ECSE 4660 Internetworking of Things

Project Report – Multi-room Audio Streamer

May 3rd, 2021

Yilu Zhou, zhouy19@rpi.edu
Cameron Barker, barkec2@rpi.edu

Abstract

We focused on the problem of indoor localization and audio streaming and presented a system that achieves seamless audio transition based on user location. Our design consists of several parts: the control server, the streaming server and three clients. The streaming server streams the music to the clients. The control server processes the RSSI information sent by the clients and determines the user location. Then the control server sends a signal to the closest client to play the music transmitted by the streaming server. Our final work is able to connect the control server, the streaming server and three clients together under the same network and play the music based on user location.

Methodology

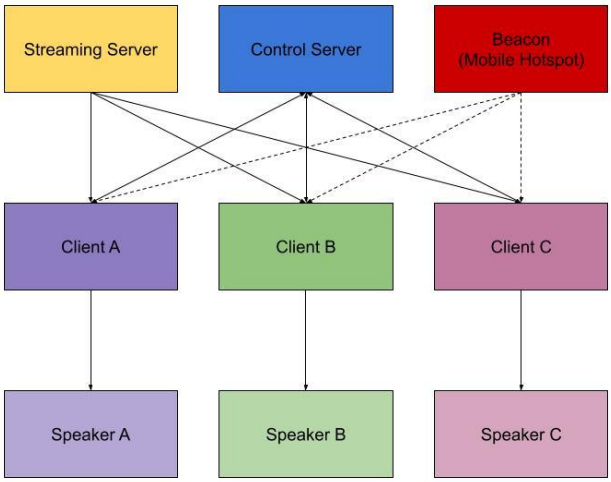

Figure 1: A schematic diagram of the network.

Our system consists of 4 main parts, the streaming server, the control server, the clients, and the beacon (user's mobile hotspot). The streaming server sends the music data to the 3 clients. The

control server sends control commands to the clients and determines which speaker to use based on which client the beacon is the closest to. The clients measure the signal strength of the beason and report the RSSI back to the control server.

Streaming Server

Because of the design choice, the clients have no information about the music being played. This design provides more freedom to the users as they can now choose their favorite music or internet radio station instead of the music existing on the clients. To fulfill the design, the streaming server is in charge of broadcasting the music to the clients. It can be internet radio stations such as WRPI or a local music server. In the test case, considering the high latency to access WRPI, a local music server is set up using the VLC player.

Control Server

The control server is the core of the streaming system. It should be in the same network with all the clients. The users need to provide the port to listen, the number of clients in the system, the streaming server address, and the mobile hotspot SSID. The control server will then send the streaming server address and the mobile hotspot SSID to the clients when the clients connect to the server. During the runtime, the control server is in charge of calculating the distance between mobile hotspot and clients and managing the audio playback on all client Raspberry Pis. It can be a desktop or laptop computer in the same network or a client Raspberry Pi. Faster devices are recommended to accelerate the data processing. When the users want to stop the streamer, they need to stop the control server. The control server will then send a “STOP” signal to all the clients to end the program.

Clients

To build a client, a wired speaker should be connected to the 3.5mm audio jack on the Raspberry Pi. Also, all the clients should be in the same network with the control server. If a personal media server is configured, it should also be in the same network. If Internet HTTP live streaming is used, the network should be connected to the internet. The clients are in charge of determining the signal strength of the mobile hotspot. It is important to select an unobstructed location for Raspberry Pi. For example, avoid placing RPi behind metal plates as the signal will be weakened. When some clients lose connection with the control server, the system will automatically stop working.

Figure 2: Example of Client Setup.

Command Set

Each TCP message consists of 64 bytes. The server and clients will create a buffer of size 64 each time to avoid data loss. The structure is as follows: the first 6 bytes of the message should be the name of the device. The next 6 bytes should be the type of the message. The remaining 52 bytes should be the real message. If the server wants to send a global message the name “ALL” is used as the indicator.

