CS 313 Project 4

Group 31: VoIP Chat Program

Keagan Selwyn Gill: 22804897@sun.ac.za

23/04/2024

Introduction/Overview

For this project, the goal was to implement a chat program where clients can send messages, form groups, send messages and voice notes within those groups and make group calls using Voice over Internet Protocol (VoIP). A VoIP system implementation was therefore also required to enable multiple users to call each other over the local network.

Unimplemented features

Java mixers was not used to remove echoes, distortions, and dead zones in the calls.

Additional features

- Users can change their username and have it update for all users, including on their corresponding GUIs.
- Voice notes recorded by a user can be played back to themself and saved locally to an audio file.
- Voice notes received from other users can be saved locally to an audio file.

Description of files

Source Files:

ServerWindow class

- This class provides a GUI for displaying connected users and user activity information. It is responsible for establishing new connections, and then creating and passing these on to the client connection handler class (ClientConnectionHandler). A new ClientConnectionHandler class is constructed for every client that connects to the server. It handles all the input from clients, including commands, and sends out the necessary data to the desired clients.

ClientWindow class

- This class provides the interface that the user interacts with. It is responsible for connecting to the server and maintaining all input and output from the server. It provides all the infrastructure for calling, voice notes and text messages. The code generated by the Netbeans IDE which handles the GUI is also present in this class. This class also

contains the code, that had to be manually implemented, which handles the operations that occur upon clicking the buttons on the GUI.

sendPacket class

- This class handles the sending of Datagram packets for voice notes and group call audio. It sends byte buffers using UDP.

Program description

The Server

The server coordinates all the user activity. Each user connects to the server via a client. Many clients can connect to the server at the same time concurrently. A new 'ClientConnectionHandler' instance is created for each client that connects. This class-instance of each client controls how the clients connect and disconnect. It also keeps track of the clients' behaviour, the sending and receiving of signals and messages. The signals are TCP packets that are sent where the data is a string prefixed with specific characters to communicate specific commands to the Client and ClientConnectionHandler instances, and suffixed with other information such as the group members, the sender and the message (if there is one). In this way, the ClientConnectionHandler class within the Server class also sets up the connection for group calls and voice notes. It also handles the changing of usernames, broadcasting messages to the global chat, sending group messages and whispering.

The Client

The client provides the interface and the infrastructure to connect to the server, chat in the global chat tab with all the clients that are connected to the server, create a group with other clients by selecting the desired clients on the GUI list, chat in the group/VoIP tab with only their group members, make group calls, record and send voice notes to group members, as well as playing said voices notes and saving them to a file.

The sendPacket class

This class has separate methods for sending Datagram packets for voice notes and group call audio. Two methods are dedicated to sending Datagram packets for voice notes, one called 'send' which divides the data into multiple packets to be sent if the data size exceeds a given buffer length, and the other called 'sendVN' which calls the 'send' method 3 times - first for sending the 'start-of-audio-data' flag, then the audio data and then the 'end-of-audio-data' flag. One method is dedicated to sending Datagram packets for group call audio called 'sendMulti', it just sends the group audio data via the Datagram socket to the given multicast IP and port.

Experiments

Text messaging

Global Chat

I tested the global chat, by connecting to the server with three clients. I sent messages from all users concurrently and monitored the receiver box on all sides.

The messages came through as expected, which shows that my global chat behaved correctly to my expectations.

Group Channel Chat

I tested the group channels by connecting to the server with four clients and creating two groups with two members each. I then sent messages amongst the clients in the groups to confirm that the correct clients were receiving the correct messages, i.e., only receiving messages from the client that they are in a group with.
I found that the correct clients received the correct messages. This shows that my channels work correctly.

Voice notes

Recording and playback

To test this, two clients were connected to the server, and they were both added to a group. To test this correctly, I would need to compare the local playback to the playback on the receiver side. This was done by first recording the voice note and then playing it locally, after which it was sent to the other user and played on their side.

This experiment was repeated 5 times for accuracy, and I determined that my voice notes were sent correctly and the sound quality on both ends were the same. This voice quality did not display any echoes, distortion, or dead zones. This demonstrates that my recording, UDP packet forwarding, and playback was working correctly and exceptionally.

Voice Notes and Text Chat

To test this, I repeated the above recording and playback tests, but also sent messages through the group channel concurrently.

The results of these tests showed me that my threading correctly separated the tasks and allowed voice to be recorded and played back while sending and receiving messages in a group and globally.

Group calls

In these experiments, I had a friend help me test the calls by being the other client(s) in the group.

One on One

This was tested by connecting two clients to the server. The clients were then put into a group and a voice call was started. I then monitored the voice quality and delay on both sides.
In this experiment, we could clearly hear each other talking and could discern separate words, but the quality was compromised by choppy audio. We did note that there were no dead zones, nor was there any distortion. We also experienced a very small delay, but as this delay was smaller than one second, it can be seen as negligible. This showed me that my voice transmission had small issues but functions overall. I was unable to fix these small issues in time for the project demonstration.

