Abertay University

Report Into Internet Speed Testing IoT Mini-Project

Isaac Basque-Rice

BSc. (Hons.) Ethical Hacking
Abertay University
Dundee, United Kingdom
1901124@abertay.ac.uk

January 24th, 2023
1497 Words

Contents

List of Figures

List of Acronyms

DoS Denial of Service
AWS Amazon Web Services
LED Light Emitting Diode
GPIO General-Purpose Input/Output
CLI Command Line Interface
LKM Linux Kernel Module
SSID Service Set Identifier
JSON JavaScript Object Notation
EC2 Elastic Compute Cloud
MQTT Message Queueing Telemetry Transport
IoT Internet of Things
SSL Secure Sockets Layer
HTTPS Hypertext Transfer Protocol Secure
GUI Graphical User Interface
SSH Secure Shell

1 Introduction

Monitoring networks is crucial as reliance on them grows in the internet age. Businesses need stable, low-latency networks to function and changes can result in significant revenue losses. Implementing a monitoring system can benefit organizations and individuals.

The project aims to create a hardware network speed tester with a web server on Amazon Web Services (AWS) to display results. It will consist of a Raspberry Pi Zero W, a button to start the test, and an Light Emitting Diode (LED) to indicate that the speedtest is running. Supporting hardware such as resistors, wires, and a breadboard will also be used, all connected to the Pi's General-Purpose Input/Output (GPIO) pins.

1.1 Relevance to System Internals & Cybersecurity

This project is relevant to System Internals in several areas, particularly the usage of GPIO pins and specific input and output devices interfacing with the device at a low level. It requires interfacing with Linux OS through Linux Kernel Module (LKM)s, which are code that can be loaded and unloaded into the kernel at runtime to provide functionality for new hardware. The project will use an LKM developed by the author to enable the button, which in turn will run the test.

Regarding Cybersecurity, the AWS web server must be set up correctly and securely. Users' IP addresses can potentially be identifying features of their network, and if the website is not set up securely then the security of the user of the device could be compromised, leaving the door open to an attack such as a network Denial of Service (DoS) or further intrusion. Additionally, if the site's integrity is not maintained then the results of the test could be open to tampering. This issue will be mitigated by the use of SSL/Hypertext Transfer Protocol Secure (HTTPS), which encrypts user traffic.

Additionally, an improperly developed LKM can have serious security implications, as it can allow an attacker to gain privileged access to the system. LKMs run with kernel-level privileges, so a vulnerability in an LKM can be used to as a method of Privilege Escalation and potentially compromise the entire system. Improperly developed LKMs can also cause system crashes or data loss. To minimize the security risks associated with LKMs, the developer will follow secure coding practices and thoroughly test the LKM before deploying the project.

1.2 Objectives

The overall objectives of this project are as follows, in order of when they are to be achieved:

- Set up basic speed testing capabilities on Raspberry Pi Command Line Interface (CLI)
- Correctly wire up all relevant hardware
- Trigger speed testing on button press
- light up LED(s) on command
- Create and host site using AWS web server
- Send data from speedtest to the web server

2 Methodology

2.1 Raspberry Pi

Project work began on the developer's Linux-based laptop initially due to the winter break restricting the hardware available to them. As such, the developer decided to develop as much as they could on their device and transfer it over at a later stage.

All of the code developed for use on the Pi is available in Appendix A, also, a new password was created on the Pi using the `passwd` command, this meant that only the developer had access to it.

The first program the developer created was `speedtester.py`, which is a CLI program making use of the `speedtest-cli` Python module (Martz 2021) that would record several pieces of information such as the network Service Set Identifier (SSID), start time, download, upload, ping, and time to complete, and write them to a JavaScript Object Notation (JSON) file for later use in the web server. Figure 1 shows the CLI output of the tool.

```
❯ ./speedtester.py
Beginning speedtest of SKYRKIZ6
Fri Jan  6 17:14:24 2023
Download speed: 48662529 b/s
Upload speed: 18340944 b/s
Ping: 31 ms
Speedtest completed in 112 seconds
Writing to JSON object...
Writing complete
```

Figure 1: The output of the `speedtester.py` tool

The main piece of work that had to be created for the Pi was the button LKM. This piece of software, designed to be run in kernel space, allows for the `speedtest.py` file to be ran on button press using a separate userspace program as an intermediary. An intermediary was used here as it is inadvisable to run a userspace application directly from kernel space, this application was intentionally left as concise as possible to ensure no security or stability complications arose. A tutorial on this was provided by **johannes4gnulinuxLetCodeLinux2022** (**johannes4gnulinuxLetCodeLinux2022**).

