Communications Subsystem Validation, Verification, and Testing (VV&T) Report

Timothy Japit[1]
North Carolina State University, Raleigh, NC, 27607, USA

This VV&T report discusses the experimental tests conducted to verify the performance of the communication subsystem of the RTDT CubeRover. The tests include the data transfer rate or bitrate test and the communications range test. The bitrate test was designed to fulfill DR 8.1.1 that states that the CubeRover shall communicate data at a rate of 40 kilobytes per second or greater, and the range test was designed to fulfill DR 8.1.2 that states that the CubeRover shall maintain communication at 100-meter radius. The bitrate test was a success with a resulting average 825 kilobytes per second from 10 data points at a distance of 100 meters from the mock lander. However, the range test did not fulfill the success criteria, the signal strength indicated at 100 meters away was recorded to be an average of -79.4 dBm over 10 data points; which is within the range of basic connectivity of -80 dBm or greater. The next steps would be to test sending diagnostic information from the rover to the lander and controlling rover movement from lander at a 100 meters distance.

Nomenclature

B	=	channel bandwidth
C	=	capacity/bitrate
D	=	size of data packet
d	=	distance between client device to router
dBm	=	decibel-milliwatts
GHz	=	gigahertz
IEEE	=	Institute of Electrical and Electronics Engineers
IP	=	Internet Protocol
kBps	=	kilobytes per second
L	=	average round trip latency
M	=	discrete levels in a digital signal
Mb/s	=	megabits per second
MHz	=	megahertz
RSSI	=	Received Signal Strength Indicator
SNR	=	signal to noise ratio
SSH	=	Secure Shell
TCP	=	Transmission Control Protocol
Wi-Fi	=	Wireless Fidelity

I. Introduction

The purpose of the tests are to verify that the RTDT CubeRover and its subsystems meet the design requirements that was specified, as well as testing to validate the CubeRover under the conditions that were specified by the project definition. Verification tests will be done by component before the system integration tests to simplify failure analysis and validation tests will be done after the completion of the verification tests. The scope of this document will cover the Communications subsystem, testing the performance of the 2.4 GHz 802.11n communications protocol with TCP/IP socket. The performance tests include a data transfer rate or bitrate test and a test on the range of the communications protocol. These are the verification tests that will confirm the functionality of the built-in Wi-Fi capabilities within the Raspberry Pi 3 Model B+ processor onboard the rover. The Communications subsystem is responsible for communicating payload and telemetry data from the rover to the ground station and commands from

[1] Undergraduate Student, Mechanical and Aerospace Engineering

the ground station to the rover. The rover is equipped with Raspberry Pi cameras for detecting obstacles, temperature sensors for internal temperature readings, accelerometers to keep track of its orientation, and many other data acquiring tools that feed their information to the onboard computer. The information collected will then be transmitted via the 2.4 GHz frequency to the lander that will act as a relay to communicate with NASA's ground station. Conversely, the commands for the rover will be sent to NASA ground station, transmitted to the lander through the S-band, and forwarded to the rover via the 2.4 GHz frequency.

The functional requirement driving the verification tests on this document is **FR 8**. This functional requirement states that the CubeRover shall communicate with the ground station through the lander. The implication is that the rover would have to use a communications protocol that is supported by the lander. The communications protocol is specified in **DR 8.1**, declaring that the rover shall use 2.4 GHz 802.11n Wi-Fi radio using TCP/IP to communicate with the lander. This protocol was chosen because of its prevalence and its success history by Astrobotic who used this protocol for their rover with the Peregrine lander [1]. Derived from this requirement are design requirements **DR 8.1.1** and **8.1.2**, which correspond to the specifications of the data rate and communications range, respectively. The data rate shall be greater than or equal to 40 kilobytes per second (kBps) to enable near instantaneous commands to be sent in case of errors or navigational problems where immediate action is required, and the communications protocol shall have a strong enough signal to maintain connection within a 100-meter radius of the lander. By verifying these design requirements, the Communications subsystem of the CubeRover shall communicate with the ground station through the lander.

A. Data Rate/Bitrate

The data transfer rate or bitrate of a radio communications protocol of a certain frequency are limited by a number of factors such as, the bandwidth that is available for that frequency band, the number of levels in the digital signal, and the level of noise or disturbance that is on a channel [2][3]. The Nyquist Bitrate defines the theoretical maximum bitrate in a noiseless channel,

$$C_{noiseless} = 2B\log_2 M \tag{1}$$

where $C_{noiseless}$ is the capacity or the bitrate of the noiseless channel in bits per second (bps), B is the bandwidth of the channel in hertz (Hz), and M is a dimensionless unit that represents the number of discrete levels in the digital signal used to represent the data being communicated [2][3]. This implies that the theoretical maximum bitrate of a noiseless channel will increase proportionally as the bandwidth of the channel increases or increase logarithmically as the number of levels in the signal increase. However, using the Nyquist theorem to calculate the theoretical maximum in a noisy channel and increasing the levels in the signal in hopes of increasing the capacity could cause problems.

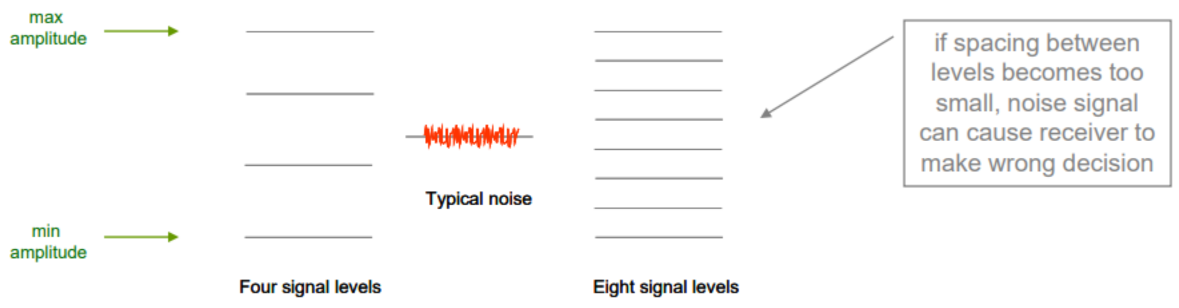

max amplitude
min amplitude
Typical noise
Four signal levels
Eight signal levels
if spacing between levels becomes too small, noise signal can cause receiver to make wrong decision

Figure I-1 Discrete Signal Level Spacing [2]

Calculations with the Nyquist theorem are usually paired with the Shannon Law which defines the maximum theoretical bitrate limit over a channel with Gaussian distributed noise [2].

$$C_{noisy} = B\log_2(1 + SNR) \tag{2}$$

where C_{noisy} is the capacity or bitrate of the noisy channel in bits per second (bps), B is the bandwidth of the channel in Hz, and *SNR* is the unitless signal to noise ratio compared in watts. The number of discrete levels in the message signal is not included in Eq. (2) implying no direct correlation between the levels and capacity. For a channel of a given bandwidth, the Nyquist theorem is used to determine the number of signal levels that should be used for the channel and Shannon Law is used to determine the maximum data rate [2][3][4]. The conversion between the signal and noise measured in dBm to milliwatts can be shown from the equation below,

$$S_{mW} = 10^{\frac{S_{dBm}}{10}} \tag{3}$$

where S_{mW} is the signal/noise strength in milliwatts and S_{dBm} is the signal/noise strength in dBm. Then once the conversion to milliwatts has been done, the signal to noise ratio can be calculated [4][5]. Otherwise, a simple method to calculate the bitrate is to determine a data size to send and then by using the *ping* command in the command line of a computer to obtain the latency of a round trip communication,

$$C = \frac{2D}{L * 10^{-3}} \tag{4}$$

where C is the data rate in bytes per second (Bps), D is the size of the data sent each trip in bytes, and L is the average round trip latency reported from the *ping* command [6].

B. Range

The range for a communications system depends on the frequency band, the protocol that is used, as well as the obstructions that is in between the two devices or an access point (AP) and a client device. Received Signal Strength Indicator (RSSI) is the measure of how strong the signal received by the client device is at some distance from the AP [7]. The Path Loss or Path Attenuation equation is used to calculate the RSSI of a device with an unobstructed line of sight to the AP,

$$RSSI(d) = RSSI(d_0) - 10n_p \log_{10}\left(\frac{d}{d_0}\right) \tag{5}$$

where *RSSI* is measured in decibel-milliwatts (dBm), d is the distance between the device to the AP in meters, and n_p is the attenuation factor in dBm. The calculation is done by assuming a baseline value or measuring the signal strength from a reference distance d_0 [8][9][10]. However, the purpose of the verification test is to determine the maximum range that the rover can drive away from the lander while maintaining a strong enough signal strength to communicate data. Therefore, a range of acceptable signal strengths that are required for usable connectivity is required and presented as follows,

Signal Strength (dBm)	Description	Required for
-30	Maximum achievable, client is a few feet from the access point	N/A
-67	Minimum for applications requiring reliable and timely packet delivery	VoIP/VoWi-Fi, video streaming
-70	Minimum for reliable packet delivery	Email, web browsing
-80	Minimum for basic connectivity	N/A
-90	Unusable	N/A

Table I-1 Acceptable Signal Strengths [9]

Eq. (5) was then rearranged to obtain a distance measure from a reference distance and RSSI, and acceptable RSSI,

$$d = d_0 * 10^{[RSSI(d_0) - RSSI(d)]/10n_p} \tag{6}$$

by measuring a reference RSSI with a distance of 1 meter away from an access point or signal source, referencing Table I-1 for signal strengths, and applying Eq. (6), it is possible to calculate the theoretical distance that the rover has to travel away from the lander to maintain the desired signal strength.

To properly verify the performance of the communications protocol, some baseline numbers were obtained through research to serve as reference. The Institute of Electrical and Electronics Engineers (IEEE) specifies that the 2.4 GHz 802.11n should have a bandwidth of 20 MHz for each channel. The theoretical maximum data rate for this frequency band is up to 288.8 Megabits per second (Mbps), it has experimental indoor maximum range of around 70-150 meters and maximum outdoor range of up to 250 meters [11][12][13][14].

II. Methodology

There are two experimental tests that will be conducted to verify the performance of the Communications subsystem of the RTDT CubeRover. These tests are the data transfer rate or bitrate test and the communications range test. The bitrate test will measure how many bits of data can be transferred between the rover and lander every second, while the range test will confirm how far the rover can drive away from the lander before it loses connection. The set-up for both tests involve connecting the Raspberry Pi onboard the rover to a desktop computer that serves as a mock lander. All of the tests have been modified due to delay in construction of the rover. The main difference from the original tests that were designed was that the Raspberry Pi will be tested without the rover body present. Modifications to the tests will be described in more detail in each section of the tests.