Test Result

The whole system is built in Yilu Zhou’s home. The approximate room layout is

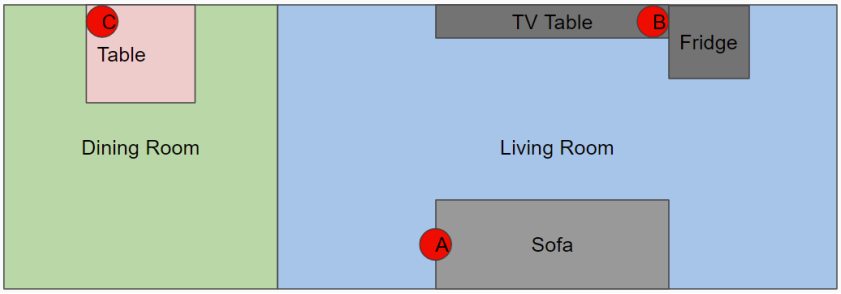

Figure 3: Test Room Layout.

The location fingerprints for the test room layout is

Name	A	B	C
Sofa	-42.9	-64.53	-67.97
Fridge	-61.3	-31.88	-58.45
Dining	-72.6	-65.02	-58.75

Table 1: Location Fingerprints.

These data are reflected in the Python source code.

Conclusion

Different parts of the system can be connected under the same network. The streaming server is set up and running correctly. The control server is able to get connections from all three clients. The clients can get the hotspot name and the streaming server address given by the user on the control server. During the runtime, all the three clients are able to get the signal strength of the mobile hotspot with reasonable accuracy. Although there is some time needed for the control server to decide the closest client and send the command, the interruption of music is avoided by first allowing the closest client to play the music and then wait for 1.75 seconds. After the 1.75 seconds, the previous client stops the music streaming. The user will always hear the music when moving around. The cost of the system is controlled by using old wired speakers instead of purchasing new Bluetooth or wireless speakers. No extra sensors are needed for the project as wireless localization is used. The user does not need to have knowledge of using ultrasonic rangers or PIR sensors when setting up the system. The clients' Raspberry Pi can be placed in a noise isolation box to avoid the noise from fans.

In conclusion, the completed system is able to achieve seamless audio transition based on user location. For further research, the users should be able to change the volume and music using the control server or their mobile phone without the need to restart the whole system. The control server can automatically detect the number of clients in the system and connect with them without the need of user configuration. The clients can be connected with each other and communicate the signal strength with other clients to eliminate the need of a dedicated control server.

To complete the project, Cameron Barker and Yilu Zhou contribute equally.

Appendix A: System Configuration

Configure Personal Media Streaming Server

To set up the streaming server, follow these steps:

1. Install VLC media player from the VLC Official Site.
2. With the VLC media player window open, click the “Media” button on the top left corner. Then select the “Stream…” option.
3. With the Open Media window open, click the “Add…” button under the “File” tab. Then choose the music file you want to stream. A file with long time duration is recommended as VLC does not support streaming list of music files.
4. After selecting the music file, click the “Stream” button on the bottom right corner. Then click the “Next” button again until you see the “Destination Setup” window.
5. In the “Destination Setup” window, select “HTTP” as the “New destination”. Then click the “Add” button. If you want to listen to the stream on the server as well, check the “Display locally” box. Under the “HTTP” tab, set the Port number. Do not change the Path. Then click the “Next” button.
6. In the “Transcoding Options” window, check “Activate Transcoding” box. Select “Audio - MP3” or “Audio - FLAC” as the profile depending on the original music file format. Then click the “Next” button.
7. In the “Option Setup” window, do not change anything, click the “Stream” button on the bottom right corner.
8. The streaming server is now set up.

Configure Control Server

To set up the control server, follow these steps:

1. Download the “Server.py” file.
2. Run the script with “python3 Server.py -p [port number] -a [address of streaming server, e.g. http://192.168.0.100:8080] -n [number of Raspberry Pis in the network] -s [SSID of the mobile hotspot]”. If there is no dependency error, you should see “waiting for connection” shown in the terminal.
3. Then we need to set up the 3 Raspberry Pis as clients.

Configure Clients

To connect the clients with the control server, follow these step:

1. Download the “Client.py” file to the Raspberry Pis.
2. Run the script with “python3 Client.py -s [server address] -p [port number] -n [name for this client]”.
3. Ideally, when the control server and Raspberry Pi are under the same network, you should see “allow connection from IP_ADDRESS?” message on the control server terminal. You

should check the prompted ip address with the actual Raspberry Pi address. To allow connection, type “y” in the control server terminal.