This intermediary, `userspace.c`, registers itself to the LKM using the `REGISTER_SAPP` macro, which contains the magic numbers 'R', 'g', which

are present in both the LKM and the userspace app, as well as the `SIGTX` macro, which is used to send the correct signal. The program then sends the signal using `SIGTX` and the `signalhandler` function which calls and runs the `speedtester.py` tool. A device file was created for the button to use at `/dev/irq_signal`, and was interacted with in the `userspace.c` app using `O_RDONLY` as an argument.

The `speedtester.py` file does the bulk of the work in userland. Functionality is present to first turn on the LED when the process gets called, start a timer run the speedtest, and save it to a JSON object. Figure 2 outlines the full process of compilation, LKM insertion, and program execution.

```
pi@speedtester:~/CMP408-Code/RPi-Code/Button-Code/LKM $ make
make -C /usr/src/linux-headers-5.15.84+ M=/home/pi/CMP408-Code/RPi-Code/Button-Code/LKM modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.84+'
  CC [M]  /home/pi/CMP408-Code/RPi-Code/Button-Code/LKM/button.o
  MODPOST /home/pi/CMP408-Code/RPi-Code/Button-Code/LKM/Module.symvers
  CC [M]  /home/pi/CMP408-Code/RPi-Code/Button-Code/LKM/button.mod.o
  LD [M]  /home/pi/CMP408-Code/RPi-Code/Button-Code/LKM/button.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.15.84+'
pi@speedtester:~/CMP408-Code/RPi-Code/Button-Code/LKM $ sudo insmod button.ko
pi@speedtester:~/CMP408-Code/RPi-Code/Button-Code/LKM $ cd ../../
pi@speedtester:~/CMP408-Code/RPi-Code $ gcc userspace.c
userspace.c: In function 'signalhandler':
userspace.c:40:12: warning: 'return' with a value, in function returning void
     return 0;
            ^
userspace.c:17:6: note: declared here
 void signalhandler(int sig) {
      ^~~~~~~~~~~~
pi@speedtester:~/CMP408-Code/RPi-Code $ ./a.out
PID: 23051
Wait for signal...
Running speedtester.py
Beginning speedtest of The Internet
Wed Jan 18 15:08:55 2023
Download speed: 18913042 b/s
Upload speed: 15711955 b/s
Ping: 30 ms
Speedtest completed in 34 seconds
Writing to JSON object...
Writing complete
Speedtest completed
```

Figure 2: The commands inputted by the developer to compile and run the LKM and associated programs

The `userspace` intermediary also triggers `mqtt_sender.py`, which sends the JSON data to the server using Message Queueing Telemetry Transport (MQTT). It does this by creating an MQTT client, setting authentication tokens, and a "topic" (an identifying string for any message being sent or received), connecting to the broker (the Elastic Compute Cloud (EC2)), and then converting the JSON file to a string and sending that string over the internet to the broker using `client.publish(topic, data)`. Figure 3 is a flow chart representing the process that takes place on the Pi.