A. Data Rate/Bitrate Test

1. Test Design

A Windows desktop computer or a laptop and a router with a known IP address will be set as a stationary mock lander. It should have a wireless local area network capability to communicate at 2.4 GHz 802.11n with TCP/IP socket. The CubeRover will be drive to a fixed distance of 100 meters from the mock lander and test 5 points in the perimeter of that distance. The mock lander shall ping the rover 3 times at each point from the command line, each ping consisting of 4 packets of 20,000-byte (20 kilobytes) data. Once entered, the mock lander will be sending 20 kilobytes and receive 20 kilobytes worth of data, adding up to 40 kilobytes. The minimum, maximum, and average round trip times will be recorded in milliseconds and averaged over the 5 points in which the data is taken. There will be a total of 60 data points averaged of lander-rover-lander transmission over this experiment. Once the round-trip times have been averaged, we divide the 40 kilobytes with the average round-trip time to get the bitrate. The modification to this test was that the Pi will be tested on its own without the rover body, and the 100 meters distance required was removed and the Pi will only be tested with a router that is less than 1 meters. The 100 meters test was included in the range test instead. Code for ping.c is appended to the Appendix.

2. Procedure

1. Power on the computer/laptop that will serve as the mock lander
2. Confirm networking capability (2.4 GHz 802.11n) by checking in Device Manager > Network Adapter
3. Verify the network in the Pi on the command line by typing: **iwconfig**
4. Record the IP of the Pi for testing with: **ifconfig**
5. Open command line on the mock lander with the files: **ping.c** and **makefile**, in the same folder
6. Type **make > ping**
7. The script will prompt for the command to use, type: **ping -l 20000**
8. Prompt for the Pi IP address, type in the given IP from **ifconfig** and enter
9. Prompt to save the results to file, type **y** to save or **n** to not save to file
10. If test results will be saved to a file, it will prompt for the name of the file, enter the name
11. Lastly, it will prompt for the number of tests to run
12. Record the minimum, maximum, and average round trip times if not saved to file, otherwise the given file should have the test results of the minimum, maximum, and average round trip times
13. Average the round-trip times from all data points
14. Calculate the data rate using Eq. (4) and convert it to kilobytes per second

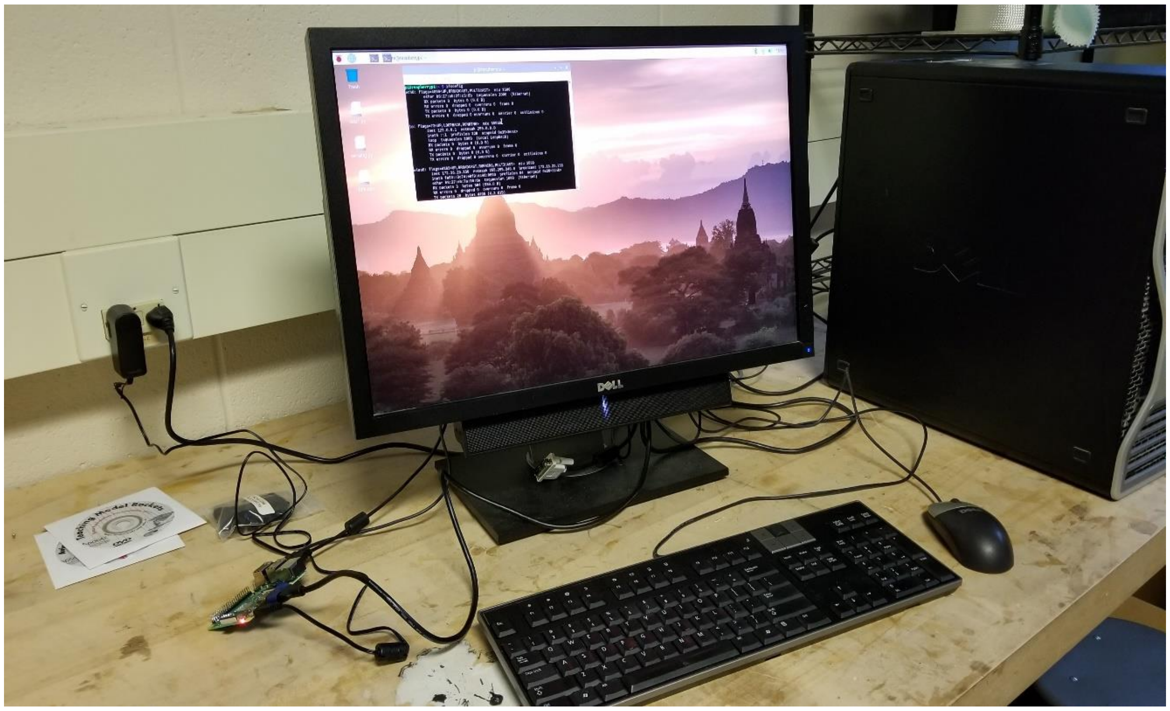
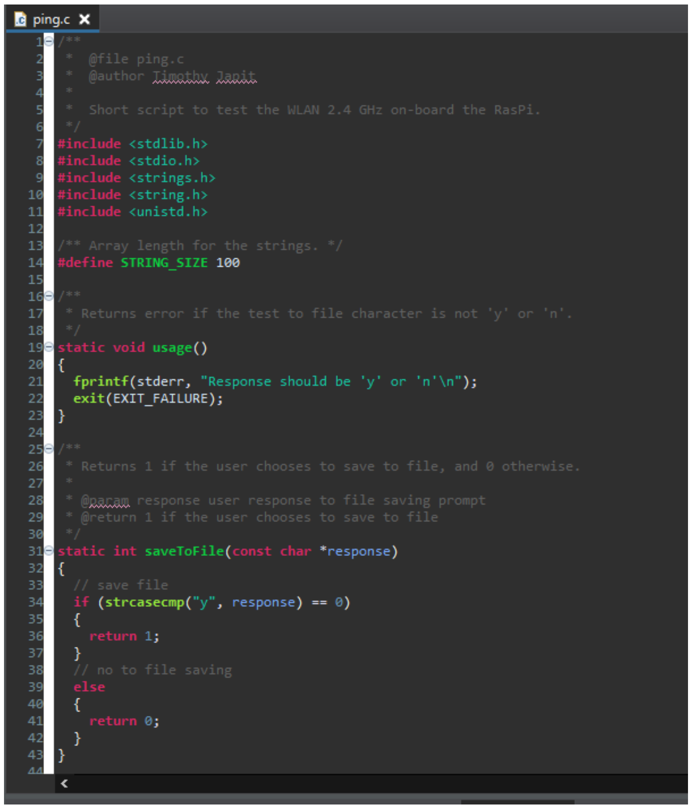
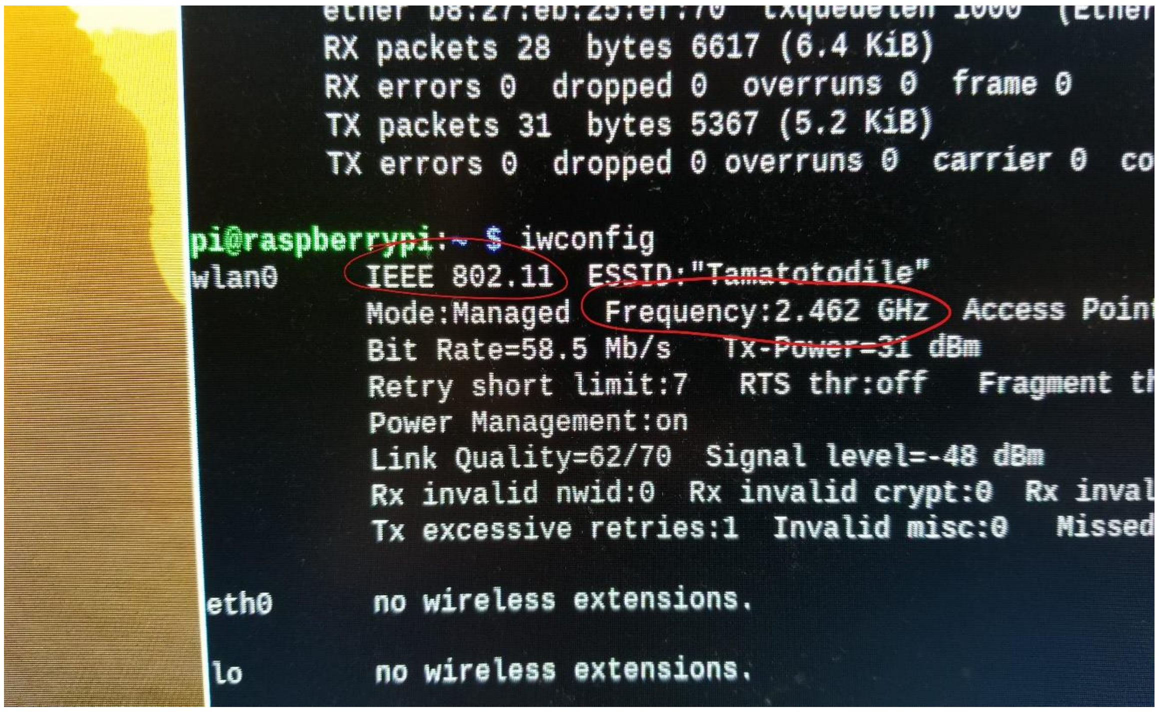


Figure II-1 Test setup



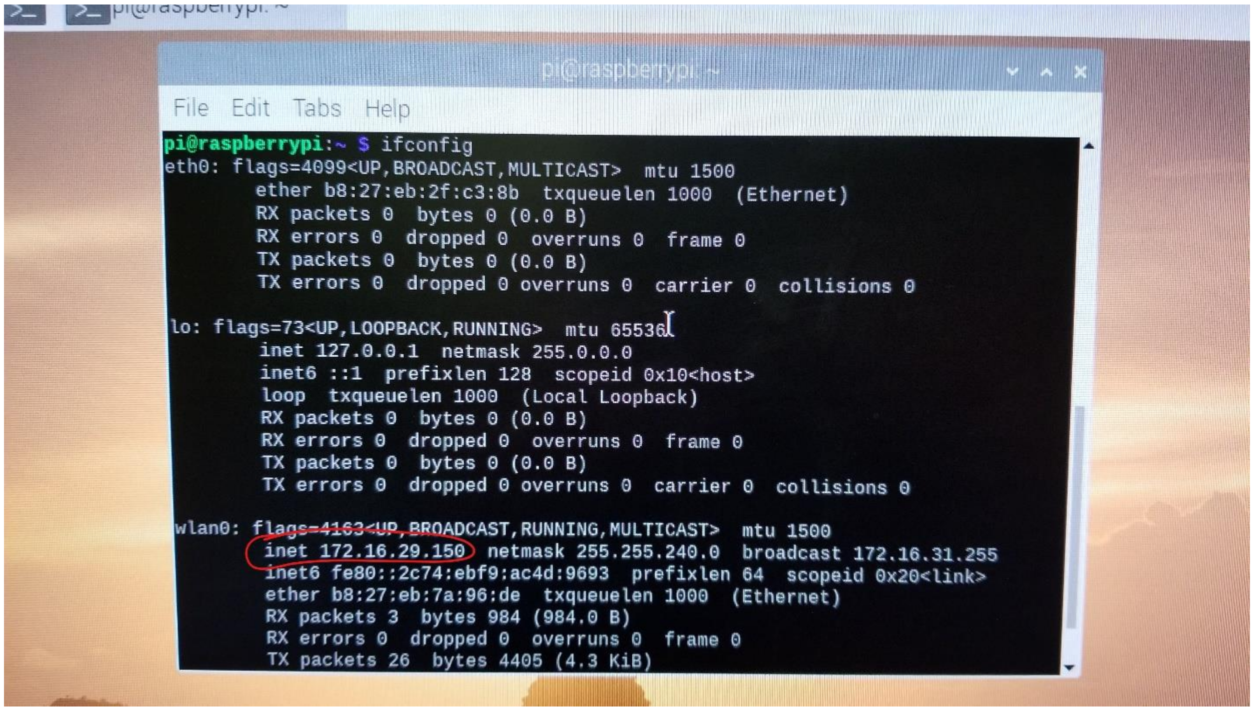
```c
/**
 *  @file ping.c
 *  @author Timothy Japit
 *
 *  Short script to test the WLAN 2.4 GHz on-board the RasPi.
 */
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <unistd.h>

/** Array length for the strings. */
#define STRING_SIZE 100

/**
 * Returns error if the test to file character is not 'y' or 'n'.
 */
static void usage()
{
  fprintf(stderr, "Response should be 'y' or 'n'\n");
  exit(EXIT_FAILURE);
}

/**
 * Returns 1 if the user chooses to save to file, and 0 otherwise.
 *
 * @param response user response to file saving prompt
 * @return 1 if the user chooses to save to file
 */
static int saveToFile(const char *response)
{
  // save file
  if (strcasecmp("y", response) == 0)
  {
    return 1;
  }
  // no to file saving
  else
  {
    return 0;
  }
}
```

