CO324-Network and Web Application Assignment

VoCe: Voice Conference

Group 02:

E/15/077-Dilhani K.P.W.A.K.K.

E/15/211-Madhuwanthi S.A.I.

E/15/2799-Premathilake L.S.W.S.

Content

1. Abstract

Peer to peer communication is a strategy used in applications such as Bit-Torrent, and Skype. In this Voce assignment, we have to design and code a basic peer to peer voice conferencing application which is similar to Skype. It has two types,

1. Implement voice communication between two parties
2. Extend the application to support multi-party call conferencing

This is a real time application that, when sender is speaking, sound is captured and the message will be broadcast to the connected wifi clients who is connected to the same wifi network.

2. Introduction

An audio conference is simply when a meeting between several parties. The party instigating the audio-conference is known as the calling party and those joining the call are known as the participants(sender and receiver). It is recognizes the best way to increase productivity by reducing time spent out of the office.

Considering a peer-to-peer network, it is much easier to setup than a client-server network. In client program, it can send the message that user is willing to sending (says) to the server and can recognize the results from server. And considering the server program, it is responsible for sending the message content from one client to others and recognize the user who is speaking at that time, by calling the voice recognition system.

There are two parts in audio conferencing system as voice and client-server program. It is provided service for people to talk about something with each other. Here at least two parties (the sender and the receiver) connect each other in the same wifi-network. The system can be provided service for people to talk about something with each other. And also it can automatically recognize the sender, who is speaking and send his name to all other clients. If one computer fails it will not disrupt any other part of the network. It just means that those files aren't available to other users at that time.

3. Implement voice communication between two parties

3.1 System Design

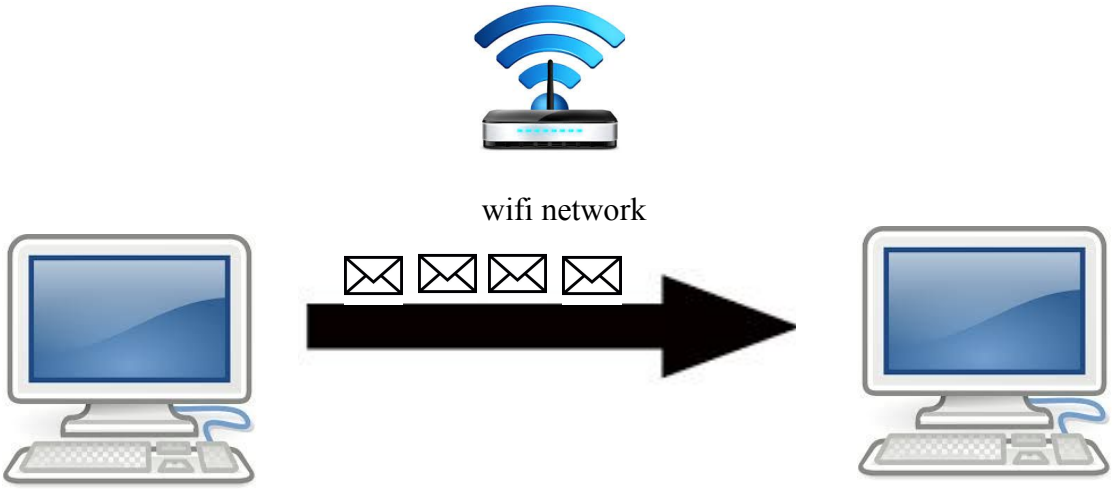

Figure1: Communication between two parties

Program can be compile as following

javac CaptureSound.java

java CaptureSound <receiver IP>

eg:- If client1 : 192.168.1.101

client2: 192.168.1.102

client1 should run ---> **java CaptureSound 192.168.1.102**

```
^Cishani@ishani-PC:~/Desktop/voip/Project 1/new/iteration1$ javac Capture.java
ishani@ishani-PC:~/Desktop/voip/Project 1/new/iteration1$ java Capture 192.168.1.104
The server is ready
Available mixers:
0 Port PCH [hw:0]
1 default [default]
1 Mic is supported!
```

Procedure:

1.When the program is started, sender sends message. It stores in data packets.