4. Repeat the above procedures to connect all clients to the control server.
5. After all clients are set up, the control server will send command to the clients to control the audio playback. You should hear the music playing from the speaker connected to the clients. Only one device will play the music at the same time depending on the distance.

Appendix B: Getting Location Fingerprints

Before running the code, the location fingerprints should be acquired following these steps:

1. Download the “Fingerprint_Finder.py” file.
2. Run “python3 Fingerprint_Finder.py -p [port number] -a [address of streaming server, e.g. http://192.168.0.100:8080] -n [number of Raspberry Pis in the network] -s [SSID of the mobile hotspot]”. If there is no dependency error, you should see “waiting for connection” shown in the terminal.
3. Then we need to set up the 3 Raspberry Pis as clients.
4. After setting up the clients, the program will automatically run the RSSI scan of the provided mobile hotspot SSID 100 times. Then the program will print the average RSSI from the three Raspberry Pis. Use the readings to build a fingerprint based localization database.
5. Modify the “Server.py” file to reflect the RSSI readings.

The location fingerprints are used to determine the Euclidean distance between the mobile hotspot and clients.

Appendix C: Command Set

Having a command set is useful for the system debugging. Below is the table for all possible types and messages:

Type	Message	Description
SADDR	the streaming server address	server → clients, the clients will set the streaming server address.
HSADDR	the name of mobile hotspot	server → clients, the clients will set the hotspot name for RSSI localization.
TIME	REPORTTIME	server → clients, the clients will send the client local time back to the server.
CMD	STOP	server → clients, the clients will disconnect from the server.
AUDIO	STARTSTREAM	server → clients, the clients will start to play the audio.
AUDIO	ENDSTREAM	server → clients, the clients will stop the audio.
RSSI	MEASURERSSI	server → clients, the clients will send the RSSI of the mobile hotspot to the server.
RSSI	RSSI reading of the hotspot	server ← clients, the clients will send the RSSI of the mobile hotspot to the server.
DIST	the measured distance	server ← clients, the clients will send the distance based on RSSI localization to the server.
IP	the local IP address	server ← clients, the clients will send the localhost IP to the server.

Table 2: Command Set.

Appendix D: Sample Control Server Log

```
PS C:\Users\zhouy\OneDrive\Documents\RPI\ECSE4660\Final
Project\Code> python Server.py -n 3 -p 1234 -s Mix2 -a
http://192.168.1.110:8080
============================================================
==============
waiting for connection
allow connection from 192.168.1.111?y
[RECV]  Sofa  IP    192.168.1.111
waiting for connection
allow connection from 192.168.1.113?y
[RECV]  FridgeIP    192.168.1.113
waiting for connection
allow connection from 192.168.1.114?y
[RECV]  DiningIP    192.168.1.114
all devices found
[SEND]  ALL   SADDR http://192.168.1.110:8080
[SEND]  ALL   SADDR http://192.168.1.110:8080
[SEND]  ALL   SADDR http://192.168.1.110:8080
[SEND]  ALL   HSADDRMix2
[SEND]  ALL   HSADDRMix2
[SEND]  ALL   HSADDRMix2
[SEND]  ALL   TIME  REPORTTIME
[RECV]  Sofa  TIME  00:57:51
[SEND]  ALL   TIME  REPORTTIME
[RECV]  FridgeTIME  12:57:51
[SEND]  ALL   TIME  REPORTTIME
[RECV]  DiningTIME  00:57:51
[SEND]  ALL   RSSI  MEASURERSSI
[SEND]  ALL   RSSI  MEASURERSSI
[SEND]  ALL   RSSI  MEASURERSSI
[RECV]  Sofa  RSSI  -65
[RECV]  FridgeRSSI  -58
[RECV]  DiningRSSI  -59
[INFO]  [-65. -58. -59.]
[INFO]  [26.158363863208262, 26.808873530978506,
3.0142826675678567]
[SEND]  DiningAUDIO STARTSTREAM
[SEND]  Sofa  AUDIO ENDSTREAM
[SEND]  FridgeAUDIO ENDSTREAM
```

Appendix E: System Photos

Figure 4: Client A (Sofa).