Figure II-2 Snippet of **ping.c**



```
        ether b8:27:eb:25:ef:70  txqueuelen 1000  (Ether
        RX packets 28  bytes 6617 (6.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 31  bytes 5367 (5.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  co

pi@raspberrypi:~ $ iwconfig
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.462 GHz  Access Poin
          Bit Rate=58.5 Mb/s   Tx-Power=31 dBm
          Retry short limit:7   RTS thr:off   Fragment th
          Power Management:on
          Link Quality=62/70  Signal level=-48 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx inval
          Tx excessive retries:1  Invalid misc:0   Missed

eth0      no wireless extensions.

lo        no wireless extensions.
```

Figure II-3 Verification of 2.4 GHz and 802.11n

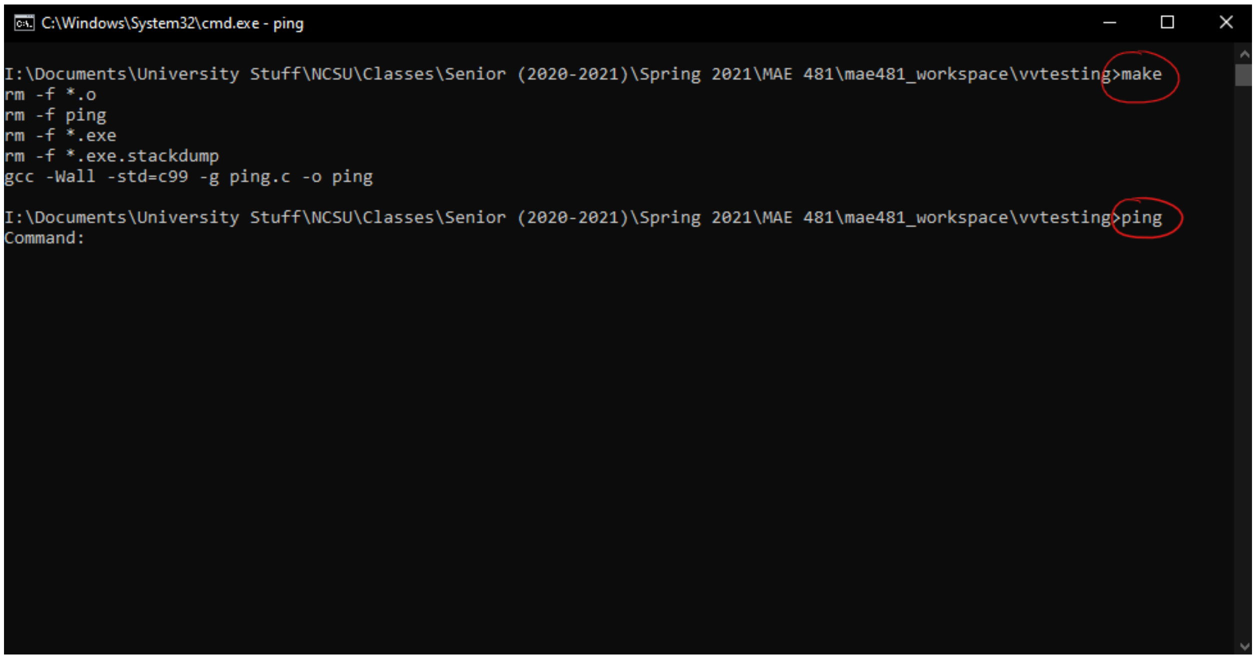


pi@raspberrypi: ~

File Edit Tabs Help

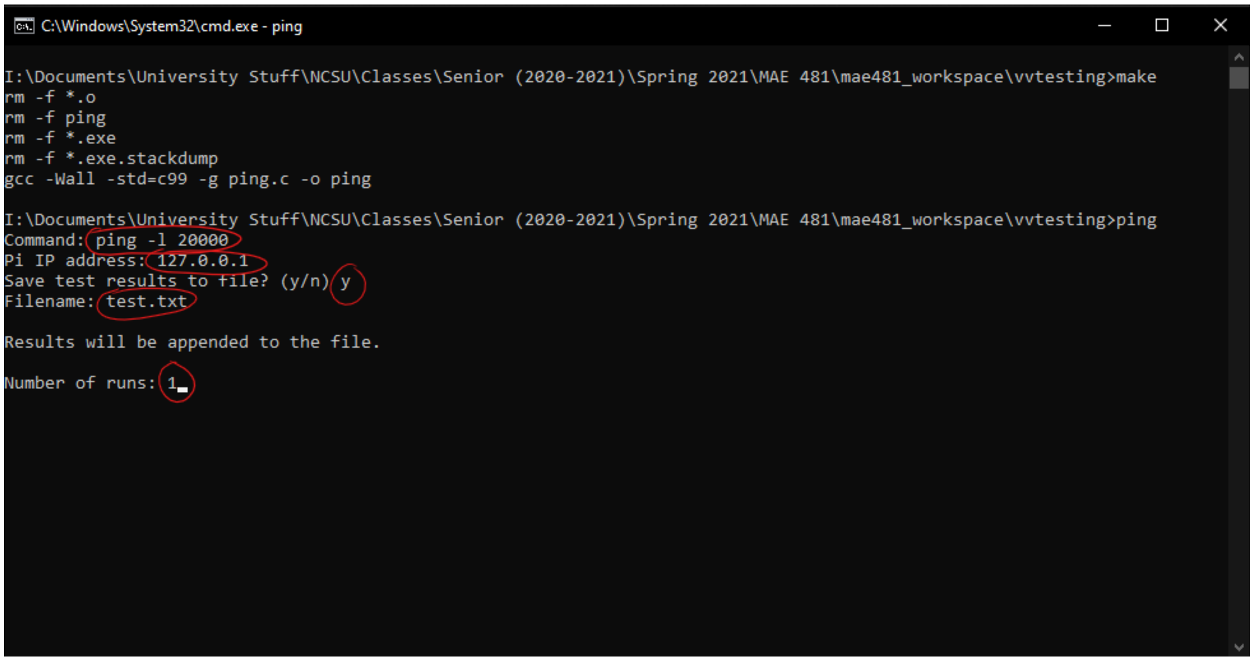
```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b8:27:eb:2f:c3:8b  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.29.150  netmask 255.255.240.0  broadcast 172.16.31.255
        inet6 fe80::2c74:ebf9:ac4d:9693  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:7a:96:de  txqueuelen 1000  (Ethernet)
        RX packets 3  bytes 984 (984.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26  bytes 4405 (4.3 KiB)
```

Figure II-4 Raspberry Pi IP address for testing



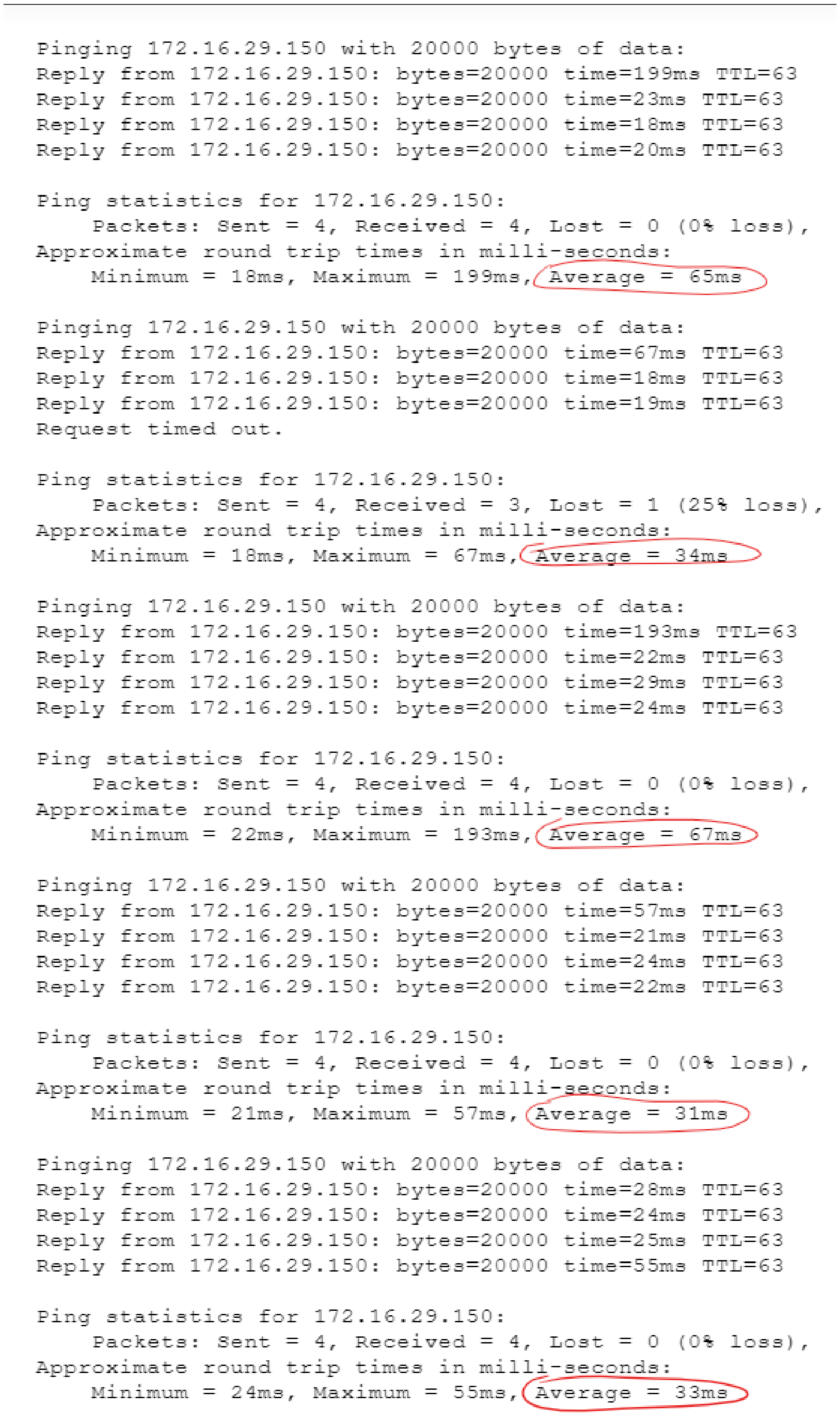
```
I:\Documents\University Stuff\NCSU\Classes\Senior (2020-2021)\Spring 2021\MAE 481\mae481_workspace\vvtesting>make
rm -f *.o
rm -f ping
rm -f *.exe
rm -f *.exe.stackdump
gcc -Wall -std=c99 -g ping.c -o ping

I:\Documents\University Stuff\NCSU\Classes\Senior (2020-2021)\Spring 2021\MAE 481\mae481_workspace\vvtesting>ping
Command:
```

