PROJECT REPORT ON
LIVE AUDIO STREAMING FROM SERVER TO CLIENT USING UDP

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

NAME: Harish Pallamala
(Reg. No.: 124003420, B.Tech Computer Science Engineering)

October 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**Live Audio Streaming from server to client using UDP**" submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri Pallamala Harish (Reg. No.124003420, B.Tech Computer Science Engineering**) during the academic year 2021-22, in the School of Computing.

Project Based Work *Viva voce* held on _____

Examiner 1 **Examiner 2**

LIST OF FIGURES

Abbreviations

UDP	User Datagram Protocol
TCP	Transmission Control Protocol
LAN	Local Area Network
IP	Internet Protocol
RTMP	Real Time Messaging Protocol
RTSP	Real Time Streaming Protocol
DNS	Domain Naming Service
VOIP	Voice Over IP
RTP	Real Time Transport Protocol

Abstract

Every time we watch a live stream or audio on demand, streaming protocols are used to transmit data over the internet. Audio streaming from one device to another device can be done using either UDP or TCP over different computer networks.

Using UDP to stream audio or videos is common. It is efficient and any loss of packets or out-of-order packets will cause minimal problems. We will illustrate this technique by steaming live audio. A UDP server will capture the microphone's sound and send it to a client. The UDP client will receive the audio and play it on the system's speakers.

The idea of a UDP streaming server is to break up the stream into a series of packets that are sent to a UDP client. The client will then receive these packets and use them to reconstitute a stream.

In order to illustrate streaming audio, we need to know a bit about how Java handles audio streams. Audio is handled by a series of classes that are found in the javax.sound.sampled package.

KEY WORDS: UDP, Connection, Microphone, UDP regardless of bandwidth constrains, UDP faster and riskier,

Table of Contents

CHAPTER 1

INTRODUCTION

Every time we watch a live stream or radio on demand, streaming protocols are used to transmit data over the internet. They can reside in the application, presentation, and session layers.

Online audio transmission uses both streaming based protocols and HTTP-based protocols. Streaming protocols like Real-Time Messaging Protocol (RTMP) enable faster audio delivery using dedicated streaming servers, whereas HTTP-based protocols use the regular web servers to enhance the viewing experience and quickly scale.

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) are both fundamental components of the internet protocol suite, which lie in the transport layer. The protocols used for streaming sit on to the above mentioned. UDP and TCP differ in terms of quality of the data transmitted and speed at which they can transmit the data.

The main difference between UDP and TCP relies on the fact that it requires a three-way handshake when transporting data. Due to this, TCP is quite reliable and can accommodate for packet loss and ordering. Whereas on the other hand, UDP, starts without requiring any handshake. It transports data without any consideration of bandwidth constrains, making it faster and riskier.

1.1 PROTOCOLS

In computing, a protocol is a convention or standard that controls or enables the connection, communication, and data transfer between computing endpoints. In its simplest form, a protocol can be defined as the rules governing the syntax, semantics, and synchronization of communication. Protocols may be implemented by hardware, software, or a combination of the two. At the lowest level, a protocol defines the behaviour of a hardware connection.

- Detection of the underlying physical connection (wired or wireless), or the existence of the other endpoint or node
- Handshaking (dynamically setting parameters of a communications channel)
- Negotiation of various connection characteristics
- How to start and end a message
- How to format a message
- What to do with corrupted or improperly formatted messages (error correction)
- How to detect unexpected loss of the connection, and what to do next
- Termination of the session and or connection

These are the set of rules that are followed during data transmission and specifies the actions, that are to be taken on sending and receiving data in the network. Examples of some protocols are:

1.1) TCP

1.2) UDP

1.3) IP

1.4) FTP

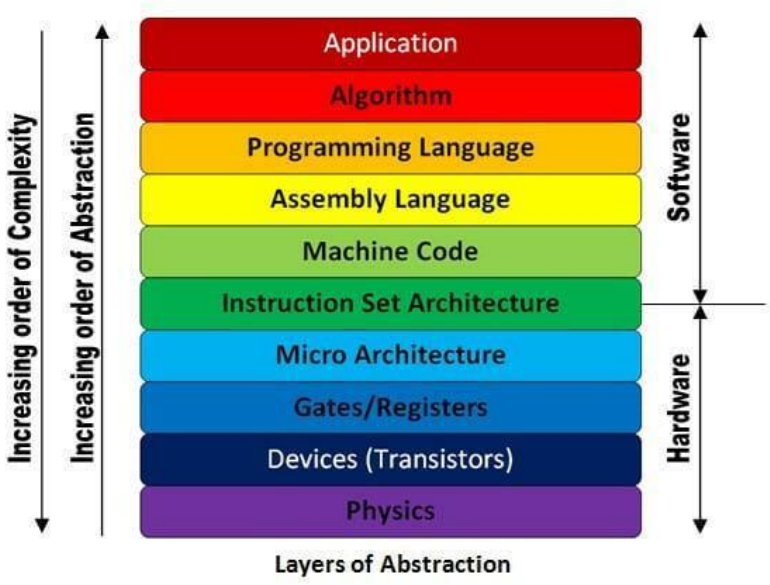
1.5) HTTP

1.6) HTTPS

1.2 LAYERS OF A COMPUTER NETWORK

In The OSI model of communication, the communication between the devices is split into seven layers in which each layer has it’s unique set of tasks to accomplish before the information passes on to the next Layer. Figure 2.1 gives a clear idea about the different layers of Computer Network.

Figure 1.2.1: Layers of a Computer Network



The seven layers of OSI model are:

1.2.i) Physical Layer

1.2.ii) Data Link Layer

1.2.iii) Network Layer

1.2.iv) Transport Layer

1.2.v) Session Layer

1.2.vi) Presentation Layer

1.2.vii) Application Layer

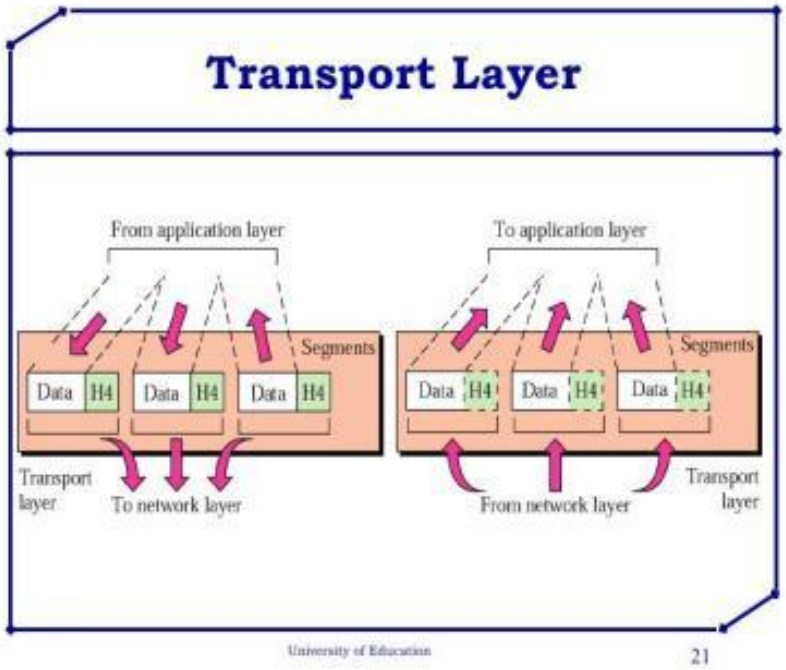
Among all the above given layers, the transport layer is most concerned to this project, hence we shall look into the transport layer below.