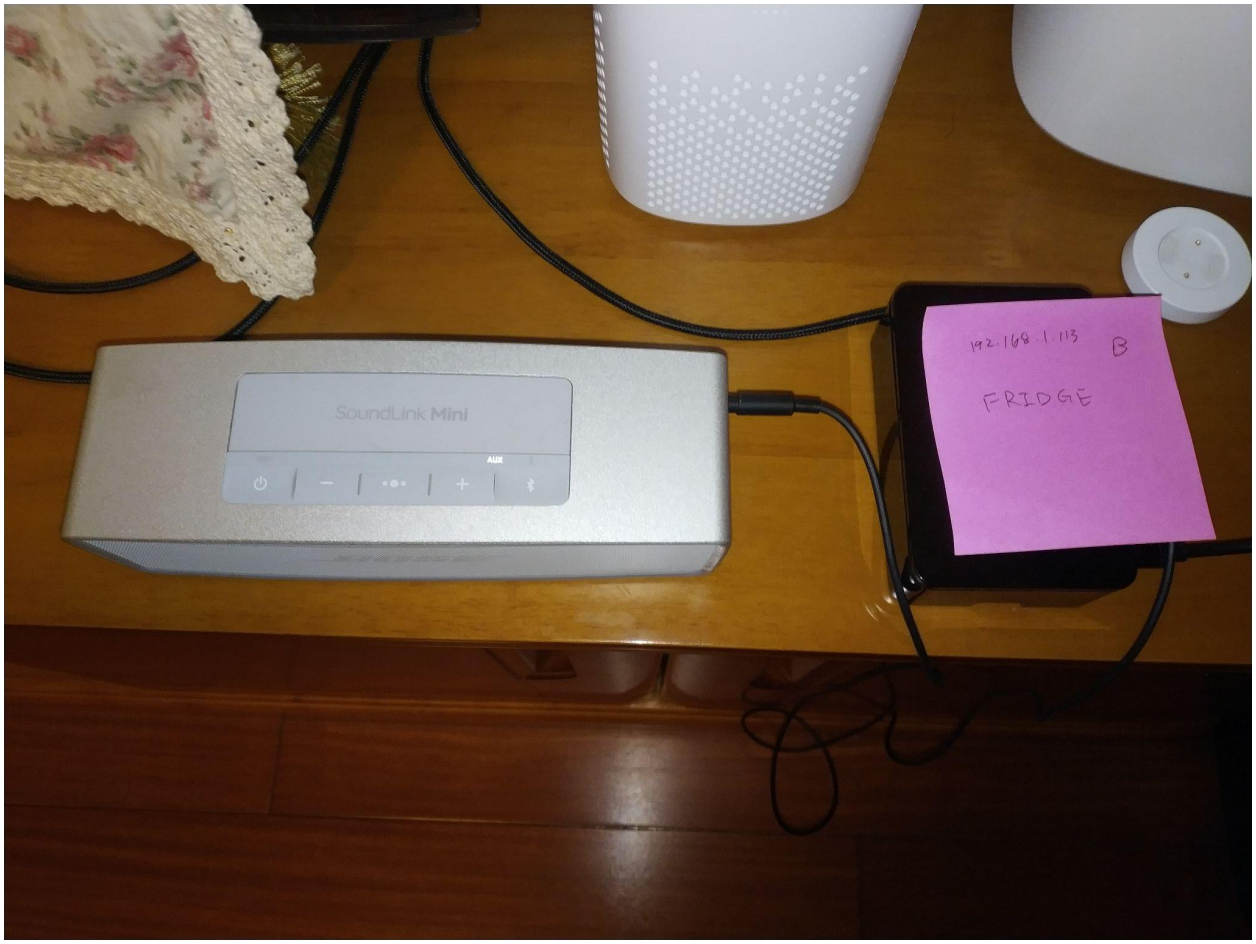

Figure 5: Client B (Fridge).

Figure 6: Client C (Dining).