Figure II-5 **make** and **ping**



```
I:\Documents\University Stuff\NCSU\Classes\Senior (2020-2021)\Spring 2021\MAE 481\mae481_workspace\vvtesting>make
rm -f *.o
rm -f ping
rm -f *.exe
rm -f *.exe.stackdump
gcc -Wall -std=c99 -g ping.c -o ping

I:\Documents\University Stuff\NCSU\Classes\Senior (2020-2021)\Spring 2021\MAE 481\mae481_workspace\vvtesting>ping
Command: ping -l 20000
Pi IP address: 127.0.0.1
Save test results to file? (y/n) y
Filename: test.txt

Results will be appended to the file.

Number of runs: 1
```

Figure II-6 Command and prompts from ping.c



```
Pinging 172.16.29.150 with 20000 bytes of data:
Reply from 172.16.29.150: bytes=20000 time=199ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=23ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=18ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=20ms TTL=63

Ping statistics for 172.16.29.150:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 18ms, Maximum = 199ms, Average = 65ms

Pinging 172.16.29.150 with 20000 bytes of data:
Reply from 172.16.29.150: bytes=20000 time=67ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=18ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=19ms TTL=63
Request timed out.

Ping statistics for 172.16.29.150:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 18ms, Maximum = 67ms, Average = 34ms

Pinging 172.16.29.150 with 20000 bytes of data:
Reply from 172.16.29.150: bytes=20000 time=193ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=22ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=29ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=24ms TTL=63

Ping statistics for 172.16.29.150:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 22ms, Maximum = 193ms, Average = 67ms

Pinging 172.16.29.150 with 20000 bytes of data:
Reply from 172.16.29.150: bytes=20000 time=57ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=21ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=24ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=22ms TTL=63

Ping statistics for 172.16.29.150:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 21ms, Maximum = 57ms, Average = 31ms

Pinging 172.16.29.150 with 20000 bytes of data:
Reply from 172.16.29.150: bytes=20000 time=28ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=24ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=25ms TTL=63
Reply from 172.16.29.150: bytes=20000 time=55ms TTL=63

Ping statistics for 172.16.29.150:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 24ms, Maximum = 55ms, Average = 33ms
```

Figure II-7 Example test result

3. Required Resources

- Windows computer/laptop with 2.4 GHz
- Router set to 802.11n
- GCC (C compiler) and GDB (GNU debugging tools for C)
- Cygwin for compiling C programs in Windows
- Monitor and HDMI/Display Port cables
- Mouse and keyboard
- Power outlet
- Table and chair for test administrator

4. Expected Results

The resulting average round trip times is expected to be less than or equal to 1000 milliseconds. A latency equal to that number will return a data rate of 40 kilobytes per second as specified in **DR 8.1.1**.

5. Success Criteria

Three levels of success and the failure condition are defined as follows,

- **Exceptional:** Average round trip times is less than or equal to 100 milliseconds, implying a data rate of greater than 400 kilobytes per second.
- **Extensive:** Average round trip times is less than or equal to 500 milliseconds, implying a data rate of greater than or equal to 80 kilobytes per second.
- **Acceptable:** Average round trip times is less than or equal to 1000 milliseconds, implying a data rate of greater than or equal to 40 kilobytes per second.
- **Failure:** Average round trip times is greater than 1000 milliseconds, implying a data rate of less than 40 kilobytes per second.

B. Range Test

1. Test Design

The test design for the range test is similar to the bitrate test. A desktop computer or a laptop and a router with a known IP address will be set as a stationary mock lander. It should have a wireless local area network capability to communicate at 2.4 GHz 802.11n with TCP/IP socket. The difference is that the CubeRover will have to drive to varying distances away from the mock lander. The rover will function as a server in this test such that the mock lander can pose as the client to connect and request data. Once connection is established through TCP/IP with a secret port number, the rover can drive away from the mock lander. This test was modified such that the Pi instead of the rover was connected to a 5V, 2A power bank and carried to 3 different locations away from the mock lander, each incremented by 25 meters. There were 10 tests on 50 meters, 75 meters, and 100 meters away from the laptop router combination that served as the mock lander. The signal strength (RSSI) and the bitrate were recorded on each of these locations. The Pi has been given a static IP address for a local area network of **192.168.1.52** with a hardcoded port number. Code for server.py and client.py are appended to the Appendix.

2. Procedure

1. Power on the computer/laptop that will serve as the mock lander
2. Confirm networking capability (2.4 GHz 802.11n) by checking in Device Manager > Network Adapter
3. Remote in with SSH to the Raspberry Pi by typing: **ssh pi@192.168.1.52**, into the mock lander command line
4. Change directory to the location of the test script: **cd /home/pi/MAE481/comms**
5. Start the server script by typing: **sudo python3 server.py**
6. Connect to the server with **client.py** by executing: **python client.py,** on the command line of the mock lander
7. The script will prompt tester to input IP address, input the IP **192.168.1.52**
8. Enter in the range test test-code: **r**
9. Once received, the server will respond by asking the number of tests to run, type in: **10**
10. The delay between tests in seconds (it accepts floating point number), any number is desirable
11. Type in the filename for the results to be saved to
12. Repeat steps 8-11 while moving the Pi to the 50-meter, 75-meter, and 100-meter mark locations while typing in the same parameters but with a different filename
13. Record the minimum, maximum, and average dBm from the test results

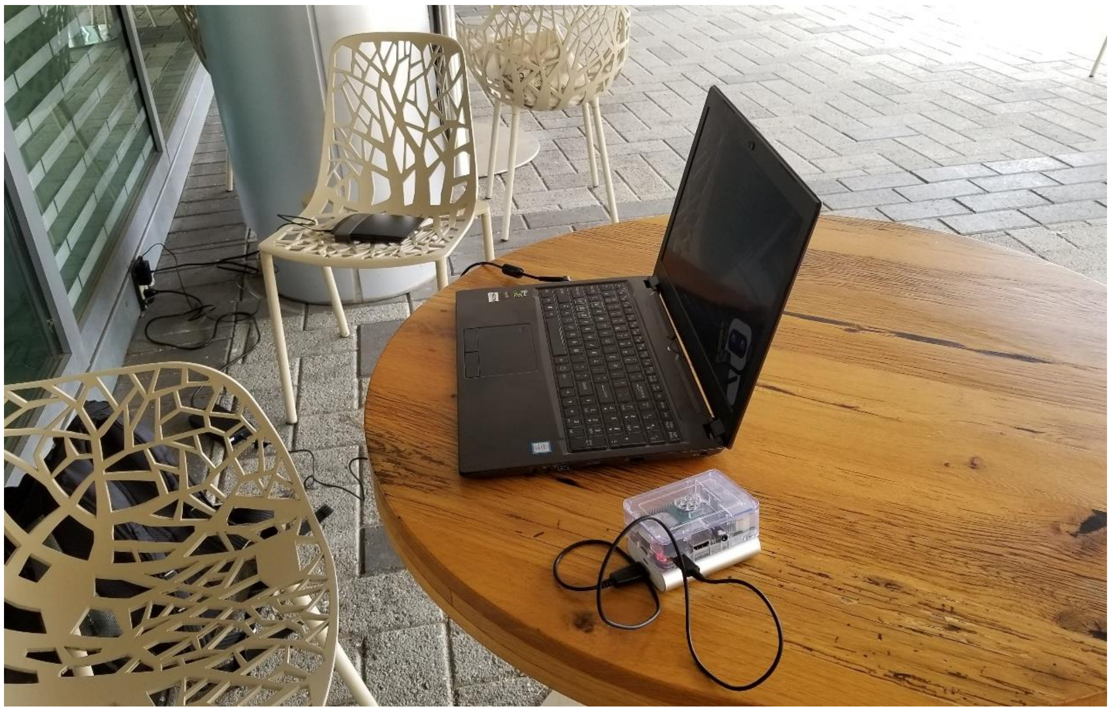
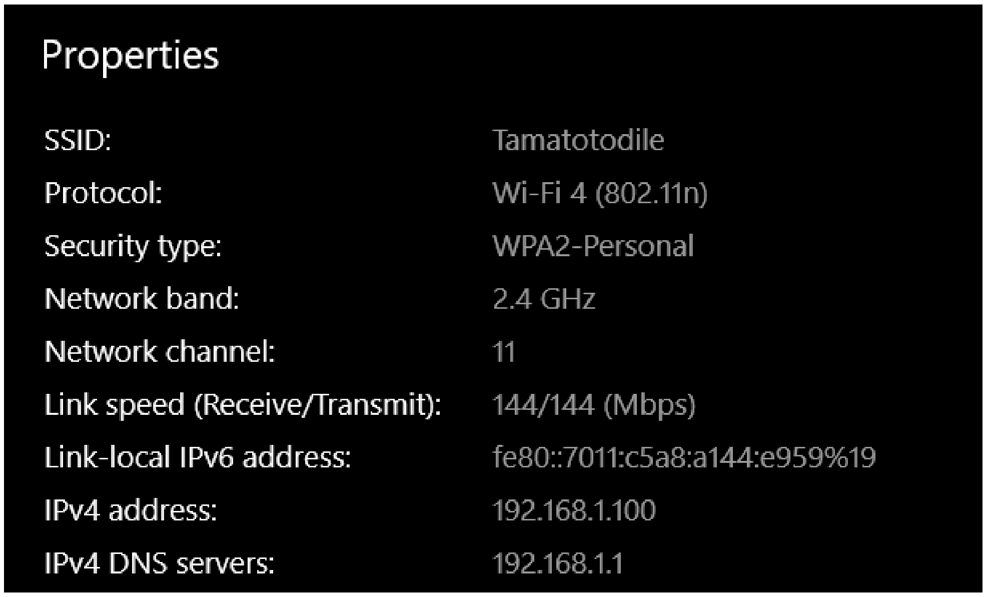
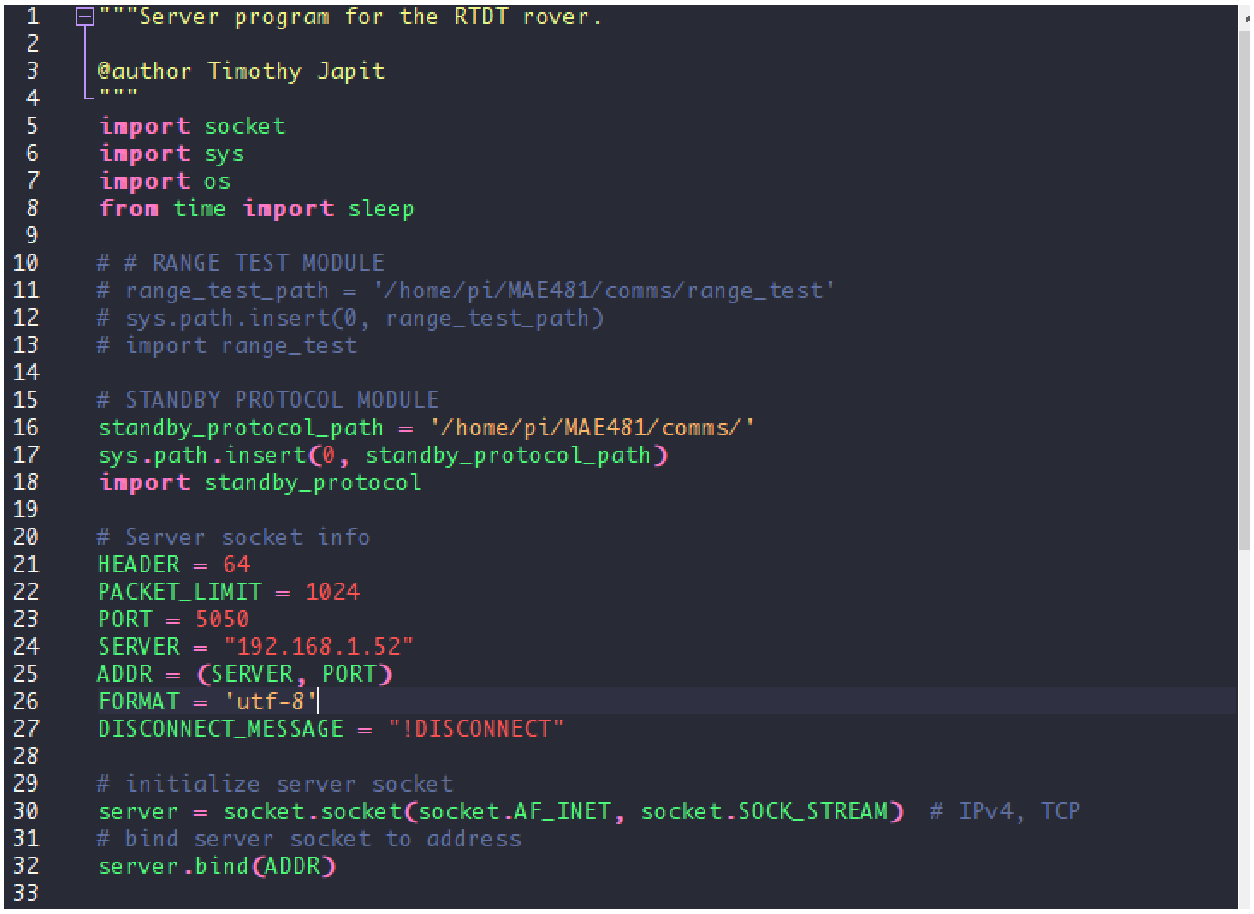