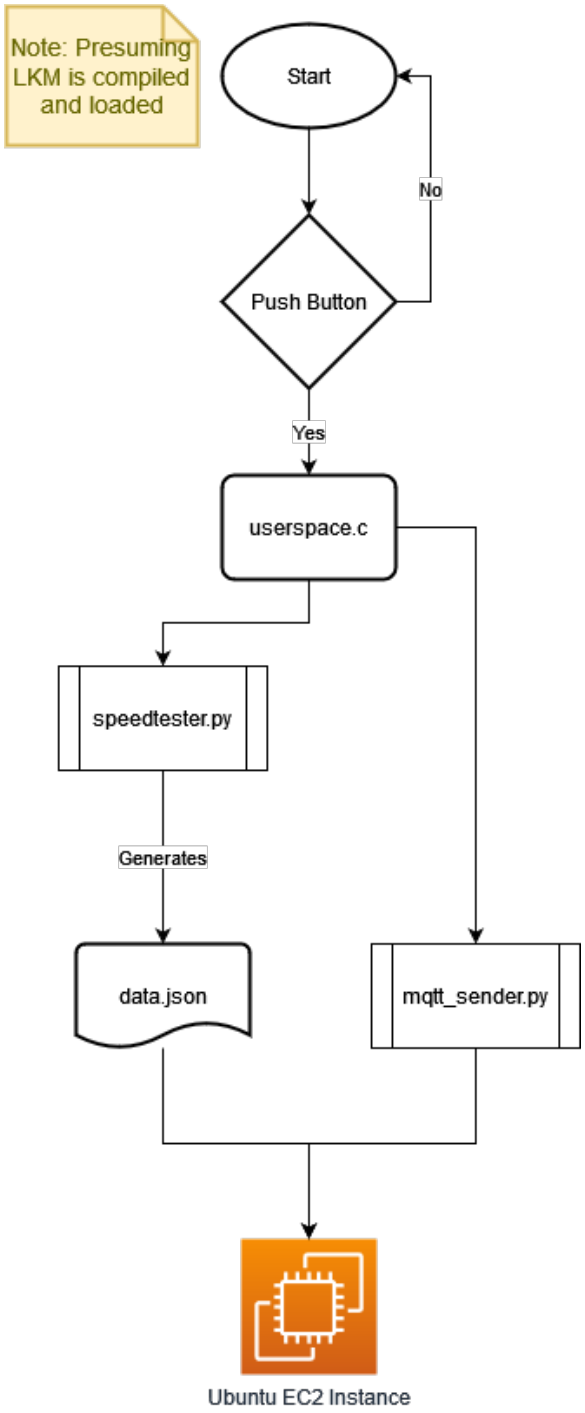

Figure 3: A flow chart of the processes that take place on the client (Raspberry Pi) side, from button press to sending data.

Finally, shown in Figure 4 is an image of the Pi wired up correctly to a button, which starts the speedtester process, and an LED, which comes on when the process has begun.

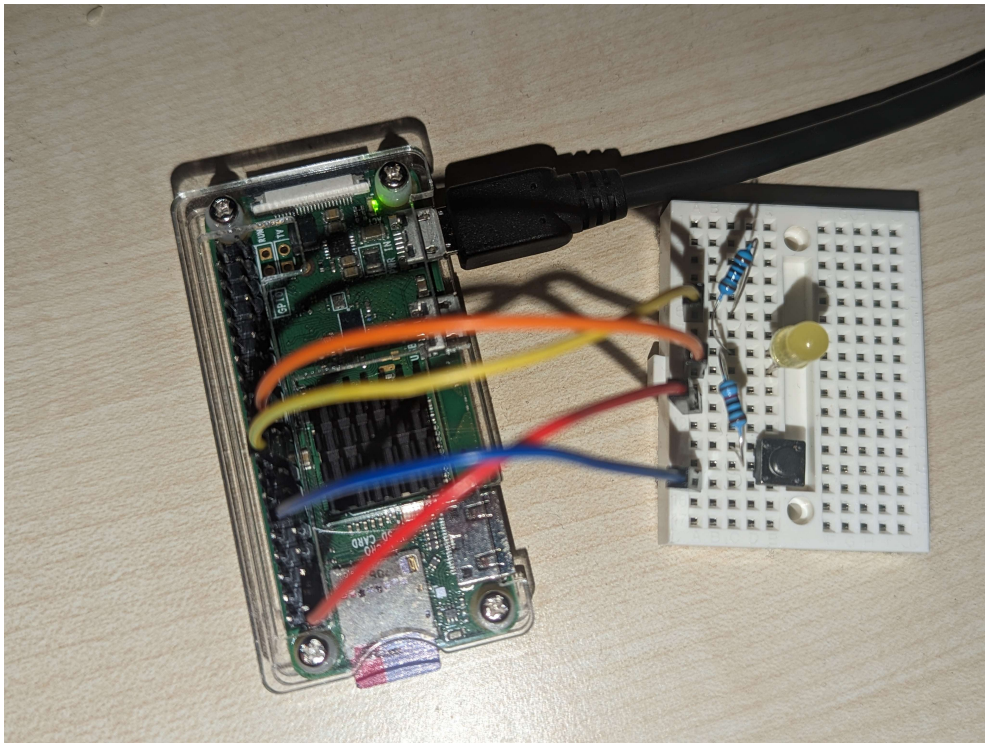

Figure 4: The Raspberry Pi in its final form, correctly wired up and plugged into a power source.