1.3 TRANSPORT LAYER

It provides a total reliable end to end communication in the computer network. TCP, IP and many other protocols, rely on transport layer to establish effective communication between two different devices.

Transport layer does the job of decomposing the messages into packets in the receiving end and combining packets into messages in the transporting end. It also controls the rate of flow of packets, in order to make sure that the source does not send the packets at a rate faster than which the destination can handle. The figure 3.1 explains the working of the transport layer.

Figure 1.3.1: Transport Layer



The transport layer consists of two protocols, which are UDP and TCP.

1.4 TCP Transmission Control Protocol

TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation through which applications can exchange data. It is a connection-based transmission protocol that facilitates the exchange of messages between computing devices in a network. It is the most familiar protocol in networks that uses the Internet Protocol (IP), and together they are sometimes referred to as TCP/IP.

TCP is a connection-oriented protocol, thus a connection is established and maintained until the applications at each end have completed exchanging messages. It decides how to break data provided by application into packets that networks can deliver, sends packets to and receives packets from the network layer, manages flow control and because it is meant to provide error-free data transmission, it handles re-transmission of dropped or garbled packets and approves all packets that arrive.

TCP is used for adjusting data in a way that it ensures the secure transmission between the server and client. It assures the integrity of data sent over the network, irrelevant of the amount. Hence, it is used to transmit data from other higher-level protocols that require all transmitted data to arrive.

TCP is important because it provides the rules and standard procedures for the way information is transmitted over the internet. It ensures that data transmission is carried out uniformly, regardless of the location, hardware or software involved, thus it exists as the foundation of the internet. Due to this, it is flexible and highly adaptable, due to which new protocols can be introduced to it and it will accommodate them. It is also non-proprietary, which means no one person or company owns it.

Video streaming protocols in TCP:

1.4.1) HTTP (over TCP/IP)

1.4.2) RTMP

1.4.3) RTSP-1

1.4.4) RTSP-2

1.4.5) HLS

1.4.6) Apple HLS(Lower latency)

1.5 UDP User Datagram Protocol

The User Datagram Protocol, or UDP, is a communication protocol used around the Internet for especially time-sensitive transmissions such as audio playback or Live Audio Streaming or DNS look-ups. It speeds up transmissions by not formally establishing a connection before data is transferred. This allows data to be transmitted very quickly, but it can also causes packets to get lost in transmission.

UDP is a standardized method for transmitting data between two devices in a network. Compared to other protocols, UDP completes this process in a simple fashion: it sends packets (units of data transmission) directly to a target computer, without building a connection first, indicating the order of given packets, or confirming whether they were received as intended. (UDP packets are referred to as ‘datagrams’.)

UDP is commonly used in time-sensitive transmissions where dropping packets sometimes is better than waiting. Voice and video transmissions are sent using this protocol because they are both time-sensitive and susceptible to handle some level of loss. For example, VOIP, which is used by many internet-based telephone services, works on UDP.

1.5.1 UDP Header

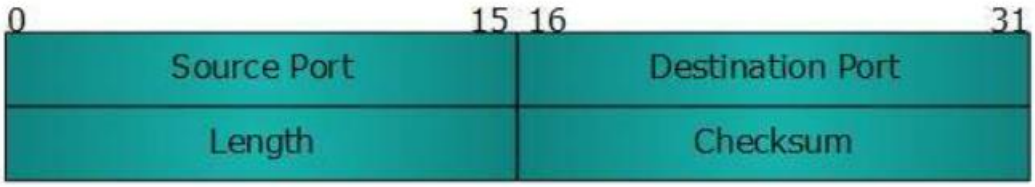


Fig 1.5.1 UDP Header

UDP header contains four main parameters:

- Source Port - This 16 bits information is used to identify the source port of the packet.
- Destination Port - This 16 bits information, is used identify application level service on destination machine.
- Length - Length field specifies the entire length of UDP packet (including header). It is 16-bits field and minimum value is 8-byte, i.e. the size of UDP header itself.
- Checksum - This field stores the checksum value generated by the sender before sending. IPv4 has this field as optional so when checksum field does not contain any value it is made 0 and all its bits are set to zero.

1.5.2 Features of UDP :

- UDP is used when acknowledgement of data does not hold any significance.
- UDP is good protocol for data flowing in one direction.
- UDP is simple and suitable for query based communications.
- UDP is not connection oriented.
- UDP does not provide congestion control mechanism.
- UDP does not guarantee ordered delivery of data.
- UDP is stateless.
- UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

Audio streaming protocols which work on UDP:

5.1) SRT
5.2) WebRTC
5.3) RTSP 3
5.4) RTP

1.5.3 Working of UDP

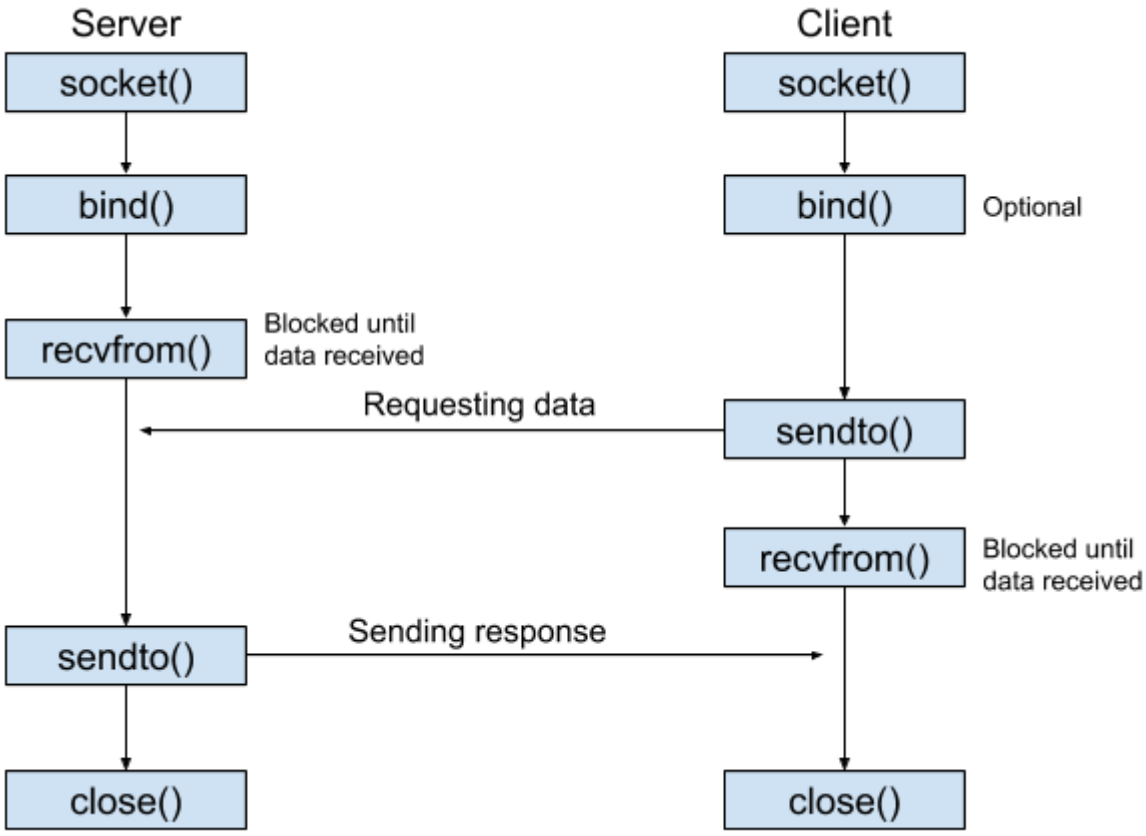


Fig 1.5.3 Client – server Connection Using UDP

UDP uses IP to get a datagram from one computer to another. UDP works by gathering data in a UDP packet and adding its own header information to the packet. This data consists of the source and destination ports on which to communicate, the packet length and a checksum. After UDP packets are encapsulated in an IP packet, they're sent off to their destinations.

Unlike TCP, UDP doesn't guarantee the packets will get to the right destinations. This means UDP doesn't connect to the receiving computer directly, which TCP does. Rather, it sends the data out and relies on the devices in between the sending and receiving computers to correctly get the data where it's supposed to go.

Most applications wait for any replies they expect to receive as a result of packets sent using UDP. If an application doesn't receive a reply within a certain time frame, the application sends the packet again, or it stops trying.

UDP uses a simple transmission model that doesn't include handshaking dialogues to provide reliability, ordering or data integrity. Consequently, UDP's service is unreliable. Packets may arrive out of order, appear to have duplicates or disappear without warning.

Although this transmission method doesn't guarantee that the data being sent will reach its destination, it does have low overhead and is popular for services that don't absolutely have to work the first time.

1.5.4 Benefits of UDP

- It uses small packet size with small header (8 bytes). This fewer bytes in the overhead makes UDP protocol need less time in processing the packet and need less memory.
- It does not require connection to be established and maintained.
- Also absence of acknowledgement field in UDP makes it faster as it need not have to wait on ACK or need not have to hold data in memory until they are ACKed.
- It uses checksum with all the packets for error detection.
- It can be used in events where a single packet of data needs to be exchanged between the hosts.

1.5.5 Drawbacks of UDP

- It is connectionless and unreliable transport protocol. There is no windowing and no function to ensure data is received in the same order as it was transmitted.
- It does not use any error control. Hence if UDP detects any error in the received packet, it silently drops it.
- There is no congestion control. Hence large number of users transmitting lots of data via UDP can cause congestion and no one can do anything about it.
- There is no flow control and no acknowledgement for received data.
- Routers can be careless with UDP. They do not retransmit a UDP datagram after collision and will often discard UDP packets before TCP packets.

CHAPTER 2
SOURCE CODE

The Source code consists of seven .java files

1. **serverGUI.java**
2. **clientGUI.java**
3. stackSender.java
4. stackClient.java
5. streamSender.java
6. streamReciever.java
7. RecordTest.java

RecordTest.java and streamSender.java are in separate folder named Recordtest.

Compilation:

Compile all files by using **javac *.java** command in cmd
Run serverGUI.java using **java serverGUI** command
Run clientGUI.java using **java clientGUI** command in another cmd

Multiple Clients can be connected to server simultaneously

2.1. serverGUI.java

```
import Recordtest.stackSender;

import javax.swing.*;
import javax.swing.border.Border;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

class serverGUI implements ActionListener{
   JFrame frame;
   JButton record,server,send,rec;
   JLabel Recording,started,aud;
   JTextArea txt;
   JTextField tf;
   JScrollPane js;
   serverGUI(){
      frame = new JFrame();
      frame.setSize(400,690);
      frame.setTitle("server");
      frame.setLayout(null);
```

```
        //frame.setLocationRelativeTo();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Recording = new JLabel("ServerInterface");
        Recording.setFont(new Font("Verdana", Font.PLAIN,18));
        Recording.setForeground(new Color(120,80,90));
        Recording.setSize(150,30);
        Recording.setLocation(130,20);
        Border border = BorderFactory.createLineBorder(Color.red);
        Recording.setBorder(border);
        frame.add(Recording);

        started = new JLabel("#ServerStatus#");
        started.setFont(new Font("Verdana", Font.PLAIN,18));
        started.setForeground(new Color(120,80,90));
        started.setSize(350,30);
        started.setLocation(35,113);
        //started.set
        frame.add(started);

        aud = new JLabel("#AudioStatus#");
        aud.setFont(new Font("Verdana", Font.PLAIN,18));
        aud.setForeground(new Color(120,80,90));
        aud.setSize(350,30);
        aud.setLocation(35,145);
        frame.add(aud);

        server= new JButton("StartLiveStream");
        server.setSize(200, 35);
        server.setLocation(94,65);
        server.addActionListener(this);
        frame.add(server);

        Font font = new Font("Monaco", Font.BOLD, 12);
        txt = new JTextArea("\t\t"+ LocalDate.now()+"\n");

txt.append("_____\n");
        txt.append("\tSend Messages to Clients\n");

txt.append("_____\n");
        txt.setFont(font);
        txt.setForeground(Color.red);
        txt.setBounds(18, 190, 350, 380);
        txt.setLineWrap(true);
        txt.setWrapStyleWord(true);
```

```
    js = new JScrollPane(txt);
    txt.setEditable(false);

    frame.add(txt);
    frame.add(js, BorderLayout.CENTER);


    tf = new JTextField();
    tf.setFont(font);
    tf.setBounds(18, 590, 250, 45);
    frame.add(tf);
    send = new JButton("send");
    send.setBounds(275,590,91,45);
    //record.addActionListener(this);
    server.addActionListener(this);
    send.addActionListener(this);
    frame.add(send);

    frame.setVisible(true);
}
public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == send){
        new SwingWorker(){
            @Override
            protected Object doInBackground() throws Exception {
                String str = "";
                DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm:ss");
                LocalDateTime now = LocalDateTime.now();
                String cur = dtf.format(now);
                str = tf.getText();
                tf.setText("");
                txt.append(cur + " : "+str+"\n");
                streamSender.main(cur + " : "+str+"\n");
                //txt.append(string);
                return null;
            }
        }.execute();

    }

    if (ae.getSource() == server) {
        started.setText("Server Started at Port 50005");
        aud.setText("Sending Live Audio . . . ");
        new SwingWorker() {
```

```
        @Override
           protected Object doInBackground() throws Exception {
              stackSender.main();
              return null;
           }
        }.execute();
     }
  }
  public static void main(String args[]){
     new serverGUI();
  }
}
```

