                        # Create a message to send over MQTT
                        message = {
                            'people_home': people_home,
                            'people_log': people_log,
                            'num_people_home': len(people_home)
                        }
                        # Convert message to json string
                        message = json.dumps(message)
                        # Send the message over MQTT
                        client.publish(topic, message)
# check if the device is not found in the scan
for ip in target_ips:
    if ip not in nm.all_hosts():
        if device_status[ip]:
            device_status[ip] = False
            print('Goodbye: {}'.format(target_ips[ip]))
            people_home.remove(target_ips[ip])

```

```

        people_log[target_ips[ip]]['out'] =
datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        # Create a message to send over MQTT
        message = {
            'people_home': people_home,
            'people_log': people_log,
            'num_people_home': len(people_home)
        }
        # Convert message to json string
        message = json.dumps(message)
        # Send the message over MQTT
        client.publish(topic, message)
        os.system("sudo insmod lkm02.ko") #Insert kernel
module
        time.sleep(6)
        os.system("sudo rmmod lkm02.ko") #Remove kernel
module
    print("Device status: ", device_status)
    print("People home: ", people_home)
    print("People log: ", people_log)
    print("Number of people home: ", len(people_home))

print("*****")
print(" ")
# Wait for 20 seconds before running the next scan
time.sleep(5)

```

APPENDIX 2 – BACKEND.PY

```

import paho.mqtt.client as mqtt

import json

#MQTT Data

broker_url = "{ENTER IP ADDRESS OF AWS EC2}" # Modify this address

broker_port = 1883

topic = "people_data"

def onMessageReceived(client, userdata, message):

    # Parse the json message

    message = json.loads(message.payload.decode("utf-8", "ignore"))

```

```

print("Message received: ", message)

# Create the HTML file

html = "<html><head>"

html += "<style>"

html += "table, th, td {border: 1px solid black; border-collapse: collapse;}"

html += "th, td {padding: 5px;}"

html += "</style>"

html += "</head><body>"

html += "<h1>People home</h1>"

html += "<p>Number of people home: " + str(message["num_people_home"]) + "</p>"

html += "<table>"

html += "<tr><th>Name</th><th>In</th><th>Out</th></tr>"

for name in message["people_log"]:

    html += "<tr>"

    html += "<td>" + name + "</td>"

    html += "<td>" + (message["people_log"][name]["in"] if message["people_log"][name]["in"] is not
None else "") + "</td>"

    html += "<td>" + (message["people_log"][name]["out"] if message["people_log"][name]["out"] is
not None else "") + "</td>"

    html += "</tr>"

html += "</table>"

html += "</body></html>"

# Save the HTML file

with open("/var/www/html/index.html", "w") as f:

    f.write(html)

client = mqtt.Client()

```

```
client.on_message = onMessageReceived
```

```
client.connect(broker_url, broker_port)
```

```
client.subscribe(topic)
```

```
# Start the loop to keep the script running, exit with CTRL + C
```

```
try:
```

```
    client.loop_forever()
```

```
except KeyboardInterrupt:
```

```
    print("Exiting script")
```

```
    client.disconnect()
```

APPENDIX 3 – LED_FLASH.C (LINUX KERNEL MODULE)

```
#include <linux/kernel.h>
```

```
#include <linux/module.h>
```

```
#include <linux/init.h>
```

```
#include <linux/fs.h>
```

```
#include <linux/device.h>
```

```
#include <linux/gpio.h>
```

```
#include <linux/delay.h>
```

```
#define DEVICE_NAME "lkm02"
```

```
#define MAJOR_NUM 42
```

```
#define PIN_NUM 16
```

```
#define FLASH_COUNT 5
```

```
static int lkm02_open(struct inode *inode, struct file *file){
```

```
    pr_info("CMP408: %s\n", __func__);
```

```
    return 0;
```

```

}

static int lkm02_release(struct inode *inode, struct file *file){
    pr_info("CMP408: %s\n", __func__);
    return 0;
}

static ssize_t lkm02_read(struct file *file,
                          char *buffer, size_t length, loff_t * offset){
    pr_info("CMP408: %s %u\n", __func__, length);
    return 0;
}

static ssize_t lkm02_write(struct file *file,
                           const char *buffer, size_t length, loff_t * offset){
    pr_info("CMP408: %s %u\n", __func__, length);
    return length;
}

struct file_operations lkm02_fops = {
    .owner = THIS_MODULE,
    .open = lkm02_open,
    .release = lkm02_release,
    .read = lkm02_read,
    .write = lkm02_write,
};

int __init lkm02_init(void){
    int ret;
    int i;
    pr_info("CMP408: %s\n", __func__);

    ret = register_chrdev(MAJOR_NUM, DEVICE_NAME, &lkm02_fops);

```



```

    if (ret != 0)
        return ret;

    gpio_request(PIN_NUM, "A1");
    gpio_direction_output(PIN_NUM, 0);

    for (i = 0; i < FLASH_COUNT; i++) {
        gpio_set_value(PIN_NUM, 1);
        msleep(1000);
        gpio_set_value(PIN_NUM, 0);
        msleep(1000);
    }

    printk("CMP408: lkm02 loaded\n");
    return 0;
}

void __exit lkm02_exit(void){
    unregister_chrdev(MAJOR_NUM, DEVICE_NAME);
    gpio_set_value(PIN_NUM, 0);
    gpio_free(PIN_NUM);
    printk("CMP408: lkm02 unloaded\n");
}

module_init(lkm02_init);
module_exit(lkm02_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Marc Geggan");
MODULE_DESCRIPTION("Rpi Zero W GPIO device driver with linux kernel GPIO library");
MODULE_VERSION("0.3");

```