2.2 Cloud

As with the previous section, work began on the developer's laptop. The first piece of work developed was a webpage designed to display the results of said speedtest program. The JSON data returned from the program was first moved to the working directory for the webpage manually, however at a later stage when it was migrated to AWS, this was altered to be performed automatically. All cloud code is available in Appendix B

The developer used the Flask web framework (Pallets n.d.), due to its lightweight approach to web design which fits with the requirements of the site, its ease of use, and the fact its scripting language is Python, which the developer is familiar with. Figure 5 shows a screenshot of the website in an early stage of development, being hosted locally.

Figure 5: The webpage the developer created to display the results of the speed tester

The `data.json` file is available through a flask endpoint called `\data` which is opened on initialisation of the site. Clicking the button here runs a javascript function called `CreateTable()` which clears the contents of a pre-existing table, makes a `GET` request to the endpoint, parses the resulting JSON data, and displays it.

The project uses an AWS EC2 instance running Ubuntu, which serves the app. Setup followed Huiyeon's (2020) step-by-step guide.

Once the web server was up and running, work began on devising a method of sending the JSON object from the Pi to the EC2. The developer settled on using MQTT due to the fact it is the de facto standard for Internet of Things (IoT) messaging. It provides lightweight, efficient, and reliable communication over many networks, including unreliable ones (MQTT.org n.d.).

As shown in Figure 3, `mqtt_sender.py`, which is called by `userspace.c`, is responsible for sending the data. Its counterpart on the server-side is `mqtt_receiver.py` which subscribes to the same topic as the sender, uses the same authentication tokens, receives the text string sent by that script, and saves it to `data.json` on the server.

Finally, to ensure a secure connection, SSL/HTTPS should be set up. A domain was registered at http://ibr-cmp408.abertaycoursework.net/ and associated to the elastic IP address for the EC2 instance (`54.22.196.181`). To gain an SSL certificate the developer used the Certbot tool (EFF n.d.), the usage of which can be seen in Figure 6. Figure 7 shows the security group configuration of the EC2.

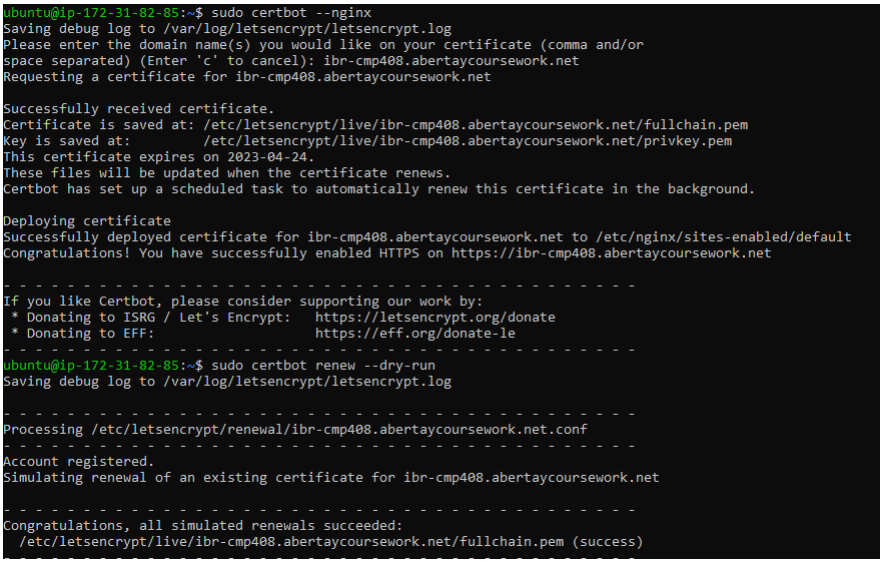

Figure 6: The Certbot tool used to gain an SSL certificate for the domain name associated with the speedtest web app