2.2. clientGUI.java

```
import Recordtest.RecordTest;

import javax.swing.*;
import javax.swing.border.Border;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class clientGUI implements ActionListener{
  //static clientGUI cli = new clientGUI();
  JFrame frame;
  JButton record,client,stop;
  JLabel Recording,prt;
  static JLabel started;
  static JTextArea area;
  clientGUI(){
     frame = new JFrame();
     frame.setSize(400,690);
     frame.setTitle("Client");
     frame.setLayout(null);
     //frame.getContentPane().setBackground(Color.GRAY);
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

     Recording = new JLabel("ClientInterface");
     Recording.setFont(new Font("Verdana", Font.PLAIN,18));
     Recording.setForeground(new Color(120,80,90));
     Recording.setSize(150,30);
     Recording.setLocation(130,20);
     Border border = BorderFactory.createLineBorder(Color.red);
```

```
        Recording.setBorder(border);
        frame.add(Recording);
        record= new JButton(" 0 Record Live Stream");
        record.setSize(175, 35);
        record.setLocation(110,110);
        record.addActionListener(this);
        frame.add(record);


        stop = new JButton("Stop");
        stop.setSize(120, 35);
        stop.setLocation(220,60);
        stop.addActionListener(this);
        frame.add(stop);

        client= new JButton("ConnectLiveStream");
        client.setSize(150, 35);
        client.setLocation(50,60);
        client.addActionListener(this);
        frame.add(client);

        prt = new JLabel("#Client Status#");
        prt.setFont(new Font("Verdana", Font.PLAIN,18));
        prt.setForeground(new Color(120,80,90));
        prt.setSize(350,30);
        prt.setLocation(35,160);
        //started.set
        frame.add(prt);


        started = new JLabel("#Audio Status#");
        started.setFont(new Font("Verdana", Font.PLAIN,18));
        started.setForeground(new Color(120,80,90));
        started.setSize(350,30);
        started.setLocation(35,193);
        //started.set
        frame.add(started);

        area = new JTextArea("\t\t"+java.time.LocalDate.now()+"\n");

area.append("_____\n")
;
        area.append("\t     Chat is connected with Server\n");
```

```
area.append("_____\n")
;
        area.setBounds(13,240,360,390);
        Font font = new Font("Monaco", Font.BOLD, 12);
        area.setFont(font);
        area.setForeground(Color.BLUE);
        area.setLineWrap(true);
        area.setWrapStyleWord(true);
        area.setEditable(false);
        frame.add(area);

        frame.setVisible(true);
    }
    public static void recv(String s) {
        area.append(s);
        //streamReceiver.main();
    }



    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == record){
            new SwingWorker(){
                @Override
                protected Object doInBackground() throws Exception {
                    RecordTest.main();
                    return null;
                }
            }.execute();
        }
        if (ae.getSource() == client){
            prt.setText("Client started at port:50005");
            new SwingWorker(){
                @Override
                protected Object doInBackground() throws Exception {
                    try {
                        stackClient.main();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return null;
                }
            }.execute();
```

```
        }
        if (ae.getSource() == stop){
            new SwingWorker(){
                @Override
                protected Object doInBackground() throws Exception {
                    stackClient.stop();
                    return null;
                }
            }.execute();
        }
    }
    public static void main(String args[]){
      clientGUI cli = new clientGUI();
      streamReciever.main();
    }
}
```

2.3. stackSender.java

```
package Recordtest;

import javax.sound.sampled.*;
import java.io.IOException;
import java.net.*;

public class stackSender
{
    public byte[] buffer;
    private int port;
    static AudioInputStream ais;

    public static void main()
    {

        System.setProperty("java.net.preferIPv4Stack", "true");

        TargetDataLine line;
        DatagramPacket dgp;

        AudioFormat.Encoding encoding = AudioFormat.Encoding.PCM_SIGNED;
        float rate = 44100.0f;
        int channels = 2;
        int sampleSize = 16;
```

```
boolean bigEndian = false;
InetAddress addr;
int port = 50005;


System.out.println("Server started at port:" + port);

AudioFormat format = new AudioFormat(encoding, rate, sampleSize, channels,
(sampleSize / 8) * channels, rate, bigEndian);

DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);
if (!AudioSystem.isLineSupported(info)) {
   System.out.println("Line matching " + info + " not supported.");
   return;
}
try
{
   line = (TargetDataLine) AudioSystem.getLine(info);

   line.open(format);

   line.start();

   byte[] data = new byte[4096];

   addr = InetAddress.getByName("225.6.7.8");
   MulticastSocket socket = new MulticastSocket();
   while (true) {
      // Read the next chunk of data from the TargetDataLine.
      line.read(data, 0, data.length);
      System.out.println("streaming\n");
      // Save this chunk of data.
      dgp = new DatagramPacket (data,data.length,addr,port);

      socket.send(dgp);
   }

}catch (LineUnavailableException e) {
   e.printStackTrace();
}catch (UnknownHostException e) {
   // TODO: handle exception
} catch (SocketException e) {
   // TODO: handle exception
} catch (IOException e2) {
```

```
            // TODO: handle exception
        }
    }

    public static void stop() {
        return;
    }
}
```

2.4. stackClient.java

```
import javax.sound.sampled.*;
import java.io.ByteArrayInputStream;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;

public class stackClient {

    AudioInputStream audioInputStream;
    static AudioInputStream ais;
    static AudioFormat format;
    static boolean status = true;
    static int port = 50005;
    static int sampleRate = 44100;

    static DataLine.Info dataLineInfo;
    static SourceDataLine sourceDataLine;

    public static void main() throws Exception
    {
        System.out.println("Client started at port:" + port);

        System.setProperty("java.net.preferIPv4Stack", "true");

        InetAddress group = InetAddress.getByName("225.6.7.8");
        MulticastSocket mSocket = new MulticastSocket(port);
        mSocket.setReuseAddress(true);
        mSocket.joinGroup(group);

        /**
         * Formula for lag = (byte_size/sample_rate)*2
         * Byte size 9728 will produce ~ 0.45 seconds of lag. Voice slightly broken.
```

```
         * Byte size 1400 will produce ~ 0.06 seconds of lag. Voice extremely broken.
         * Byte size 4000 will produce ~ 0.18 seconds of lag. Voice slightly more broken
then 9728.
         */
        byte[] receiveData = new byte[4096];

        format = new AudioFormat(sampleRate, 16, 2, true, false);
        dataLineInfo = new DataLine.Info(SourceDataLine.class, format);
        sourceDataLine = (SourceDataLine) AudioSystem.getLine(dataLineInfo);
        sourceDataLine.open(format);
        sourceDataLine.start();

        //FloatControl volumeControl = (FloatControl)
sourceDataLine.getControl(FloatControl.Type.MASTER_GAIN);
        //volumeControl.setValue(1.00f);

        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
        ByteArrayInputStream baiss = new
ByteArrayInputStream(receivePacket.getData());

        while (status == true)
        {
           mSocket.receive(receivePacket);
           ais = new AudioInputStream(baiss, format, receivePacket.getLength());
           toSpeaker(receivePacket.getData());
        }

        sourceDataLine.drain();
        sourceDataLine.close();
    }

    public static void toSpeaker(byte soundbytes[]) {
        try
        {
           clientGUI.started.setText("Receiving Audio.....................");

           System.out.println("At the speaker");
           sourceDataLine.write(soundbytes, 0, soundbytes.length);
           clientGUI.started.setText("Receiving Audio from server");
        } catch (Exception e) {
           System.out.println("Not working in speakers...");
```

```
            e.printStackTrace();
        }
    }

    public static void stop() {
        System.exit(0);
    }
}
```