Figure II-8 Range test setup


Properties

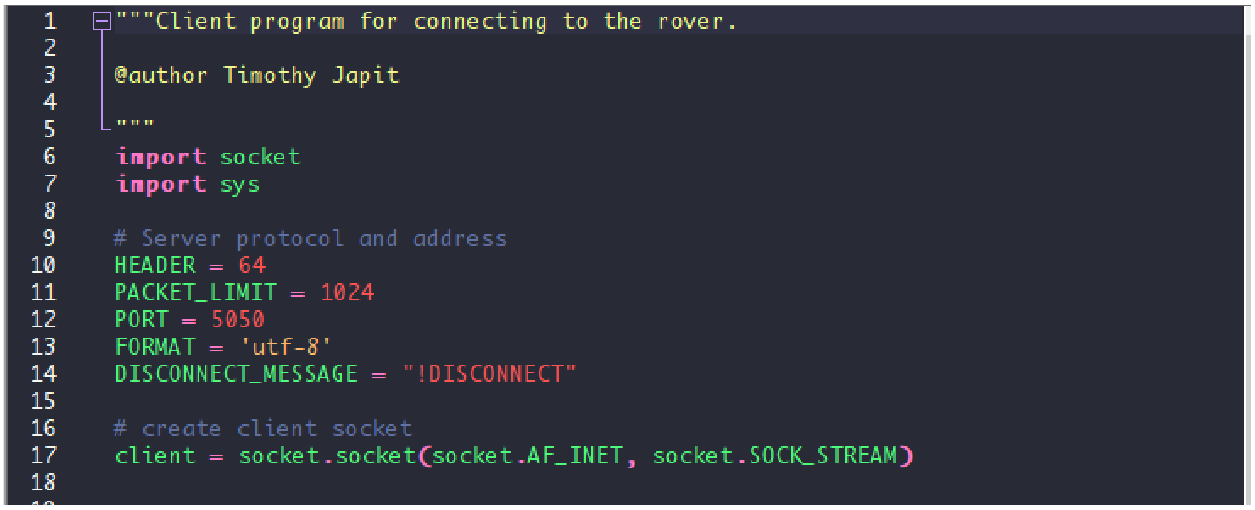
SSID:	Tamatotodile
Protocol:	Wi-Fi 4 (802.11n)
Security type:	WPA2-Personal
Network band:	2.4 GHz
Network channel:	11
Link speed (Receive/Transmit):	144/144 (Mbps)
Link-local IPv6 address:	fe80::7011:c5a8:a144:e959%19
IPv4 address:	192.168.1.100
IPv4 DNS servers:	192.168.1.1

Figure II-9 Verification of 2.4 GHz, 802.11n



```python
"""Server program for the RTDT rover.

@author Timothy Japit
"""
import socket
import sys
import os
from time import sleep

# # RANGE TEST MODULE
# range_test_path = '/home/pi/MAE481/comms/range_test'
# sys.path.insert(0, range_test_path)
# import range_test

# STANDBY PROTOCOL MODULE
standby_protocol_path = '/home/pi/MAE481/comms/'
sys.path.insert(0, standby_protocol_path)
import standby_protocol

# Server socket info
HEADER = 64
PACKET_LIMIT = 1024
PORT = 5050
SERVER = "192.168.1.52"
ADDR = (SERVER, PORT)
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

# initialize server socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # IPv4, TCP
# bind server socket to address
server.bind(ADDR)

```

Figure II-10 Snippet of server.py



```python
"""Client program for connecting to the rover.

@author Timothy Japit

"""
import socket
import sys

# Server protocol and address
HEADER = 64
PACKET_LIMIT = 1024
PORT = 5050
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

# create client socket
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