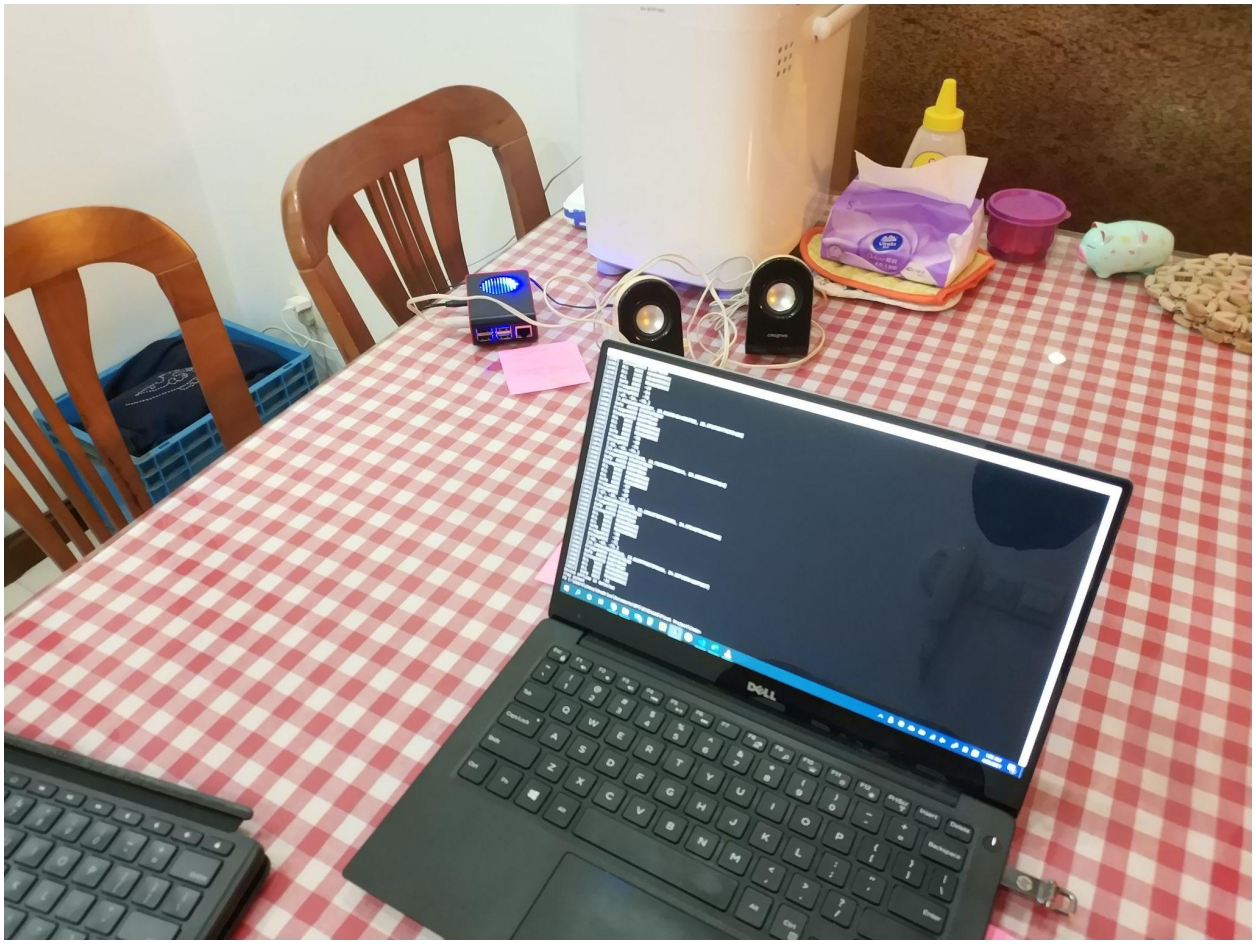

Figure 7: Control Server & Streaming Server.

Appendix F: Demo Address

Short Demo: https://youtu.be/-1pG2TTVRKY
Full Demo: https://youtu.be/Ytn-jPnTPTY

Appendix G: Python RSSI Package Error Solution

RSSI package is used to determine the signal strength of the mobile hotspot. However, the package is made for Python 2 and there is type error in the code. To solve the problem, follow https://github.com/jvillagomez/rssi_module/issues/1 to change the code of the package.