2.5. streamSender.java

```
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;


class streamSender {
    public static void main(String message) {
        System.setProperty("java.net.preferIPv4Stack", "true");
        try {
            InetAddress group = InetAddress.getByName("225.6.7.8");
                MulticastSocket socket = new MulticastSocket();
                DatagramPacket packet = new DatagramPacket(message.getBytes(),
message.length(), group, 3456);
                socket.send(packet);
                socket.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

2.6. streamReciever.java

```
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

class streamReciever {

    public static void main() {
        System.setProperty("java.net.preferIPv4Stack", "true");
```

```
        try {
            InetAddress group = InetAddress.getByName("225.6.7.8");
            MulticastSocket mSocket = new MulticastSocket(3456);
            mSocket.joinGroup(group);

            while(true) {
                DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm:ss");
                LocalDateTime now = LocalDateTime.now();
                String cur = dtf.format(now);
                byte[] buffer = new byte[100];
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                mSocket.receive(packet);
                clientGUI.area.append(new String(buffer));
            }
        } catch (Exception e) {e.printStackTrace();}
    }

}
```

2.7. RecordTest.java

```
package Recordtest;
import javax.sound.sampled.*;
import javax.swing.*;
import java.io.File;

public class RecordTest {

    public static void main(){
        try{
            AudioFormat audioFormat=new
AudioFormat(AudioFormat.Encoding.PCM_SIGNED, 44100.0f, 16, 2, 4, 44100,
false);
            DataLine.Info dataInfo = new DataLine.Info(TargetDataLine.class,
audioFormat);
            if(!AudioSystem.isLineSupported(dataInfo)){
                System.out.println("Not Supported!...");
            }
            TargetDataLine targetLine = (TargetDataLine)AudioSystem.getLine(dataInfo);
            targetLine.open();

            JOptionPane.showMessageDialog(null, "Click OK to start recording.");
            targetLine.start();
```

```
        Thread audioRecorderThread = new Thread(){
            @Override
            public void run() {
                AudioInputStream recordingStream = new AudioInputStream(targetLine);
                File outputFile = new File("record.wav");
                try{
                    AudioSystem.write(recordingStream, AudioFileFormat.Type.WAVE,
outputFile);

                }catch (Exception e){
                    System.out.println(e);
                }
                System.out.println("Stopped Recording");
            }
        };
        audioRecorderThread.start();
        JOptionPane.showMessageDialog(null, "Clock ok to stop recording.");
        targetLine.stop();
        targetLine.close();