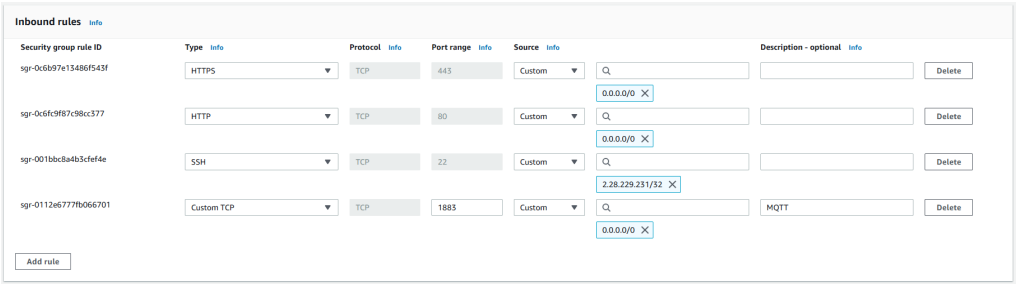

Figure 7: The list of inbound rules for the speedtester website, showing that SSH is only accessible via the developer's IP address

3 Conclusion

3.1 Discussion

This project makes use of several publicly available Python libraries to implement a network speedtester on a Raspberry Pi, write the data to a JSON library, and send it to an AWS EC2 instance using MQTT brokers. An LKM was implemented that registered a button click action and triggered the speed testing script before sending it to the server, and a custom Graphical User Interface (GUI) was developed to securely display the results of the speedtest. Additionally, an LED is used to inform the user that the speedtest is running when they do not have access to the Pi's CLI.

3.2 Future Work

The implementation of this project was generally successful, as mentioned above the majority of aims were met. Given further time and resources, however, some improvements could be made.

Firstly, the developer's preference was to use two LEDs, one for when the test is running and another for standby, due to a lack of resistors only one LED was able to be implemented. If another were to be added without a resistor the LED may draw more power than required, burning out the Pi (The Pi Hut 2015).

Additionally, the web interface could be improved with more statistics, possibly graphing capabilities to demonstrate how each network has changed over time, and filtering by only one field such as Up/Download, ping, etc.

References

EFF (n.d.). *Certbot Instructions*. URL: https://certbot.eff.org/instructions?ws=nginx&os=ubuntufocal (visited on Jan. 24, 2023).

Huiyeon, K. (June 9, 2020). *Step-by-Step Visual Guide on Deploying a Flask Application on AWS EC2*. Tech Front. URL: https://medium.com/techfront/step-by-step-visual-guide-on-deploying-a-flask-application-on-aws-ec2-8e3e8b82c4f7 (visited on Jan. 22, 2023).

Johannes 4GNU_Linux, director (Feb. 25, 2022). *Let's Code a Linux Driver - 15: Sending a Signal from Kernel to Userspace*. URL: https://www.youtube.com/watch?v=nt_z07t7qMc (visited on Jan. 23, 2023).

Martz, M. (Apr. 8, 2021). *Speedtest-Cli: Command Line Interface for Testing Internet Bandwidth Using Speedtest.Net*. Version 2.1.3. URL: https://github.com/sivel/speedtest-cli (visited on Dec. 16, 2022).

MQTT.org (n.d.). *MQTT - The Standard for IoT Messaging*. URL: https://mqtt.org/ (visited on Jan. 23, 2023).

Pallets (n.d.). *Welcome to Flask — Flask Documentation (2.2.x)*. URL: https://flask.palletsprojects.com/en/2.2.x/ (visited on Jan. 13, 2023).

Soren (Feb. 9, 2018). *Using a Push Button with Raspberry Pi GPIO — Raspberry Pi HQ*. URL: https://raspberrypihq.com/use-a-push-button-with-raspberry-pi-gpio/ (visited on Dec. 16, 2022).

The Pi Hut (June 11, 2015). *Turning on an LED with Your Raspberry Pi's GPIO Pins — The Pi Hut*. URL: https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins (visited on Jan. 23, 2023).