2.Then the packets are send to a specified receiver through wifi, who is connected to the same wifi network with sender.

3.The sender should press “ENTER” to send the audio message to the receiver client for capturing. At the same time, receiver also should press “ENTER” for playing the received audio message.Then they can hear what sender is saying.

Here, **captureAndSend()** function is used to record voice and transmit it using a socket. So here we read incoming data packets and capture sound into a temporary Buffer (tempBuffer) and then write data into a byte array using **ByteArrayOutputStream()** interface. Then construct the datagram packet with the length of temporary buffer length to specified port no. on the specified host.Finally, send the packet to other device while closing the voice stream. **captureAudio()** starts to capture audio recordings using the default audio recording application in the device.It allows the device user to capture multiple recordings in a single session.

And in the main method we have used two **threads**, because we want to do different kind of works should be done parallel at one time. They are,

-capture sounds and transmit it to other device

-play record message

In our implementation we have used three files to develope peer-to-peer communication application. Program will be started when run **CaptureSound.java** and it is for record voice and transmit it using a socket. **PlayRecord.java** file is for play back the sounds when program runs.

3.2 Implementation

CaptureSound.java - After record voice and transmit it using a socket.

```
private void captureAndSend() {
   this.byteArrayOutputStream = new ByteArrayOutputStream(); //to write data into byte array
        this.stopCapture = false;
        try {
 int sequence=0;
        while (!this.stopCapture) {
        getTargetDataLine().read(this.tempBuffer, 0, this.tempBuffer.length);//read incoming data
pakkets and capture sound into tempBuffer
         sequence=sequence%16;
                 tempBuffer[99]=(byte)sequence++;
                 System.out.println("Bit "+tempBuffer[99]);
                this.byteArrayOutputStream.write(this.tempBuffer, 0, readCount);//in this    condition
write data into a byte array
 DatagramPacket packet = new DatagramPacket(this.tempBuffer, this.tempBuffer.length,
this.host,55001);  // Construct the datagram packet
                        this.socket.send(packet);// Send the packet to other device
        }
        }
        this.byteArrayOutputStream.close();
        } catch (IOException e){
        e.printStackTrace();
        }
        }
        public void run() {
        try {
        this.socket = new DatagramSocket(this.port);
```

```
this.captureAudio();
this.captureAndSend();//capture sound and transmit
} catch (Exception e) {
e.printStackTrace();
} finally {
this.socket.close();
  }
}

public Capture(InetAddress host) {
        this.host = host;
}
public Capture() {
super();
}
public static void main(String[] args) {  // Check the whether the arguments are given
if (args.length != 1) {
System.out.println("DatagramClient host ");
return;
}

/*used two threads because we want to different kind of works should be done parallel */
        try {
        Thread capture = new Thread(new   Capture(InetAddress.getByName(args[0])));
capture.start();//start capturing
        Thread play = new Thread(new Play());//play record message
        play.start();//start play
        } catch (Exception e) {
        e.printStackTrace();
          }
```

PlayRecord.java - run play back sound

```
public void run() {
        try { // Construct the socket
        DatagramSocket socket = new DatagramSocket(this.port);
        System.out.println("The server is ready");

        DatagramPacket packet = new DatagramPacket(new byte[this.packetsize], (this.packetsize));//
Create a packet
        this.playAudio();
        for (;;) {
        try {
                socket.receive(packet);// Receive a packet (blocking)
                    this.getSourceDataLine().write(packet.getData(), 0, this.packetsize);
        } catch (Exception e) {
                e.printStackTrace();
        }
        }
        } catch (Exception e) {
        e.printStackTrace();}
```