Appendix H: Server.py

```
import argparse
import socket
import re
from datetime import datetime
import time
import numpy as np


def sendMessage(connection, name, type, message):
    name = str(name)
    type = str(type)
    message = str(message)
    fname = name.ljust(6, " ")
    ftype = type.ljust(6, " ")
    fmessage = message.ljust(52, " ")
    tdata_exp = fname + ftype + fmessage
    print("[SEND]  " + tdata_exp)
    try:
        connection.send(tdata_exp.encode())
    except:
        print(name, "is offline")
        exit(1)


def sendGlobalMessage(clist, name, type, message):
    for i in range(len(clist)):
        sendMessage(clist[i], name, type, message)


def recvMessage(connection):
    try:
        rdata_bytes = connection.recv(64)
    except:
        print("Device offline is detected")
        exit(1)

    else:
        rdata = rdata_bytes.decode()
        print("[RECV]  " + rdata)
```

```
        dname = rdata[0:6].strip(" ")
        dtype = rdata[6:12].strip(" ")
        dmessage = rdata[12:64].strip(" ")
        return (dname, dtype, dmessage)


parser = argparse.ArgumentParser()
parser.add_argument("-p", "--port", type=int, default=1234,
                    help="port of the server to connect",
required=True)
parser.add_argument(
    "-a", "--address", help="address of the music streaming
server", default = "http://192.168.0.103:8080",
     required=True)
parser.add_argument("-n", "--number", type=int, default=1,
help="number of clients in the network", required=True)
parser.add_argument("-s", "--ssid", default = "Mix2",
help="Hotspot SSID", required=True)
args = parser.parse_args()
port = args.port
stream = args.address
num_clients = args.number
ssid = args.ssid
rejected_ip = set()
allowed_ip = set()
name = "ALL"

threads = [None] * num_clients
results = [None] * num_clients
old_results = [-100] * num_clients
error_counts = [0] * num_clients
connections = [None] * num_clients
names = [None] * num_clients

pos_A = np.array((-42.9, -61.3, -72.6))
pos_B = np.array((-64.53, -31.88, -65.02))
pos_C = np.array((-67.97, -58.45, -58.75))

try:
    print("=" * 79)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    s.bind(('', port))
    s.listen(num_clients)
    count = 0
    while True:
        print("waiting for connection")
        c, addr = s.accept()
        if addr[0] in rejected_ip:
            c.close()
            continue
        if addr[0] not in allowed_ip:
            flag = input("allow connection from "+addr[0]+"?")
            if (flag == "y" or flag == "Y"):
                allowed_ip.add(addr[0])
            else:
                rejected_ip.add(addr[0])
                c.close()
                continue
        msg = recvMessage(c)
        names[count] = msg[0]
        connections[count] = c

        count += 1

        if(count == num_clients):
            break


    print("all devices found")

    sendGlobalMessage(connections, name, "SADDR", stream)
    sendGlobalMessage(connections, name, "HSADDR", ssid)

    for i in range(len(connections)):
        sendMessage(connections[i], "ALL", "TIME", "REPORTTIME")
        msg = recvMessage(connections[i])

    while True:
        sendGlobalMessage(connections, "ALL", "RSSI",
"MEASURERSSI")
        dist = [100,100,100]
        # First get distance measurement from client
```

```
        for i in range(len(connections)):
            msg = recvMessage(connections[i])
            # Get the RSSI reading from the clients.
            if (msg[0] == names[0]):
                if (msg[2] == "-100"):
                    results[0] == old_results[0]
                else:
                    results[0] = float(msg[2])
                    old_results[0] = results[0]
            elif (msg[0] == names[1]):
                if (msg[2] == "-100"):
                    results[1] == old_results[1]
                else:
                    results[1] = float(msg[2])
                    old_results[1] = results[1]
            else:
                if (msg[2] == "-100"):
                    results[2] == old_results[2]
                else:
                    results[2] = float(msg[2])
                    old_results[2] = results[2]

        # Create np array for calculation.
        results = np.array(results)
        print("[INFO]  " + str(results))
        dist[0] = np.linalg.norm(results - pos_A)
        dist[1] = np.linalg.norm(results - pos_B)
        dist[2] = np.linalg.norm(results - pos_C)
        print("[INFO]  " + str(dist))

        # Find the smallest distance in the dist list
        min_value = min(dist)
        min_index = dist.index(min_value)

        sendMessage(connections[min_index], names[min_index],
"AUDIO", "STARTSTREAM")
        time.sleep(1.75) # Minimize the delay

        for i in range(len(connections)):
            if(i != min_index):
```

```
                sendMessage(connections[i], names[i], "AUDIO",
"ENDSTREAM")

    for i in range(len(connections)):
        sendMessage(connections[i], "ALL", "CMD", "STOP")
        connections[i].close()


except ConnectionRefusedError:
    print("Connection refused. You need to run server program
first.")
finally:  # must have
    print("free socket")
    s.close()
```