A Raspberry Pi Code

A.1 button.c LKM

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/gpio.h>
#include <linux/cdev.h>
#include <linux/interrupt.h>
#include <linux/fs.h>
#include <linux/sched/signal.h>
#include <linux/ioctl.h>
#include <linux/kernel.h>

// This kernel module will register an interrupt request (IRQ) on the falling
// (i.e., button press) of the specified GPIO pin. This will then trigger the

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Isaac Basque-Rice");
MODULE_DESCRIPTION("A simple Linux kernel module to detect button press on a Ras

// Ensure button is always on this GPIO pin, if it's not change this value
#define BUTTON_GPIO 17

#define BUTTON_MAJOR 64

// pin number / interrupt controller to which BUTTON_GPIO is mapped
unsigned int irq_number;

// Variables for speedtester registration
#define REGISTER_SAPP _IO('R', 'g')
static struct task_struct *task = NULL;

// Define for signal sending
#define SIGNR 44

// This calls an interrupt service routine when the interrupt is triggered
static irq_handler_t button_signal_handler(unsigned int irq, void *dev_id, struc
{
    struct siginfo info;
    printk("Interrupt was triggered and ISR was called\n");
```

```
        if(task != NULL) {
                memset(&info, 0, sizeof(info));
                info.si_signo = SIGNR;
                info.si_code = SI_QUEUE;

                /* Send the signal */
                if(send_sig_info(SIGNR, (struct kernel_siginfo *) &info, task) <
                        printk("Error sending signal\n");
        }
        return (irq_handler_t) IRQ_HANDLED;
}

// This is an IOCTL function for registering the speedtester to the LKM
static long int button_ioctl(struct file *file, unsigned cmd, unsigned long arg)
{
    if(cmd == REGISTER_SAPP) {
        task = get_current();
        printk("Userspace app with PID %d is registered \n", task->pid);
    }
    return 0;
}

// This function is called when the device file is opened
static int button_close(struct inode *device_file, struct file *instance) {
        if(task != NULL)
                task = NULL;
        return 0;
}

static struct file_operations fops = {
        .owner = THIS_MODULE,
        .release = button_close,
        .unlocked_ioctl = button_ioctl,
};


static int __init button_init(void)
{
    int err = 0;
```

```
    printk("Loading module... ");

// Request the GPIO
if (!gpio_is_valid(BUTTON_GPIO)) {
    printk(KERN_ERR "Invalid GPIO\n");
    return -ENODEV;
}

    if(gpio_direction_input(BUTTON_GPIO)) {
            printk("Error!\nCan not set GPIO 17 to input!\n");
            gpio_free(BUTTON_GPIO);
            return -1;
    }

if ((err = gpio_request(BUTTON_GPIO, "button_gpio"))) {
    printk(KERN_ERR "Failed to request GPIO\n");
    return err;
}

    gpio_set_debounce(BUTTON_GPIO, 300);

// set up interrupt
irq_number = gpio_to_irq(BUTTON_GPIO);

free_irq(irq_number, NULL);

// Enable IRQ on falling edge (button press)
if ((err = request_irq(irq_number, (irq_handler_t) button_signal_handler, IR
    printk(KERN_ERR "Failed to request IRQ\n");
    gpio_free(BUTTON_GPIO);
    return err;
}

    if(register_chrdev(BUTTON_MAJOR, "gpio_irq_signal", &fops) < 0)
{
            printk("Error!\n Can't register device Number!\n");
            gpio_free(BUTTON_GPIO);
            free_irq(irq_number, NULL);
    }

printk(KERN_INFO "Button module initialized\n");
```

```
        printk("GPIO pin is mapped to IRQ no.: %d\n", irq_number);

    return 0;
}

static void __exit button_exit(void)
{
    printk("Unloading module...");
    // Free the IRQ and GPIO
    free_irq(irq_number, NULL);
    gpio_free(BUTTON_GPIO);
        unregister_chrdev(BUTTON_MAJOR, "gpio_irq_signal");
    printk("Module unloaded\n");
}

module_init(button_init);
module_exit(button_exit);
```

A.2 Makefile

```
obj-m := button.o

KDIR := /usr/src/linux-headers-$(shell uname -r)

all:
    $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

A.3 userspace.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <signal.h>

#define SIGTX 44
```

```c
#define REGISTER_SAPP _IO('R', 'g')

/* This app serves as a userspace intermediary between the LKM running on the
Python is seemingly unable to interface between itself and the LKM so as a res
Partial credit is due to Johannes4Linux who provided boilerplate to allow for

void signalhandler(int sig) {
        FILE *fp;
    char path[1035];

    printf("Userspace: Signal Received!\n");
    printf("Userspace: Running speedtester.py\n\n");