4. Implement multi-cast communication

4.1 System Design

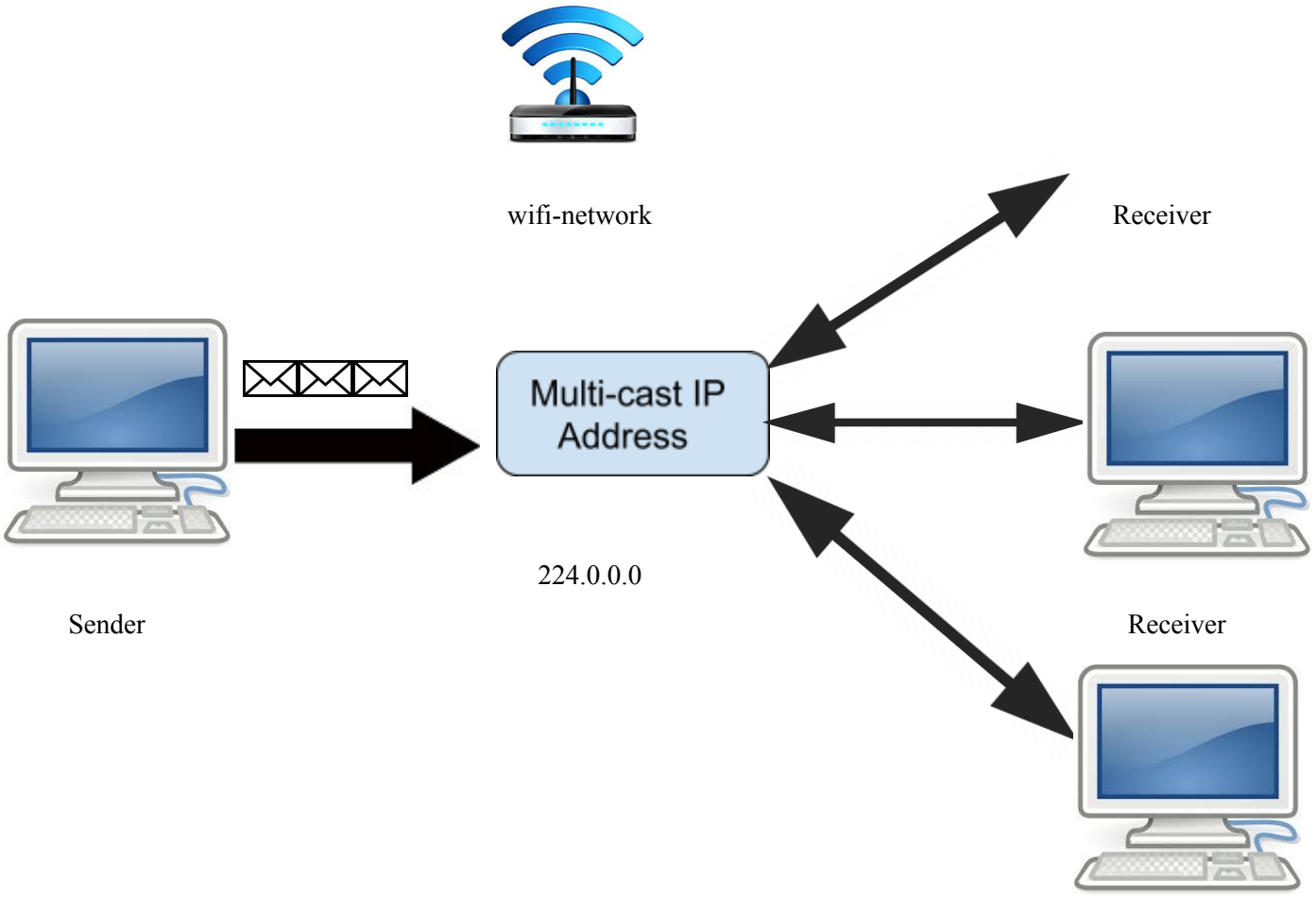

Figure2: Communication between multiple clients

To start the program, when running the program the command line argument should differ from peer-to-peer communication between two parties.

javac Capture 224.0.0.0 <224.0.0.0 is the multicast IP address for every device connected>

```
ishani@ishani-PC:~/Desktop/voip/Project 1/Mulicast/src$ javac PlayRecord.java
ishani@ishani-PC:~/Desktop/voip/Project 1/Mulicast/src$ java PlayRecord 224.0.0.0
Available mixers:
0 Port PCH [hw:0]
1 default [default]
1 Mic is supported!
```

Procedure:

1.When the program is started, sender sends message. It stores in data packets.

2.Then the packets are send to a specified receivers through wifi, who are connected to the same wifi network with sender.(There are multiple receivers)

3.The sender should press “ENTER” to send the audio message to the receiver client for capturing. At the same time, receivers also should press “ENTER” for playing the received audio message.Then they can hear what sender is saying.