      }catch(Exception e){
        System.out.println(e);
      }
    }

}
```

CHAPTER 3

SNAPSHOTS

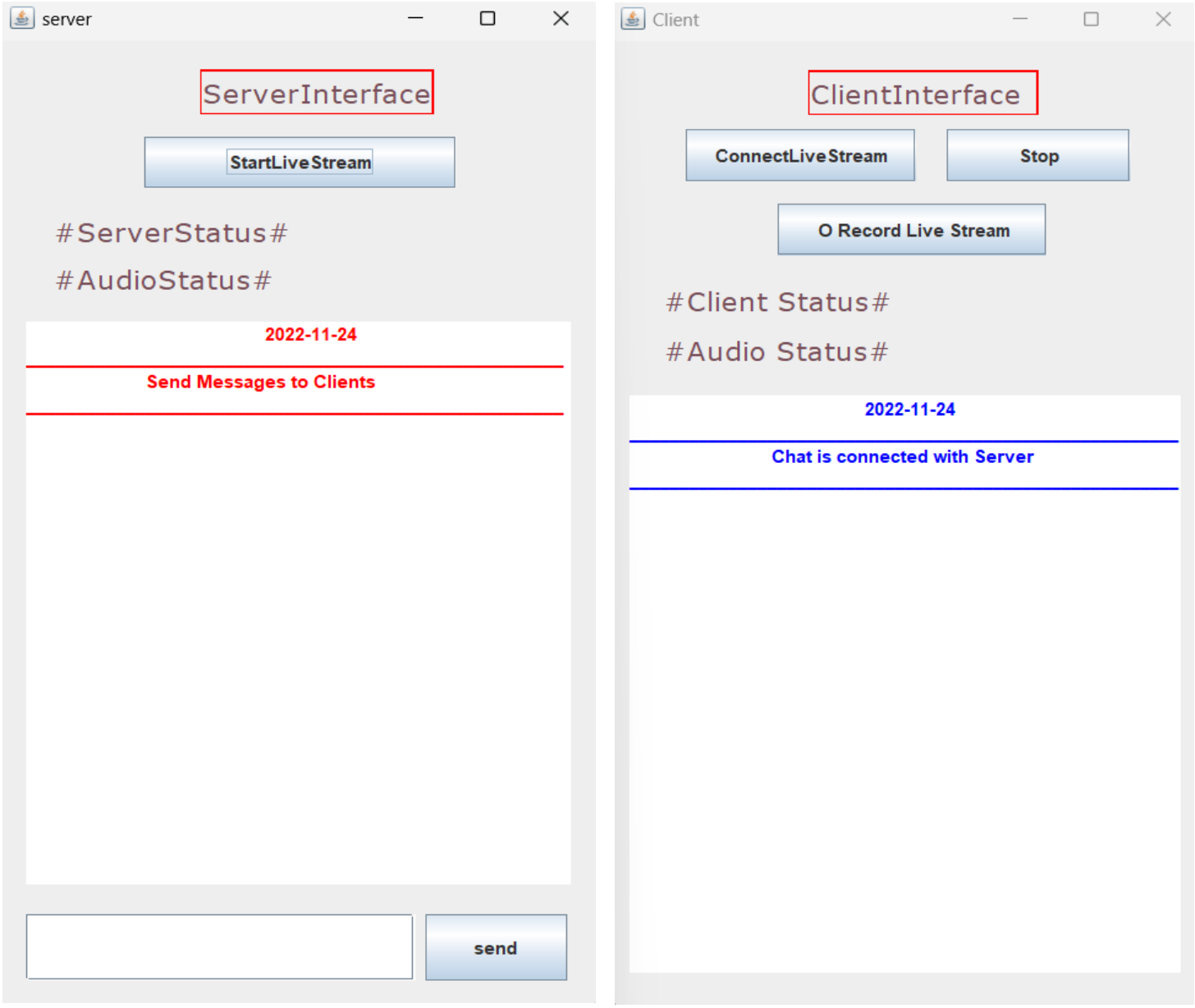


Fig 3.1 Initial Stage of server and client

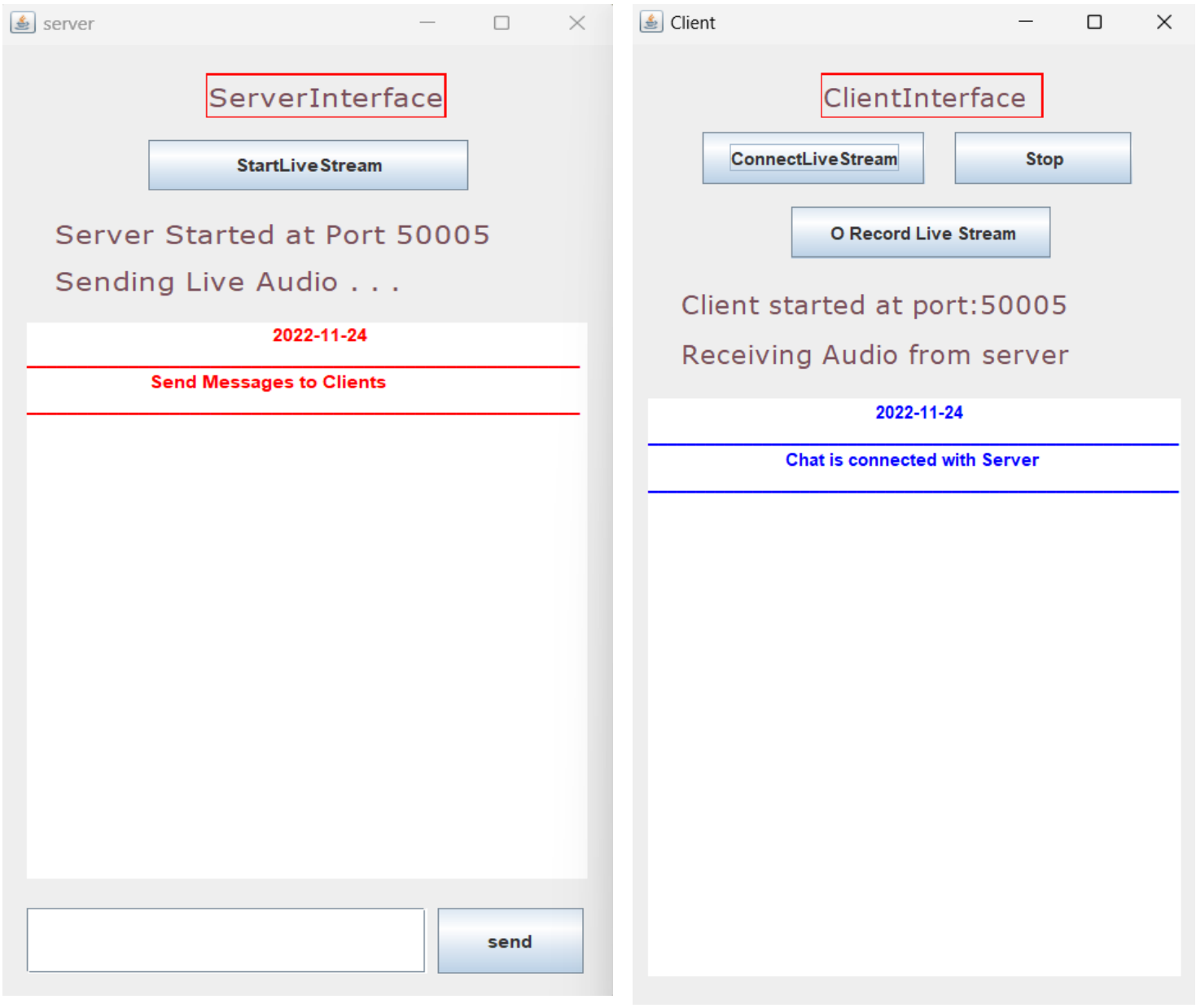


Fig 3.2 Client is now connected with server port:50005
while server sending live audio, client is receiving.

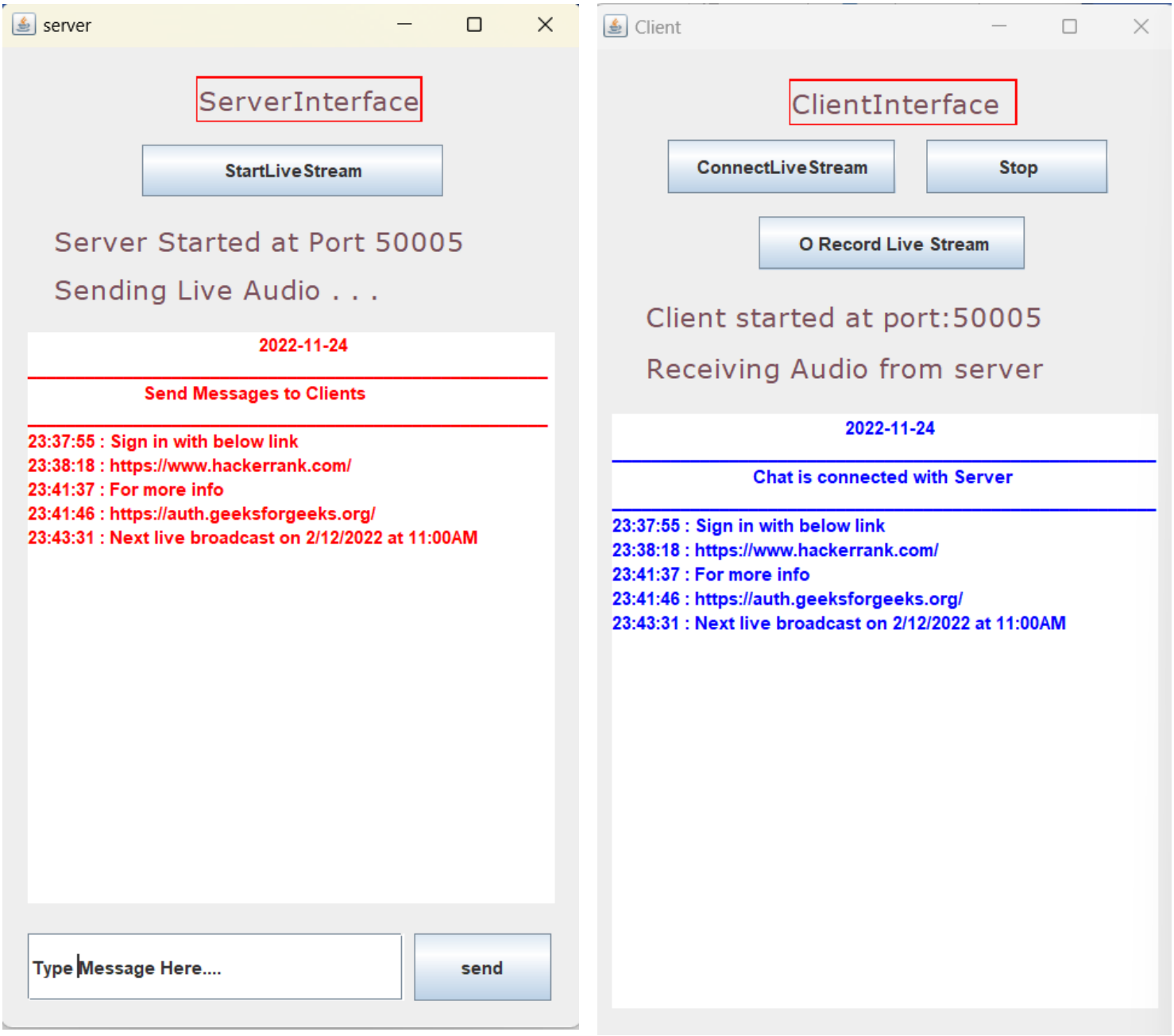


Fig 3.3 server sends messages in chat box while streaming

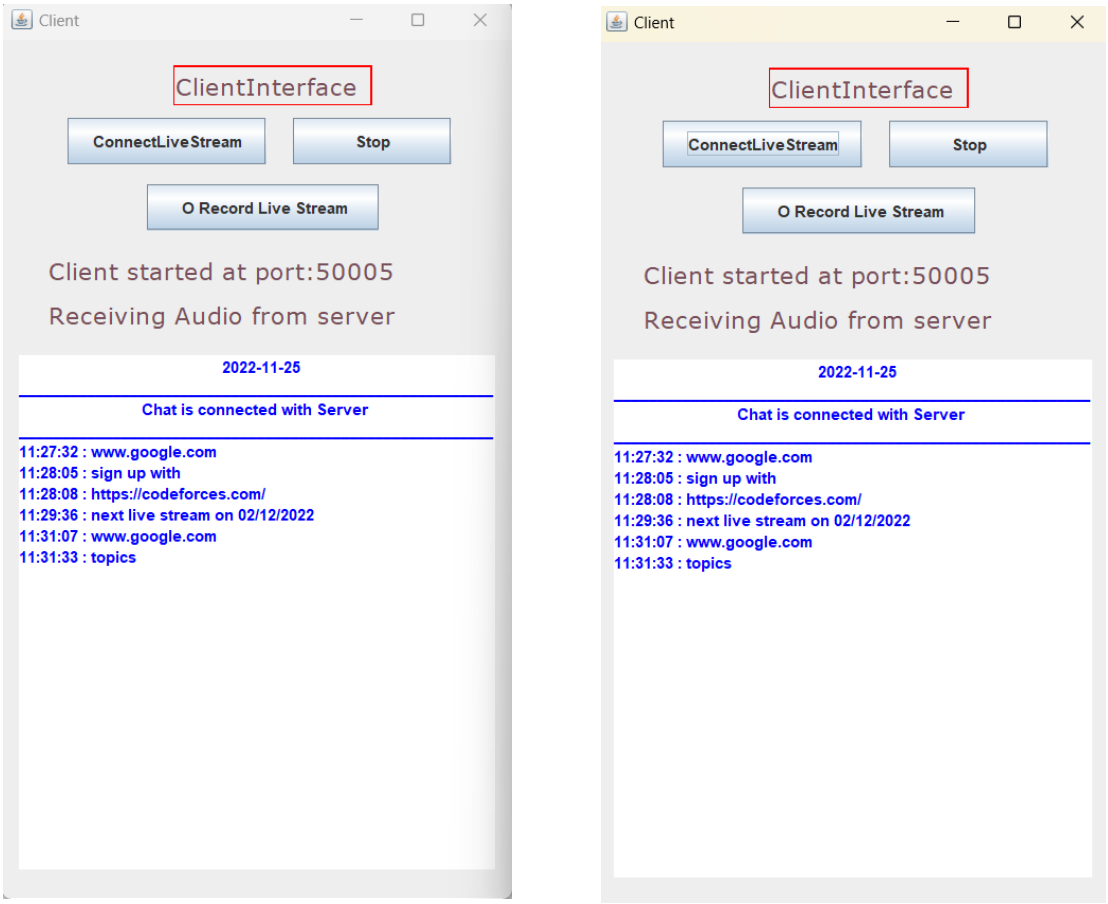


Fig 3.4 Multiple Clients Connected with server initially

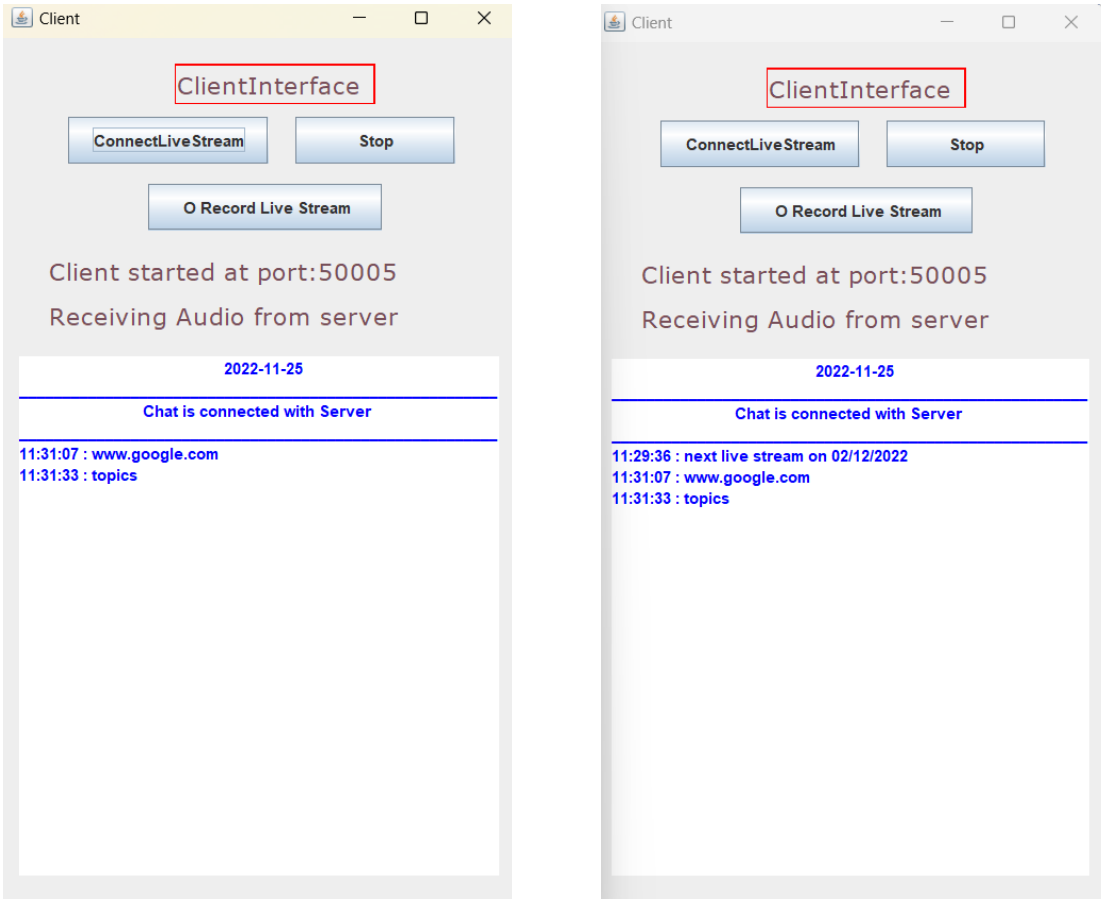


Fig 3.5 Some more Clients joining in middle

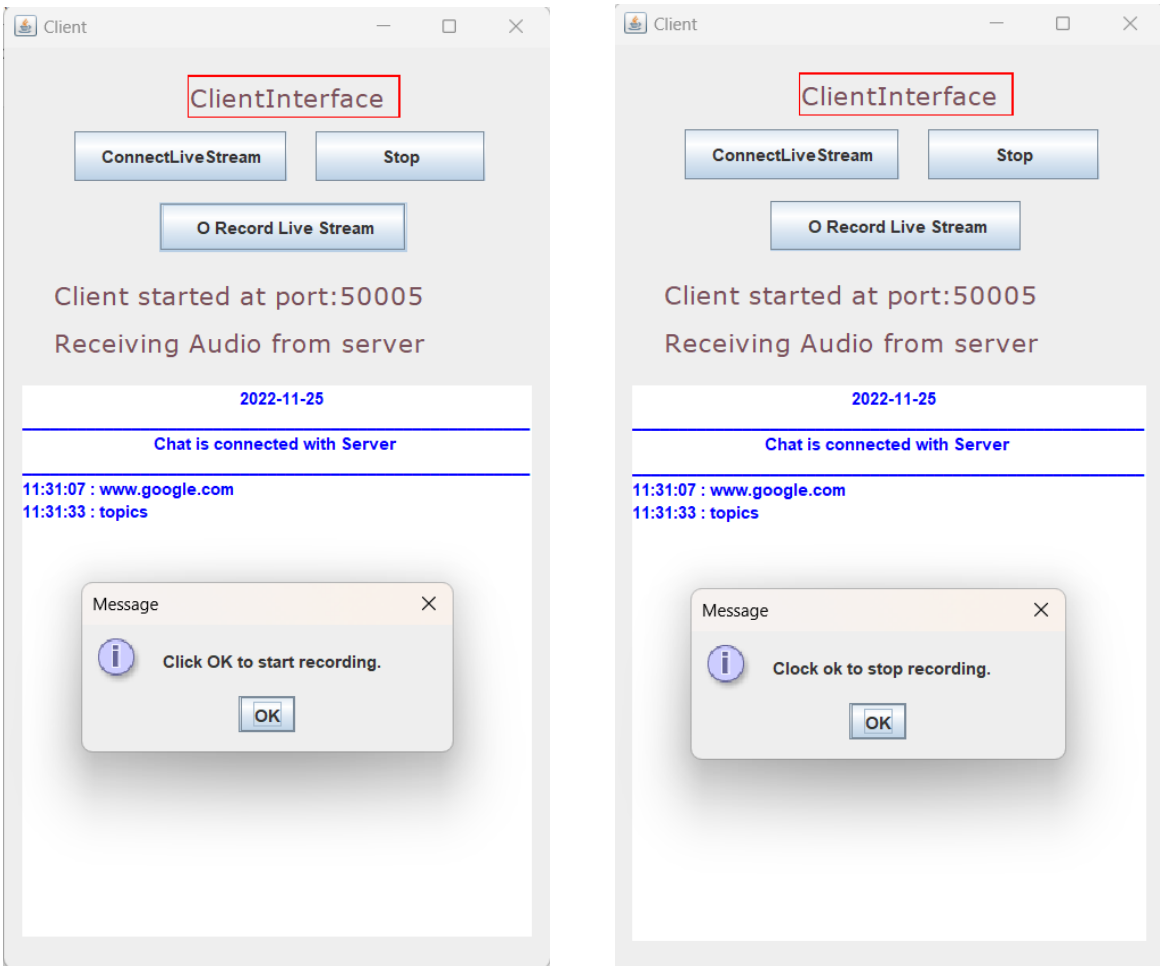


Fig 3.6 Client is Recording the Audio

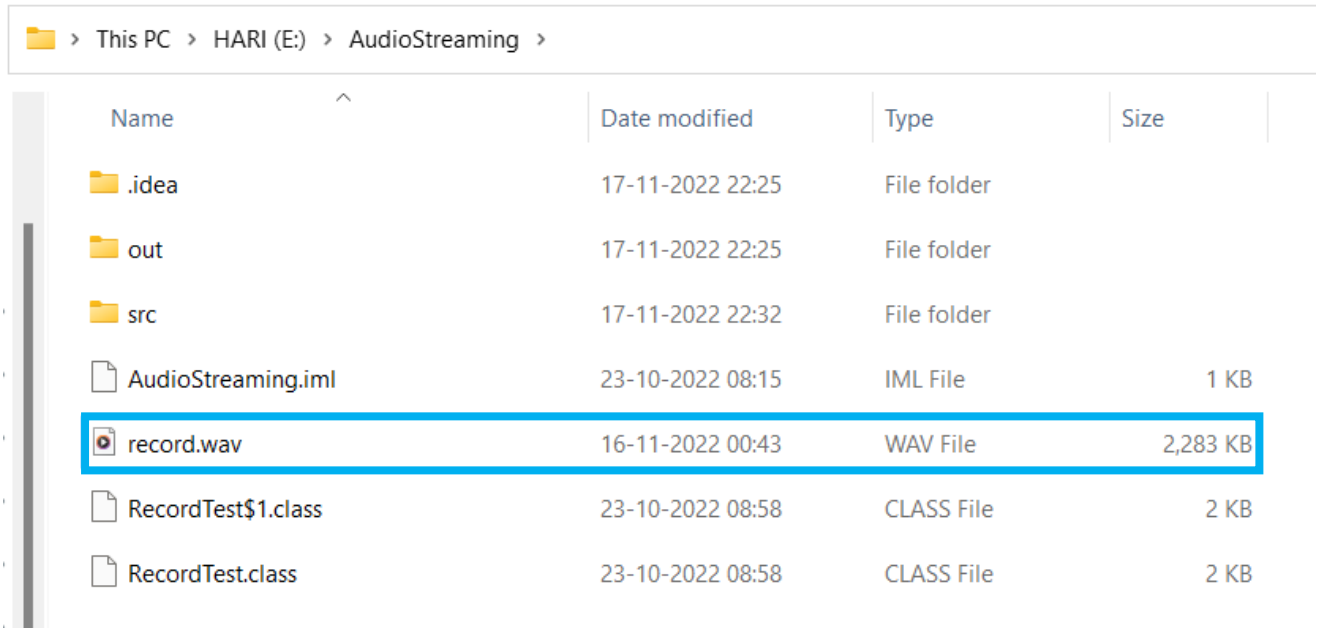