    // Open the command for reading.
    fp = popen("python3 speedtester.py", "r");
    if (fp == NULL) {
        printf("Userspace: Failed to run speedtester\n" );
        exit(1);
    }

    // Read the output a line at a time - output it.
    while (fgets(path, sizeof(path)-1, fp) != NULL) {
        printf("%s", path);
    }

    // close
    pclose(fp);

    printf("Userspace: Running mqtt_sender.py\n\n");

    // Open the command for reading.
    fp = popen("python3 mqtt_sender.py", "r");
    if (fp == NULL) {
        printf("Userspace: Failed to run mqtt_sender\n" );
        exit(1);
    }

    // Read the output a line at a time - output it.
    while (fgets(path, sizeof(path)-1, fp) != NULL) {
        printf("%s", path);
    }
```

```
    // close
    pclose(fp);


    printf("Userspace: Wait for signal...\n");

    return 0;
}

int main() {
        int fd;
        signal(SIGTX, signalhandler);

        printf("PID: %d\n", getpid());

        // Open the device file
        fd = open("/dev/irq_signal", O_RDONLY);
        if(fd < 0) {
                perror("Userspace: Could not open device file");
                return -1;
        }

        // Register app to KM
        if(ioctl(fd, REGISTER_SAPP, NULL)) {
                perror("Userspace: Error registering app");
                close(fd);
                return -1;
        }


        // Wait for Signal
        printf("Userspace: Wait for signal...\n");
        while(1)
                sleep(1);

        return 0;
}
```

A.4 speedtester.py

```
#!/usr/bin/env python3

"""
Name: speedtester.py
Desc: a program to test the speed of a computer network
Auth: Isaac Basque-Rice
Date: 24/12/2022
"""

import time
import json
import os

import speedtest
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO

def send_to_json(now, ssid, down_speed, up_speed, ping, runtime):
    """ sends data from test function to a JSON object """

    ssid = str.strip(ssid) # removes \n

    # full list of information to send to JSON object
    dictionary = {
        "Start time": now,
        "Network name": ssid,
        "Down speed": down_speed,
        "Up speed": up_speed,
        "Ping": ping,
        "Run time": runtime
    }

    filename = "data.json"

    # loads the existing data from the file
    with open(filename, "r", encoding='ascii') as infile:
        data = []
        while True:
            try:
```

```
                obj = json.load(infile)
                data.extend(obj)
            except ValueError:
                # reached the end of the file
                break

    # append the new dictionary to the list of data
    data.append(dictionary)

    # dumps formats it properly (dump makes it a one-liner)
    json_object = json.dumps(data, indent = 4)

    # write the new list of data to the file
    with open(filename, "w", encoding='ascii') as outfile:
        outfile.write(json_object)

def test():
    """ Performs the speedtest and prints download, upload speeds and ping """

    speed = speedtest.Speedtest()

    # apparently I need this for ping? idk
    server_names = []
    speed.get_servers(server_names)

    down_speed = speed.download()
    up_speed = speed.upload()
    ping = speed.results.ping

    # will eventually find a neat way to convert bits to megabits or gigabits
    print("Download speed: " + str(int(down_speed)) + " b/s")
    print("Upload speed: " + str(int(up_speed)) + " b/s")
    print("Ping: " + str(int(ping)) + " ms")

    # casting to int to make the numbers less unwieldy lol
    return int(down_speed), int(up_speed), int(ping)


def main():
    """ The main function """
```

```
    start = time.time()

    # GPIO Settings
    pin_number = 9
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(pin_number,GPIO.OUT)

    # Turn on LED
    GPIO.output(pin_number, GPIO.LOW)

    now = time.ctime(start) # neater format (no epoch for human consumption)
    ssid = os.popen("iwgetid -r").read() # get SSID from OS shell

    print("SSID: " + ssid + "\nStart Time: " +  str(now))

    # run the function and grab all the values
    down_speed, up_speed, ping = test()

    end = time.time()

    print("Speedtest completed in " + str(int(end-start)) + " seconds")

    print("Writing to JSON object...")

