Project 3: VoIP Chat Program

GROUP 7

Group Members & Roles

Thaakir Fernandez (group leader)
26479443@sun.ac.za

- Sending audio for calls (AudioSender.java).
- Assisted group members with debugging.
- Assisted in the integration of all classes.

Priyal Bhana
27040607@sun.ac.za

- Receiving audio for calls (AudioReceiver.java)
- README.md

Gideon Daniel Botha
26894319@sun.ac.za

- GUI integration to the backend
- Backend group chat creation

Sulaiman Bandarkar
24234850@sun.ac.za

- Backend of voice notes (VoiceNotes.java, VoiceNoteTransfer.java)
- Experiments (VoiceNoteSender.java, VoiceNoteReceiver.java)
- Makefile

NOTE: This project uses our code from project 1 (A Client-Server Chat Program). It is also based on Project 2 (Reliable Blast User Datagram Protocol (RBUDP)) for UDP and TCP implementation needed for Project 3. We build onto our previous code from the last two projects to implement VoIP calls and voice notes.

Project Introduction/Overview

This project implements a Voice over Internet Protocol (VoIP) chat application with both server and client components. This program enables voice calls between clients alongside text messaging capabilities. It also allows users to send pre-recorded voice notes and listen to them through the GUI. This report outlines the implementation features, provides descriptions of our source files, explains the program flow, lists the experiments conducted, the design of the project and finally any issues our group experienced during implementation.

AI Declaration

Priyal Bhana

I found the concept of audio mixing to be difficult. I first used the resources provided to me within the project specification. I then used Chatgpt and Claude.ai as tools to help me understand the theory behind how audio data is received in both private and group calls. ChatGPT explained the idea of how jitter buffers are beneficial for smoothing out the audio data. Claude.ai helped me understand the purpose of a mixing algorithm, and this helped me to build my own logic and framework to then use as a basis when beginning to implement the calls.

Gideon Daniel Botha
I used ChatGPT to get a better understanding of how the Objects and EventHandlers that JavaFX provides function in relation to each other, as well as multithreading with the JavaFX application thread..

Thaakir Fernandez:
Used ChatGPT to explain how audio works in Java, how they are read in, and to give examples of its implementation. I used Codeium to help clean up any spelling or grammar mistakes in my Javadocs.

Sulaiman Bandarkar
Used Claude.ai to understand the concept of how voice notes can be recorded and how audio would work for voice notes.

Features

All features mentioned in the project specification are implemented.

***Additional features*:**

- Custom group chat creation, along with custom names for the groups.

File Descriptions

ServerGui.java
Provides a simple GUI that displays the actions of users connected to the server, as well as all currently connected members

ClientGui.java
GUI for the client to interact with, providing an interactive client list where you can whisper from, then from the different interactive chat windows, you can initiate the different types of voice note sharing and calling.

Client.java
This class handles the client-side functionality for a VoIP chat application, handling both text messaging and voice communication. It maintains a socket connection to the server for messaging, manages audio calls using separate sender and receiver threads, and incorporates voice note capabilities with recording, storage, and playback features. The class includes functionality for user authentication, call management (including group calls), message handling, voice note recording and transfer, group chat management and with appropriate error handling and resource cleanup.

ClientHandler.java
This class manages communication between a server and an individual client. It handles global broadcasting, private messaging (via @username), group creation and messaging, and IP-based voice call setup (private, group, and global).

AudioSender.java
This class handles sending audio data for private and group calls using a UDP implementation in Java. Audio is recorded through a TargetDataLine using a 16000 sample rate, 16-bit sample size and one channel. In private calls, packets are sent via a DatagramSocket, buffered, and sent in sequence. For group calls, it loops through the receivers and sends them all the same packets.

AudioReceiver.java
This class handles receiving and playing audio data for private and group voice calls using UDP in Java. A jitter buffer (TreeMap) to reorder incoming audio packets by sequence number. Audio is played through a SourceDataLine using 16000 samples per second, 16-bit sample size and 1 channel. In private calls, packets are received via a DatagramSocket, buffered, and played in sequence. For group calls, it maintains a separate jitter buffer for each sender (by IP address). Audio from different senders is received, buffered, and mixed together by averaging their 16-bit samples before playback.