Appendix I: Fingerprint_Finder.py

```
import argparse
import socket
import re
from datetime import datetime
import time


def sendMessage(connection, name, type, message):
    name = str(name)
    type = str(type)
    message = str(message)
    fname = name.ljust(6, " ")
    ftype = type.ljust(6, " ")
    fmessage = message.ljust(52, " ")
    tdata_exp = fname + ftype + fmessage
    print("[SEND]  " + tdata_exp)
    try:
        connection.send(tdata_exp.encode())
    except:
        print(name, "is offline")
        exit(1)


def sendGlobalMessage(clist, name, type, message):
    for i in range(len(clist)):
        sendMessage(clist[i], name, type, message)


def recvMessage(connection):
    try:
        rdata_bytes = connection.recv(64)
    except:
        print("Device offline is detected")
        exit(1)

    else:
        rdata = rdata_bytes.decode()
        print("[RECV]  " + rdata)
        dname = rdata[0:6].strip(" ")
```

```
        dtype = rdata[6:12].strip(" ")
        dmessage = rdata[12:64].strip(" ")
        return (dname, dtype, dmessage)


parser = argparse.ArgumentParser()
parser.add_argument("-p", "--port", type=int, default=1234,
                    help="port of the server to connect",
required=True)
parser.add_argument(
    "-a", "--address", help="address of the music streaming
server", default = "http://192.168.0.103:8080",
     required=True)
parser.add_argument("-n", "--number", type=int, default=1,
help="number of clients in the network", required=True)
parser.add_argument("-s", "--ssid", default = "Mix2",
help="Hotspot SSID", required=True)
args = parser.parse_args()
port = args.port
stream = args.address
num_clients = args.number
ssid = args.ssid
rejected_ip = set()
allowed_ip = set()
name = "ALL"

threads = [None] * num_clients
results = [None] * num_clients
connections = [None] * num_clients
names = [None] * num_clients
RSSI0 = list()
RSSI1 = list()
RSSI2 = list()


try:
    print("=" * 79)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('', port))
    s.listen(num_clients)
    count = 0
```

```
    while True:
        print("waiting for connection")
        c, addr = s.accept()
        if addr[0] in rejected_ip:
            c.close()
            continue
        if addr[0] not in allowed_ip:
            flag = input("allow connection from "+addr[0]+"?")
            if (flag == "y" or flag == "Y"):
                allowed_ip.add(addr[0])
            else:
                rejected_ip.add(addr[0])
                c.close()
                continue
        msg = recvMessage(c)
        names[count] = msg[0]
        connections[count] = c

        count += 1

        if(count == num_clients):
            break

    print("all devices found")

    sendGlobalMessage(connections, name, "SADDR", stream)
    sendGlobalMessage(connections, name, "HSADDR", ssid)

    for i in range(len(connections)):
        sendMessage(connections[i], "ALL", "TIME", "REPORTTIME")
        msg = recvMessage(connections[i])

    for _ in range(100):
        sendGlobalMessage(connections, "ALL", "RSSI",
"MEASURERSSI")
        for i in range(len(connections)):
            msg = recvMessage(connections[i])
            if (msg[2] != "-100"):
                if i == 0:
                    RSSI0.append(float(msg[2]))
                elif i == 1:
```

```
                    RSSI1.append(float(msg[2]))
                else:
                    RSSI2.append(float(msg[2]))
        time.sleep(0.5)

    rssi_avg0 = 0
    rssi_avg1 = 0
    rssi_avg2 = 0
    if (len(connections) == 1):
        rssi_avg0 = sum(RSSI0) / len(RSSI0)
    elif (len(connections) == 2):
        rssi_avg0 = sum(RSSI0) / len(RSSI0)
        rssi_avg1 = sum(RSSI1) / len(RSSI1)
    else:
        rssi_avg0 = sum(RSSI0) / len(RSSI0)
        rssi_avg1 = sum(RSSI1) / len(RSSI1)
        rssi_avg2 = sum(RSSI2) / len(RSSI2)

    print(rssi_avg0, rssi_avg1, rssi_avg2)

    for i in range(len(connections)):
        sendMessage(connections[i], "ALL", "CMD", "STOP")
        connections[i].close()


except ConnectionRefusedError:
    print("Connection refused. You need to run server program
first.")
finally:  # must have
    print("free socket")
    s.close()
```