Fig 3.7 Recorded file saved in clients file directory

CONCLUSION AND FUTURE PLANS

In this project, we have seen how audio streaming works in computer networks. We have streamed a audio to the same Computer, using javax.sound package. The audio was streamed using UDP, streamed to the address “225.6.7.8”(local host) and received on “ udp ://@:50005”.

This code can work in different computers with server in one and client on other. We can extend this to online internet streaming. From this, we have concluded that, UDP based stream are faster in response and are more suited for live streaming, as it has a notable delay in response and a minute delay in stream as compared to UDP due to 3-way handshake. But if delay was not an issue and the quality of the audio was preferred over time-sensitivity, then TCP based streaming protocols need to be implemented, because, even though they have a delay in response, the stream quality is better than the one being streamed over an UDP based protocol.

REFERENCES

1) **Hernandez C. I.** (February 24, 2017). *Abstraction Layers: From Atoms to Apps*

2) **McCarty S** (July 29, 2015). Architecting Containers Part1: *Why Understanding User Space vs. Kernel Space Matters*

3) **Anonymous**, Transport Layer Protocols, *Transport Layer, en.wikipedia.org/wiki/Transport_layer*

4) **Jim Kurose**, Keith Ross Authors' website, Differences in TCP and UDP, *Computer Networking: A Top-down Approach - 8th Edition, http://gaia.cs.umass.edu/kurose_ross/online_lectures.htm*

5) *https://stackoverflow.com/*