```

Figure II-11 Snippet of client.py

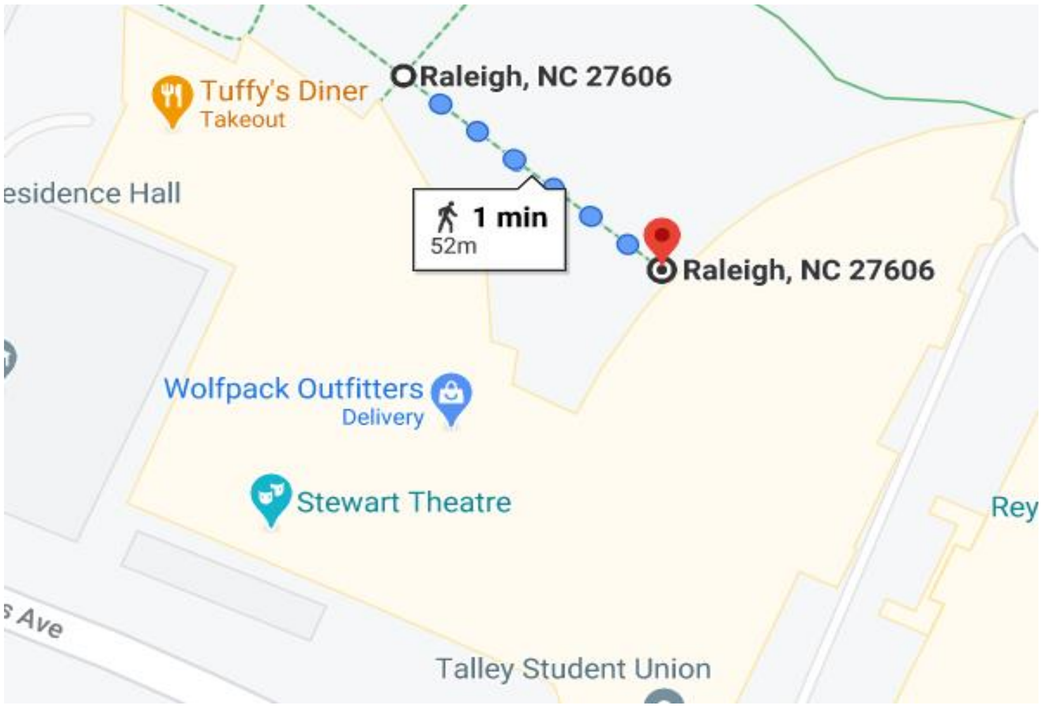


Figure II-12 Range test location 1

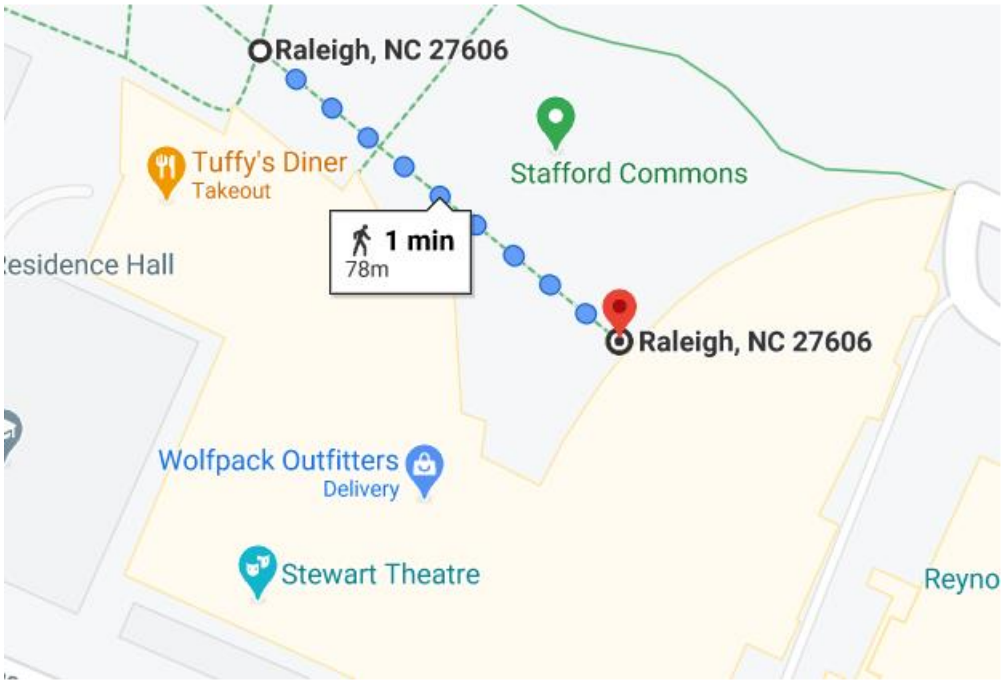


Figure II-13 Range test location 2

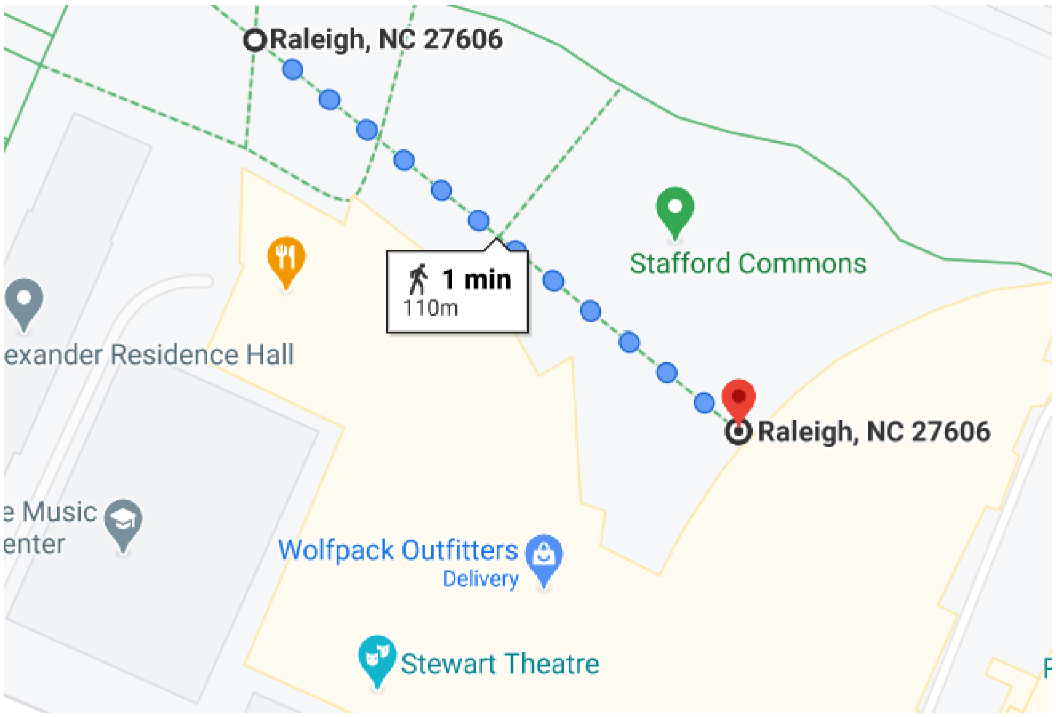


Figure II-14 Range test location 3

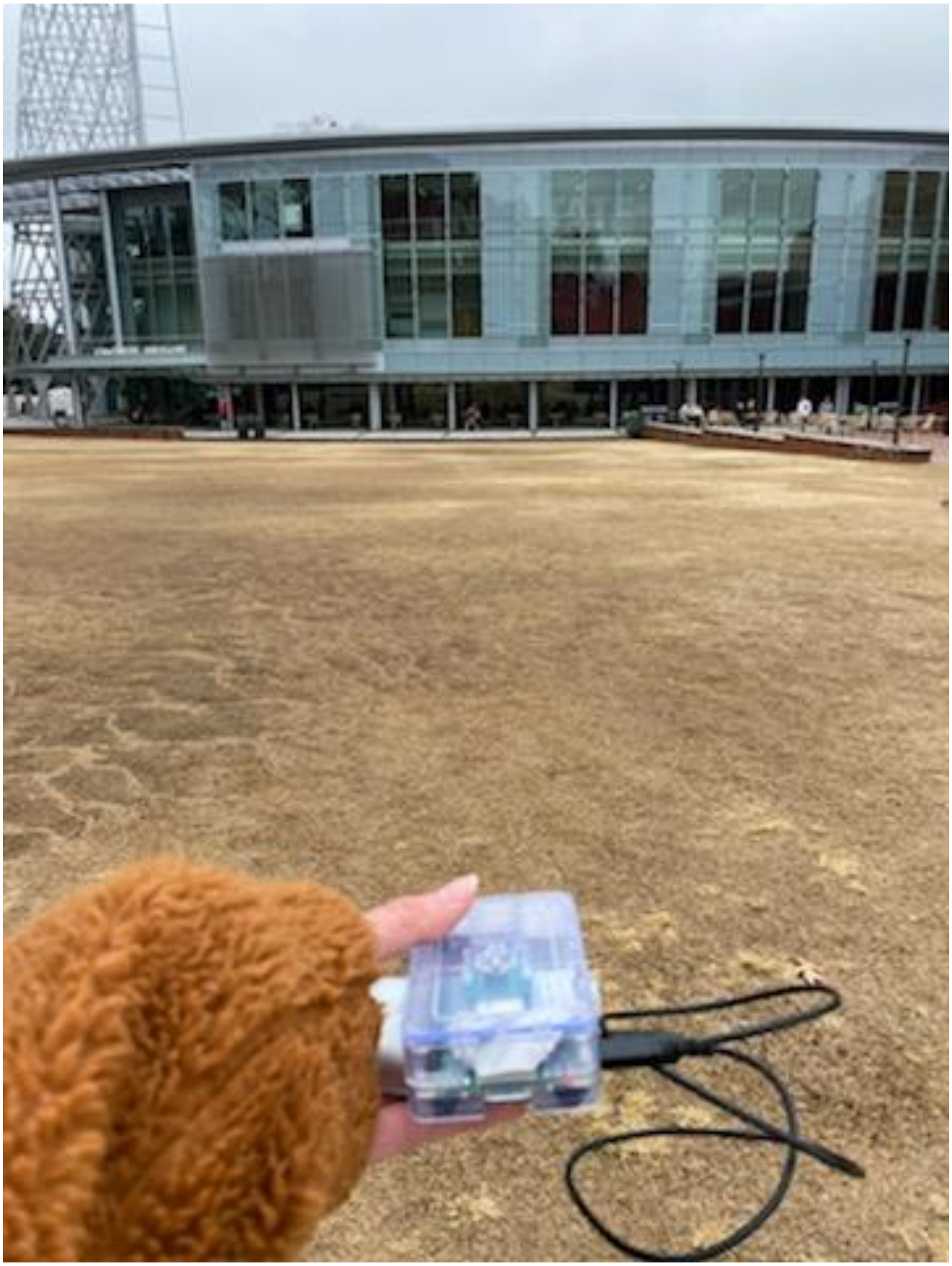
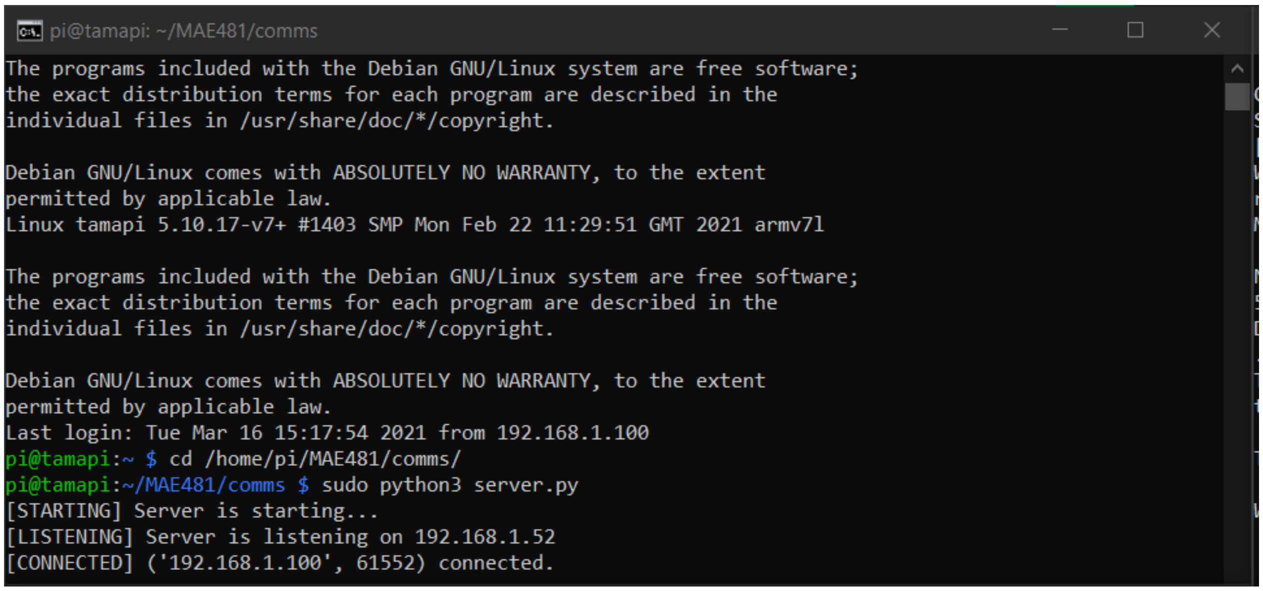