    # send that to the JSON object
    send_to_json(now, ssid, down_speed, up_speed, ping, int(end-start))

    # Turn off LED
    GPIO.output(pin_number, GPIO.HIGH)

    print("Speedtest complete\n")



if __name__ == "__main__":
    main()
```

A.5 mqtt_sender.py

```
#!/usr/bin/env python3

"""
Name: mqtt_sender.py
Desc: an app to send the data.json file over MQTT to the EC2 instance
Auth: Isaac Basque-Rice
Date: 22/01/2023
"""

import paho.mqtt.client as mqtt
import json

# MQTT settings
broker_address = "52.22.196.181"
username = "user"
password = "password"
topic = "speedtest-data"

# Create MQTT client
client = mqtt.Client()

# Set username and password as auth tokens
client.username_pw_set(username, password)

# Connect to broker
client.connect(broker_address)

# Read the JSON file
with open("data.json", "r") as f:
    data = json.load(f)

# Convert the JSON data to a string
data_str = json.dumps(data)

# Send the JSON data to the server
client.publish(topic, data_str)

# Disconnect from broker
client.disconnect()
```

B Cloud Code

B.1 app.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():  # put application's code here
    return render_template("index.html")

@app.route('/data')
def data():
    with open('data.json', 'r') as file:
        data = file.read()
        file.close()
    return data

if __name__ == '__main__':
    app.run()
```

B.2 mqtt_receiver

```
#!/usr/bin/env python3

"""
Name: mqtt_receiver.py
Desc: an app to receive the data.json file over MQTT from the Raspberry Pi
Auth: Isaac Basque-Rice
Date 22/01/2023
"""

import paho.mqtt.client as mqtt
import json

# MQTT settings
broker_address = "3.91.200.59"
username = "user"
password = "password"
```

```
topic = "testdata"

# Create MQTT client
client = mqtt.Client()

# Set username and password as auth tokens
client.username_pw_set(username, password)

# Connect to broker
client.connect(broker_address)

# Subscribe to topic
client.subscribe(topic)

# Define callback function for incoming messages
def on_message(client, userdata, message):
    # convert the received message (string) to json format
    data = json.loads(message.payload)
    # save the json data to a file named "data.json"
    with open("data.json", "w") as f:
        json.dump(data, f)
    print("Data saved to disk")

# Set callback function
client.on_message = on_message

client.loop_forever()
```

B.3 index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Network Speedtester</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',filename='s
    </head>
<body>
```

```
    <h1>Isaac's Speed Tester</h1>
    <table>
        <thead>
            <th>Start Time</th>
            <th>Network Name</th>
            <th>Down Speed (b/s)</th>
            <th>Up Speed (b/s)</th>
            <th>Ping (ms)</th>
            <th>Run Time (s)</th>
        </thead>
        <tbody id="table">
        </tbody>
    </table>
    <input type="button" class="button" onclick="CreateTable()" value="Click Her

    <script>
        function CreateTable() {

            const table = document.getElementById("table");
            table.innerHTML = "";
            fetch("/data").then(res=>res.json()).then(res => {
                const items = res;  // Change this line to access the data dire
                items.forEach(item => {

                    const tr = document.createElement("tr");
                    ["Start time", "Network name", "Down speed", "Up speed", "Pi
                        const td = document.createElement("td");
                        td.innerText = item[label];
                        tr.appendChild(td);
                    });
                    table.appendChild(tr);

                });
            });
    }

    </script>
</body>
</html>
```

B.4 main.css

```
body {
    background-color: #282a36;
}

h1 {
    text-align: center;
    padding: 10px;
    color: #f8f8f2;
    font:32px helvetica, verdana, sans-serif;
}

table {
    border: solid 1px #282828;
    border-collapse: collapse;
    padding: 2px 3px;
    text-align: center;
    margin: 0 auto;
    width: 50%;
    background-color: #44475a;
    color: #f8f8f2;
}

th, td {
    font:14px helvetica;
    color: #ffffff;
}

th {
    font-weight: bold;
}

p, input {
    font:14px helvetica;
    color: #ffffff;
}

.button {
    margin: 0 auto;
    display: block;
```

```
    width: 50%;
    background-color: #6272a4;
    color: #f8f8f2;
    border: none;
    padding: 10px;
    text-align: center;
}

.button:hover {
    box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24),0 17px 50px 0 rgba(0,0,0,0.19);
}
```