Appendix J: Client.py

```
import argparse
import socket
import re
import json
import sys
from datetime import datetime
import vlc
import rssi
import time


def sendMessage(name, type, message):
    name = str(name)
    type = str(type)
    message = str(message)
    fname = name.ljust(6, " ")
    ftype = type.ljust(6, " ")
    fmessage = message.ljust(52, " ")
    tdata_exp = fname + ftype + fmessage
    print("[SEND]  " + tdata_exp)
    s.send(tdata_exp.encode())


def recvMessage():
    try:
        rdata_bytes = s.recv(64)
    except:
        print("Server is down")
        exit(1)
    finally:
        rdata = rdata_bytes.decode()
        print("[RECV]  " + rdata)
        dname = rdata[0:6].strip(" ")
        dtype = rdata[6:12].strip(" ")
        dmessage = rdata[12:64].strip(" ")
        if (dname == "ALL" or dname == name):
            return (dtype, dmessage)
        else:
            print("Data compromised")
```

```
            exit(1)


def get_host_ip():
    try:
        s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        s.connect(('8.8.8.8',80))
        ip=s.getsockname()[0]
    finally:
        s.close()
    return ip


def hotspotDistance(SSIDs, SU, MP, N):
    ap_info = rssi_scanner.getAPinfo(networks=SSIDs, sudo=SU)
    try:
        rssi = ap_info[0]["signal"]
    except:
        rssi = -100

    distance = 10**((MP-rssi)/(10*N))
    return distance


def hotspotRSSI(SSIDs, SU):
    ap_info = rssi_scanner.getAPinfo(networks=SSIDs, sudo=SU)
    try:
        rssi = ap_info[0]["signal"]
    except:
        rssi = -100
    return rssi


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-s", "--server", type=str,
default="127.0.0.1",
                        help="A string of server IP (default:
127.0.0.1)")
    parser.add_argument("-p", "--port", type=int,
```

```
                        help="port of the server to connect",
required=True)
    parser.add_argument("-n", "--name", help="name of device",
required=True)
    args = parser.parse_args()

    # get the ip and port of the server
    ip = args.server
    port = args.port
    name = args.name
    streamServer = ""
    streamMedia = vlc.MediaPlayer()

    interface = "wlan0"
    rssi_scanner = rssi.RSSI_Scan(interface)
    ssid = list()

    print("Trying to connect to %s:%d" % (ip, port))

    msg = b""
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((ip, port))
        myip = get_host_ip()

        sendMessage(name, "IP", myip)

        while True:
            msg = recvMessage()
            if(msg[0] == "CMD"):
                if(msg[1] == "STOP"):
                    s.close()
                    break
                elif(msg[1] == "MEASURESSID"):
                    # Hotspot based distance measurement
(currently used)
                    dist = hotspotDistance(ssid, True, -40, 3)
                    dist_str = str(dist)
                    sendMessage(name, "DIST", dist_str)
            elif(msg[0] == "TIME"):
                if(msg[1] == "REPORTTIME"):
```

```
                now = datetime.now()
                current_time = now.strftime("%H:%M:%S")
                sendMessage(name, "TIME", current_time)
            elif(msg[0] == "SADDR"):
                streamServer = msg[1]
                streamMedia = vlc.MediaPlayer(streamServer)
            elif(msg[0] == "HSADDR"):
                ssid.append(msg[1])
            elif(msg[0] == "AUDIO"):
                if(msg[1] == "STARTSTREAM"):
                    try:
                        streamMedia.play()
                    except:
                        streamMedia.stop()
                    finally:
                        streamMedia.play()
                elif(msg[1] == "ENDSTREAM"):
                    streamMedia.stop()
            elif(msg[0] == "RSSI"):
                if(msg[1] == "MEASURERSSI"):
                    rssi = hotspotRSSI(ssid, True)
                    rssi_str = str(rssi)
                    sendMessage(name, "RSSI", rssi_str)
            else:
                continue

    except ConnectionRefusedError:
        print("Connection refused. You need to run server
program first.")
    finally:
        s.close()
```