Figure II-15 Physical test location 1

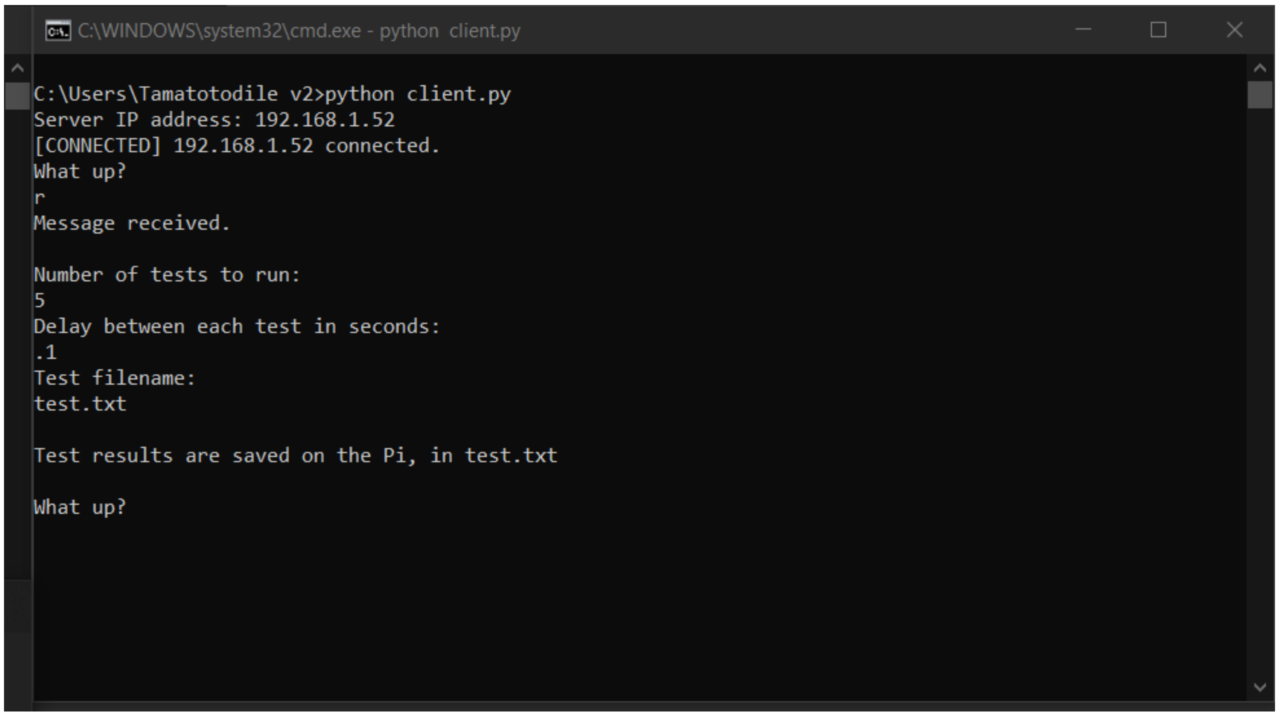
Figure II-16 Physical location 2

Figure II-17 Physical location 3



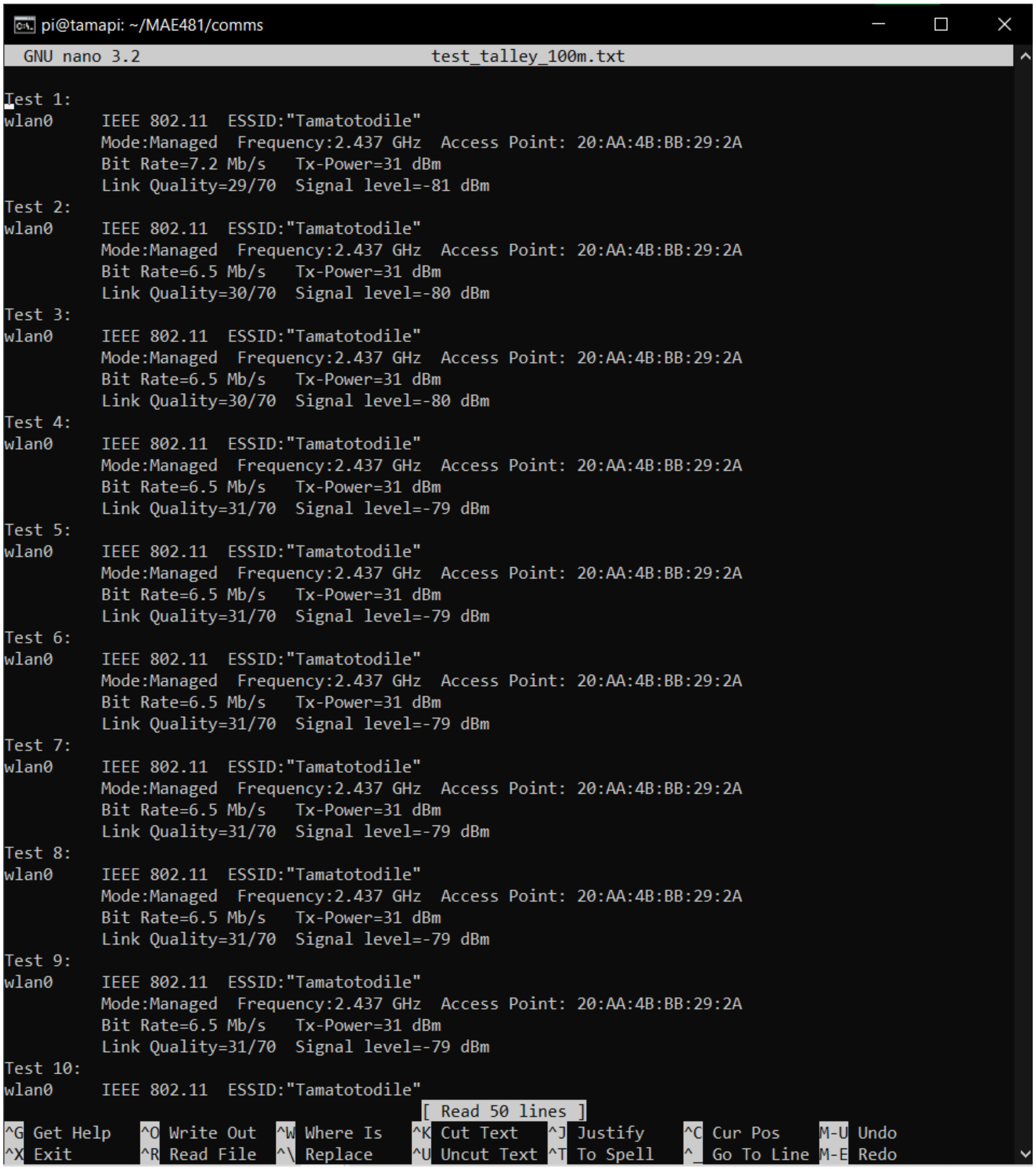
```
pi@tamapi: ~/MAE481/comms
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Linux tamapi 5.10.17-v7+ #1403 SMP Mon Feb 22 11:29:51 GMT 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Mar 16 15:17:54 2021 from 192.168.1.100
pi@tamapi:~ $ cd /home/pi/MAE481/comms/
pi@tamapi:~/MAE481/comms $ sudo python3 server.py
[STARTING] Server is starting...
[LISTENING] Server is listening on 192.168.1.52
[CONNECTED] ('192.168.1.100', 61552) connected.
```

Figure II-18 SSH into Raspberry Pi and starting server



```
C:\WINDOWS\system32\cmd.exe - python  client.py

C:\Users\Tamatotodile v2>python client.py
Server IP address: 192.168.1.52
[CONNECTED] 192.168.1.52 connected.
What up?
r
Message received.

Number of tests to run:
5
Delay between each test in seconds:
.1
Test filename:
test.txt

Test results are saved on the Pi, in test.txt

What up?
```

Figure II-19 Client program and running test



```
pi@tamapi: ~/MAE481/comms
  GNU nano 3.2                               test_talley_100m.txt

Test 1:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=7.2 Mb/s   Tx-Power=31 dBm
          Link Quality=29/70  Signal level=-81 dBm
Test 2:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=30/70  Signal level=-80 dBm
Test 3:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=30/70  Signal level=-80 dBm
Test 4:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=31/70  Signal level=-79 dBm
Test 5:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=31/70  Signal level=-79 dBm
Test 6:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=31/70  Signal level=-79 dBm
Test 7:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=31/70  Signal level=-79 dBm
Test 8:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=31/70  Signal level=-79 dBm
Test 9:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 20:AA:4B:BB:29:2A
          Bit Rate=6.5 Mb/s   Tx-Power=31 dBm
          Link Quality=31/70  Signal level=-79 dBm
Test 10:
wlan0     IEEE 802.11  ESSID:"Tamatotodile"
                                           [ Read 50 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo
```

Figure II-20 Example test results

3. *Required Resources*
   - Windows computer/laptop with 2.4 GHz
   - Router set to 802.11n
   - Monitor and HDMI/Display Port cables
   - Mouse and keyboard
   - Python interpreter
   - Power outlet
   - Open field for the Pi to be distanced from the mock lander
   - Table and chair for test administrator

4. *Expected Results*

The theoretical expectations from a measured -40 dBm at 1 meter away from the lander calculated using Eq. (6) are tabulated as follows,

Range (m)	**Signal Strength (dBm)**	**Minimum Requirement for**
1	-40	N/A (baseline)
100	-60	N/A
500	-67	VoIP/VoWiFi, video streaming
1000	-70	Email, web browsing
10,000	-80	Basic connectivity
100,000	-90	N/A (unusable)

Table II-1 Theoretical Signal Strength at Varying Range

The experimentally measured signal strength is expected to be around -60 dBm with the CubeRover at a distance of 100 meters from the mock lander, adhering to design requirement **DR 8.1.2**.

5. *Success Criteria*

Three levels of success and the failure condition are defined as follows,

- **Exceptional:** Signal strength is indicated to be greater than or equal to -60 dBm with the rover at a distance of less than or equal to 500 meters away from the mock lander.
- **Extensive:** Signal strength is indicated to be greater than or equal to -60 dBm with the rover at a distance of less than or equal to 250 meters away from the mock lander.
- **Acceptable:** Signal strength is indicated to be greater than or equal to -60 dBm with the rover at a distance of less than or equal to 100 meters away from the mock lander.

Failure: Signal strength is indicated to be less than -60 dBm with the rover at a distance of less than or equal to 100 meters away from the mock lander.

III. Results

A. Bitrate Test Results

Test #	Ping 1 (ms)	Ping 2 (ms)	Ping 3 (ms)	Ping 4 (ms)
1	74	34	13	19
2	15	42	33	19
3	22	26	33	14
4	13	48	35	33
5	12	61	25	22
6	31	17	15	26
7	26	25	14	20
8	97	14	23	14
9	28	23	16	22
10	48	49	23	29

Table III-1 Bitrate Test Results

B. Range Test Results

Test #	50 meters		75 meters		100 meters	
	RSSI (dBm)	Bitrate (kBps)	RSSI (dBm)	Bitrate (kBps)	RSSI (dBm)	Bitrate (kBps)
1	-69	5413	-80	3250	-81	900
2	-69	5413	-80	2438	-80	813
3	-69	5413	-78	2438	-80	813
4	-69	5413	-76	2438	-79	813
5	-70	5413	-76	1625	-79	813
6	-70	5413	-76	813	-79	813
7	-70	5413	-76	813	-79	813
8	-70	5413	-76	813	-79	813
9	-70	5413	-77	813	-79	813
10	-71	5413	-77	813	-79	813

Table III-2 Range Test Results

IV. Conclusion

The tests have been conducted with an exceptional success criterion achieved for the bitrate test and a failure criterion fulfilled for the communications range test. The max, min, and average RSSI and bitrate for the Pi at 50 meters, 75 meters, and 100 meters away from the mock lander for 10 data points are as follows:

	50 meters		**75 meters**		**100 meters**	
	RSSI (dBm)	**Bitrate (kBps)**	**RSSI (dBm)**	**Bitrate (kBps)**	**RSSI (dBm)**	**Bitrate (kBps)**
MAX	-69	5400	-76	3300	-79	900
MIN	-71	5400	-80	810	-81	810
AVERAGE	-69.7	5400	-77.2	1600	-79.4	825

The test results did not have a significant outlier and resulted in a predictable range of maximum, minimum, and average. This gives us the confidence that the connection will be more stable on the surface of the moon because there were a lot of other devices near the mock lander and the Pi when tested that would have interfered with the 2.4 GHz network band. The bitrate achieved at the 100 meters is significantly greater than the 40 kilobytes per second that was set as the minimum requirement and satisfied the success criterion of a bitrate greater than 400 kilobytes per second. The average RSSI did not fulfill any success criteria at the 50-meter mark, but it was still able to communicate with the mock lander and perform the test at 100-meter mark. The test results satisfied the failure criterion of the signal strength indicated is less than -60 dBm at a distance less than or equal to 100 meters. However, this may not pose a problem because the expectation was that the Pi would completely lose connection or be unable to communicate at RSSI any lower than -80 dBm.