Three Clients

This test was almost exactly the same as the one-on-one test, but with another client added to the group.
We found that the audio quality and delay was exactly the same as in the previous test. This demonstrated that my project scaled correctly with the transmission and receiving of audio.

Voice Calls and Text Chat

To test this, I ran the previous two voice call tests, but sent messages concurrently from all clients in both the group and global chats.
The results of this test showed me that my threading of sending and receiving voice was done correctly, as there was no interruption to the voice call on any sending or receiving of messages.

Audio format

In this section, I experimented with different sample sizes and bit sizes to find the clearest and most efficient combination.
From this table, it can be noted that the best audio qualities were found at the sample sizes of 8000 and 44100 with bit size 16. I chose to use 44100 as my sample size 16 bits as my bit size.

Sample size	Bit size	Observed audio quality
8000	8	Constant static
8000	16	Clear quality
16000	8	Slight distortion
16000	16	Constant distortion
44100	8	Slight static
44100	16	Absolutely clear

Table 1: Audio Format Combinations

Issues encountered

I ran into an issue where the client sent their voice note data but the server was not receiving it, so it could not be sent to the other group member(s). I eventually solved this, but then a similar issue happened when trying to group call. Upon starting a call, no data was being sent or received, so you could not hear the other clients talking. The strange thing is that when I initially implemented and tested the group calls between my desktop PC and laptop the calls worked fine but then the next day upon testing again, this issue happened. I realised that it was perhaps due to the program not picking up my microphone audio anymore and therefore there was no data to send and receive, so it hanged. When I switched to testing on two laptops, it worked fine again.

Design

A notable design choice was the algorithms used to record, playback, save, send, and receive audio.

For recording a voice note, the program reads from a TargetDataLine for the audio input into a temporary buffer. Then it writes that temporary buffer to a ByteArrayOutputStream, which is then available for playback and saving it to a file. For transmitting voice for group calls, the bulk of the work is done by an almost identical algorithm. The difference is that instead of writing the temporary buffer to a ByteArrayOutputStream, it calls the 'sendMulti' method from the sendPacket class which sends the buffer to the multicast socket.

For the playback audio algorithm, I created a ByteArrayInputStream attached to the audio output. Then created an AudioInputStream with that ByteArrayInputStream. This algorithm then takes the given audio input stream and iterates through it, writing to the SourceDataLine until the input is exhausted. This is done in a separate thread to a separate SourceDataLine for each instance. For voice notes, it passes the desired ByteArrayOutputStream (the locally recorded voice note or the received voice note) into the playAudio method. For voice calls, it writes each packet of audio data that it receives separately and sequentially to the ByteArrayOutputStream and then plays it by calling the playAudio method so that it appears as if there is non-stop transmission.

Compilation and execution

Since this project was built using the Apache Netbeans 14 IDE, it cannot be compiled and run from the command-line/terminal. The project must be first opened and then compiled and ran from within the Apache Netbeans IDE. To do so, follow these steps:

1. You must have the Apache Netbeans 14 IDE installed on your computer.
2. Open the IDE.
3. Move your cursor to the top left of the application and click on "File" and then "Open Project" (or use the keyboard shortcut Ctrl+Shift+O).
4. Navigate to the directory of your desired project that you want to open.
5. Select the project folder that contains the folders generated by Netbeans upon creating a project, namely "nbproject" and "src". In this case the project folder is called "VoIP-Chatroom2" and it should have a little coffee-cup-on-a-saucer icon to

the left of it indicating that it is a Java project. Once selected click on “Open Project” at the bottom right of the window.

6. To run the Server, first click on the ServerWindow.java file to open it, **click on any line of code in the file (so that Netbeans knows that this file is the file you want to run), then either left click to open a drop-down list and click on “Run File” or use the keyboard shortcut Shift+F6**. Now the built-in terminal will be displayed and the GUI window for the Server will also be displayed.
7. To run the Client, first click on the ClientWindow.java file to open it and then repeat the process in step 6 above that is **bold**. Now the built-in terminal will be displayed and the GUI for the Client will then be displayed.

Libraries used

- java.io.ByteArrayOutputStream
- java.io.DataInputStream
- java.io.DataOutputStream
- java.io.IOException
- java.io.PrintStream
- java.net.DatagramSocket
- java.net.InetAddress
- java.net.ServerSocket
- java.net.Socket
- java.sql.Timestamp
- java.util.ArrayList
- java.util.Arrays
- java.net.DatagramPacket
- java.io.ByteArrayInputStream
- java.io.File
- java.io.InputStream
- java.net.MulticastSocket
- java.util.List
- javax.sound.sampled.AudioFileFormat
- javax.sound.sampled.AudioFormat
- javax.sound.sampled.AudioInputStream
- javax.sound.sampled.AudioSystem
- javax.sound.sampled.DataLine
- javax.sound.sampled.SourceDataLine
- javax.sound.sampled.TargetDataLine