Finding multicast IP address:

In computer networking, multicasting is a group communication where data transmission is addressed to a group of destination computers simultaneously. Multicast can be one-to-many or many-to-many distribution. Here, we are implemented for one-to-many communication.

Special addressing must be used for multicasting. These multicast addresses identify not single devices but rather multicast groups of devices that listen for certain datagrams sent to them. In IPv4, 1/16th of the entire address space was set aside for multicast addresses.Since multicast addresses represent a group of IP devices (sometimes called a host group), the full range of multicast addresses is **from 224.0.0.0 to 239.255.255.255** can be used as the destination of a datagram; never the source.

4.2 Implementation

```
private void Transfer() {
        this.stopCapture = true;
        try {
        int count;
        while (true) {
        if (!this.stopCapture) {
                count = getTargetDataLine().read(this.tempBuffer, 0, this.tempBuffer.length);
//capture sound into tempBuffer

                if (count > 0) {

                // Construct the datagram packet
                DatagramPacket packet = new DatagramPacket(this.tempBuffer,
this.tempBuffer.length, this.host, 55001);

                // Send the packet
                this.socket.send(packet);
                }
        }
        }
        } catch (IOException e) {
        e.printStackTrace();
        }
        }

        public void stopCapture(){
        this.stopCapture = true;
```

```
        }

        public void startCapture(){
        this.stopCapture = false;
        }

        public void run() {
        try {
        this.socket = new MulticastSocket();
        this.socket.joinGroup(this.host);
        this.captureAudio();
        this.Transfer();

        } catch (Exception e) {
        e.printStackTrace();
        } finally {
        this.socket.close();
        }
        }

        public CaptureSound(InetAddress host) {
        this.host = host;
        }

        public CaptureSound() {
        super();
        }
}
```

5.Testing

Testing can be done by collecting the statistics by varying UDP packet size, and the netem loss and delay parameters.Network latency and packet loss emulation is a technique you can use to test a program or system on a simulated network connection which is faulty or error prone. This is a valuable tool to make sure your application works when the network has packet loss, jitter or high latency. Loss emulation is a standardized method to simulate a bad network connection using a system on your local LAN. Using the local LAN keeps your testing environment secure and allows full control over the topology.

We have seen many games which are released without this method of testing. They probably developed their game and tested it on their local LAN; a flawless low latency high bandwidth connection. Then when the product was released, the users and reviewers rated it horribly due to its jittery, unplayable on-line model.

We have used Linux network packet loss emulation with netem (Ubuntu) to measure packet loss,to add delays and etc.

In order to use netem, you need to install the iproute package through the Ubuntu package manager.

>>sudo apt-get install iproute

First find an ip address you can ping. As a base line we will ping the router on our internal LAN.

>>ping -c 3 192.168.0.1
As you can see the average latency for pinging the machine was 0.816 ms. This is the expected result of a local LAN. It is a flawless network setup.

Now, lets configure netem using the tc binary to add 250ms of latency to our interface eth0.

>>sudo tc qdisc add dev eth0 root netem delay 250ms

To disable the netem delay products on the interface we can delete the rules.

>>sudo tc qdisc del dev eth0 root netem

Using the properties of the network stated above we can use netem (tc) to emulate this network. Enable the new delay product scheme using the following:

>>sudo tc qdisc add dev eth0 root netem delay 200ms 40ms 25% loss 15.3% 25% duplicate 1% corrupt 0.1% reorder 5% 50%

With this flexibility you could even change to profile so that traffic coming back from the server is faster then the traffic being sent from the clients. You have the flexibility to emulate any topology you need to.
>>sudo tc qdisc add dev eth0 root netem delay 200ms 40ms 25% loss 15.3% 25% duplicate 1% corrupt 0.1% reorder 5% 50%
>>sudo tc qdisc add dev eth1 root netem delay 200ms 40ms 25% loss 15.3% 25% duplicate 1% corrupt 0.1% reorder 5% 50%

6. Conclusion

Now we can send voice messages like mobile system, via Wi-Fi network and receive it by multi-client or group of clients in same Wi-Fi network.