The next steps to test would be the theoretical -80 dBm for "basic connectivity" and to find out what it means. From the range test, it was shown that the TCP line was still open between the mock lander (client) and the Pi (server), and the mock lander was still able to issue commands even with a signal strength less than -80 dBm. That is why the Communications sub-team will move forward with testing by completing the test for standby protocol, mobilizing the rover with TCP at a distance less than or equal to 100 meters, and have the rover communicate back its diagnostics (temperature, pointing direction, speed, etc.) to be displayed on the mock lander.

Appendix

A. ping.c

```
/**
 *   @file ping.c
 *   @author Timothy Japit
 *
 *   Short script to test the WLAN 2.4 GHz on-board the RasPi.
 */
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <unistd.h>

/** Array length for the strings. */
#define STRING_SIZE 100

/**
 * Returns error if the test to file character is not 'y' or 'n'.
 */
static void usage()
{
  fprintf(stderr, "Response should be 'y' or 'n'\n");
  exit(EXIT_FAILURE);
}

/**
 * Returns 1 if the user chooses to save to file, and 0 otherwise.
 *
 * @param response user response to file saving prompt
 * @return 1 if the user chooses to save to file
 */
static int saveToFile(const char *response)
{
  // save file
  if (strcasecmp("y", response) == 0)
  {
    return 1;
  }
  // no to file saving
  else
  {
    return 0;
  }
}

/**
 * Program starting point.
 *
 * @param argc number of arguments from the command-line
 * @param argv String array of the arguments from the user
 * @return exit status code
 */
int main(int argc, char *argv[])
{
```

```
/** Number of runs */
int runs = 0;

/** Save to file response */
char *response = (char*) malloc(STRING_SIZE * sizeof(char));

/** Filename */
char *filename = (char*) malloc(STRING_SIZE * sizeof(char));

/** Pi IP address */
char *addr = (char*) malloc(STRING_SIZE * sizeof(char));

/** Ping command */
char *command = (char*) malloc(STRING_SIZE * sizeof(char));

// ping command
printf("Command: ");
gets(command);

// prompt for ip address
printf("Pi IP address: ");
gets(addr);

// save test to file?
printf("Save test results to file? (y/n) ");
gets(response);

// response error checking
if (strcasecmp("y", response) != 0 && strcasecmp("n", response) != 0)
{
  usage();
}
else
{
  // valid response
  if (saveToFile(response))
  {
    // filename
    printf("Filename: ");
    gets(filename);

    /*
     * Code from https://stackoverflow.com/questions/230062/whats-the-best-way-to-check-if-a-file-exists-in-c
     */
    // if file exists, append
    if (access(filename, F_OK) == 0)
    {
      printf("\nResults will be appended to the file.\n\n");
    }
    // otherwise create new
    else
    {
      printf("\nResults will be written to a new file.\n\n");
    }
    strcat(addr, " >> ");
```

```
      // concatenate ip address to file
      strcat(addr, filename);

    }
  }

  // prompt for number of runs
  printf("Number of runs: ");
  (void) scanf("%d", &runs);
  printf("\n");

  // concatenating command, address, and output redirection to file
  strcat(command, " ");
  strcat(command, addr);

  for (int i = 0; i < runs; ++i)
  {
    // running command on shell
    system(command);
  }

  // open in console if saved to file
  if (saveToFile(response))
  {
    char consoleOut[STRING_SIZE] = "more ";
    strcat(consoleOut, filename);
    system(consoleOut);
  }

  printf("\n");

  // free mem
  free(response);
  free(filename);
  free(addr);
  free(command);

  exit(EXIT_SUCCESS);
}
```

B. server.py

```
"""Server program for the RTDT rover.

@author Timothy Japit
"""
import socket
import sys
import os
from time import sleep

# RANGE TEST MODULE
range_test_path = '/home/pi/MAE481/comms/range_test'
sys.path.insert(0, range_test_path)
import range_test

# STANDBY PROTOCOL MODULE
standby_protocol_path = '/home/pi/MAE481/comms/'
sys.path.insert(0, standby_protocol_path)
import standby_protocol

# Server socket info
HEADER = 64
PACKET_LIMIT = 1024
PORT = 5050
SERVER = "192.168.1.52"
ADDR = (SERVER, PORT)
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

# initialize server socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # IPv4, TCP
# bind server socket to address
server.bind(ADDR)

def main():
    """Server starts listening for a new connection.

    """
    # indicate server starting
    print("[STARTING] Server is starting...")
    server_up = True
    while server_up:
        # listening, only 1 device allowed to connect at any time
        server.listen(1)
        print(f"[LISTENING] Server is listening on {SERVER}")

        # store client socket and address after accepting connection
        conn, addr = server.accept()

        # indicate connected by printing to cmd and client
        print(f"[CONNECTED] {addr} connected.")
        conn.send(f"[CONNECTED] {SERVER} connected.".encode(FORMAT))


        # connected to client and listening for message
        connected = True
        while connected:
            # prompt client for action
            conn.send(b"What up?")
            # receive message from client
            msg = conn.recv(PACKET_LIMIT).decode(FORMAT)
            # print message received and indicate to client that message was
            # received
            print('\n' + msg + '\n')
            conn.send(b"Message received.\n")
            # DISCONNECT FROM ROVER
            if msg == DISCONNECT_MESSAGE:
                # prompt to keep server up or otherwise
                conn.send(b"Keep server running?(y/n)")
                # receive response
                resp = conn.recv(PACKET_LIMIT).decode(FORMAT)
                # server down
                if resp == 'n':
                    server_up = False
                # connection flag
                connected = False
                # disconnect message
                print(f"[DISCONNECTED] {addr} connection closed.")
             # RANGE TEST
             elif msg == 'r':
                 # number of reps
                 conn.send(b"Number of tests to run: ")
                 n = int(conn.recv(PACKET_LIMIT).decode(FORMAT))

                 # delay
                conn.send(b"Delay between each test in seconds: ")
                 delay = float(conn.recv(PACKET_LIMIT).decode(FORMAT))

                 # filename
                 conn.send(b"Test filename: ")
                 filename = conn.recv(PACKET_LIMIT).decode(FORMAT)

                 # run range test by calling the script that actually tests it
                 # debug: need to create the file first and change mod permission before writing to it
                 # otherwise permission denied error
                 range_test.range_test(n, delay, filename)

        # close connection
        conn.close()
    # server down message
    print(f"[SERVER DOWN] Server terminated by {addr}")


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

C. client.py

```
"""Client program for connecting to the rover.

@author Timothy Japit

"""
import socket
import sys

# Server protocol and address
HEADER = 64
PACKET_LIMIT = 1024
PORT = 5050
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

# create client socket
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

def main():
    """Starts connection with server and requests the test file.

    """
    # prompt for server IP
    server_ip = input("Server IP address: ")
    addr = (server_ip, PORT)
    # connect to server address
    client.connect(addr)
    # CONNECTED
    print(client.recv(PACKET_LIMIT).decode(FORMAT))
    # connected to server
    connected = True
    while connected:
        # receive prompt
        print(client.recv(PACKET_LIMIT).decode(FORMAT))
        # answer prompt
        response = input("response: ")
        # send response
        client.send(response.encode(FORMAT))
        # receive "message was received" response
        print(client.recv(PACKET_LIMIT).decode(FORMAT))
        # DISCONNECT
        if response == DISCONNECT_MESSAGE:
            # receive prompt to keep server up or otherwise
            print(client.recv(PACKET_LIMIT).decode(FORMAT))
            # send response
            client.send(input().encode(FORMAT))
            # break out of loop
            connected = False
            # send disconnect message
            client.send(response.encode(FORMAT))
            # close connection using client socket
            client.close()
            # DISCONNECTED
            print(f"[DISCONNECTED] {server_ip} connection closed.")
        # RANGE TEST
        elif response == 'r':
            # number of reps
            print(client.recv(PACKET_LIMIT).decode(FORMAT))
            n = input()
            client.send(n.encode(FORMAT))

            # delay
            print(client.recv(PACKET_LIMIT).decode(FORMAT))
            delay = input()
            client.send(delay.encode(FORMAT))

            # filename
            print(client.recv(PACKET_LIMIT).decode(FORMAT))
            filename = input()
            client.send(filename.encode(FORMAT))

            # message indicating file is saved on pi
            print(f"\nTest results are saved on the Pi, in {filename}\n")

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

References

[1] "CubeRover Payload User Guide," *Astrobotic*, 2020, www.astrobotic.com/cuberover-payload-user-guide.

[2] "Data Rate Limits in Digital Transmission," *York University*, 2019, https://www.eecs.yorku.ca/course_archive/2015-16/F/3213/CSE3213_06_AnalogDigitalTransmission_F2015_posted_part3.pdf

[3] Goyal, A., "Maximum Data Rate (Channel Capacity) for Noiseless and Noisy Channels," *GeeksforGeeks*, 2019, https://www.geeksforgeeks.org/maximum-data-rate-channel-capacity-for-noiseless-and-noisy-channels/

[4] Pierce, J. R., "An introduction to information theory: Symbols, signals & noise," *Dover Publications*, 1980, 2nd edition, New York.

[5] Ian, "Difference Between dB and dBm," *DifferenceBetween.net*, 2011, http://www.differencebetween.net/science/difference-between-db-and-dbm/

[6] zx8754, "Calculate Upload/Download Speed by Ping," *stackoverflow*, 2019, https://stackoverflow.com/questions/4575537/calculate-upload-download-speed-by-ping

[7] "RSSI," *speedcheck*, retrieved Oct. 9, 2020, Available: https://www.speedcheck.org/wiki/rssi/#signal-strength

[8] Premachandran, R., "SNR, RSSI, EIRP and Free Space Path Loss," *Cisco*, retrieved Oct. 9, 2020, Available: https://community.cisco.com/t5/wireless-mobility-documents/snr-rssi-eirp-and-free-space-path-loss/ta-p/3128478

[9] "Understanding RSSI," *metageek*, retrieved Oct. 9, 2020, Available: https://www.metageek.com/training/resources/understanding-rssi.html

[10] Chruszczyk, L., Zajac, A., & Grzechca, D., "Comparison of 2.4 and 5 GHz WLAN network for purpose of indoor and outdoor location," *International Journal of Electronics and Telecommunications*, Vol. 62, No. 1, 2016, pp. 71-79. http://dx.doi.org.prox.lib.ncsu.edu/10.1515/eletel-2016-0010

[11] "IEEE 802.11," *Wikipedia*, https://en.wikipedia.org/wiki/IEEE_802.11#Protocol, retrieved Oct. 29, 2020

[12] "IEEE 802.11ac: What Does it Mean for Test?," *Litepoint*, https://web.archive.org/web/20140816132722/http://litepoint.com/whitepaper/80211ac_Whitepaper.pdf, retrieved Oct. 29, 2020

[13] IEEE, "802.11-2016 - IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE*, 2016, retrieved Oct. 8, 2020, Available: https://ieeexplore.ieee.org/document/7786995

[14] Villegas, E. G., López-Aguilera, E., Vidal, R., & Paradells, J., "Effect of Adjacent-channel Interference in IEEE 802.11 WLANs," *2007 2nd International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, pp. 118-125, 2007. doi:10.1109/crowncom.2007.4549783