VoiceNotes.java
This class manages the recording, storage, and playback of voice notes in the VoIP application. It implements audio capture using the system microphone at 8000Hz with 16-bit mono format, stores recordings as WAV files in a user-specific temporary directory, and provides thread-safe recordings of voice notes. The class handles recording in a separate thread to avoid blocking the main application, manages a collection of saved notes, and automatically cleans up temporary directories when the program exits.

VoiceNoteTransfer.java
This class contains methods for transferring voice note files between clients over TCP connections, separate from the main chat protocol. It implements buffered binary file transfer using 4KB chunks, supports data transmission including sender username and file size information and creates temporary files for received voice notes. The class handles all necessary socket communication and I/O operations for reliable voice note exchange between users.

VoiceNoteReceiver.java
This class is for experiment 1. It serves as a test utility for receiving voice notes over a network connection, operating on port 9876. It continuously listens for incoming connections, receives voice note files using the VoiceNoteTransfer class, and saves them with unique timestamps to an "experiment_received" directory. The class tracks statistics on received files providing a simple experimental framework for measuring voice note transmission performance in the VoIP application.

VoiceNoteSender.java
This class functions as the companion testing utility to VoiceNoteReceiver, designed to evaluate voice note transmission performance. It creates synthetic voice notes of various durations (30, 60, 300, and 600 seconds), sends them to a specified IP address over TCP, and collects transmission statistics including file size and transfer time. The class performs multiple trials and produces a summary report of average transmission times for each duration.

Program Description

Client-Server Interaction

- The server accepts clients, then assigns them their own personal ClientHandler that runs on its own thread.

- The Clienthandler is an extension of the server.
- The server maintains a list of connected clients and their IP addresses.
- The Clienthandler routes text messages, call requests, and other control messages between clients.

Text messaging

Global Messages:

- The client enters a message in the global chat input field.
- The message is sent to the ClientHandler.
- The Clienthandler broadcasts the message to all connected clients.
- Each client displays the message in their global chat area.

Private Messages (Whispers):

- The client selects a recipient from the user list.
- The message is sent to the ClientHandler with the recipient information.
- The ClientHandler routes the message to the specified recipient.
- The recipient displays the message in a dedicated whisper window.

Group Chat Messages:

- Groups are created by selecting multiple users and providing a group name.
- The ClientHandler maintains group membership information per client.
- When a client sends a message to a group, the ClientHandler routes it to all group members.
- Recipients display the message in the corresponding group chat area.

Voice Notes

Recording:

- The client starts recording by clicking "Start Voicenote".
- The VoiceNotes class captures audio from the microphone.

Sending:

- Once the recording has stopped, it is saved to a temporary .wav file.
- The Clienthandler retrieves the IP address of the recipient(s) and this is relayed back to the client.
- The client establishes TCP connections to the IP(s) of the recipient(s).
- VoiceNoteTransfer sends the voice note file directly to each recipient using TCP.

Receiving:

- The receiving client accepts the incoming voice note connection.
- The voice note is saved to a "received_voice_notes" directory.
- The recipient can play the voice note through the GUI.

Voice Calls

Call Initiation:

- Client initiates a call by clicking the "Call" button.
- A call request is sent to the ClientHandler with the recipient's username.
- The ClientHandler verifies the recipient is available to call and sends call setup information to both parties.

- Both clients receive IP addresses and port numbers for direct communication

Audio Transmission:

- AudioSender starts capturing audio from the microphone.
- Audio data is broken into packets, sequenced using sequence numbers, and sent via UDP.
- AudioReceiver receives packets, reorders them using the jitter buffer based on the sequence numbers, and plays them.
- Sequence numbers ensure proper ordering of the audio data packets that are sent via UDP.
- For group calls, the AudioReceiver maintains separate jitter buffers for each sender. Audio from all participants is mixed together before playback.

Call Termination:

- Either party can end the call by clicking "Stop Call".
- The server notifies the other participant(s) that the call has ended.
- Audio transmission threads are stopped, and resources are released.

Client Disconnections

- The client clicks on the “Client disconnect” button
- The Server removes the client from the active user list.
- The Clienthandler broadcasts the updated user list to all remaining clients.
- The server logs the disconnection in its activity log.
- Any temporary resources associated with the client are cleaned up.

Experiments

Experiment 1: Voice Note Transmission

Question: How does the voice note file size affect the transmission time?

Hypothesis: The larger the voice note file is, the longer it will take to transmit.

Independent variable: How long the voice note is in seconds.
Dependent variable: The time it took to transmit the voice note in milliseconds.

Method: VoiceNoteReceiver.java is run by the receiver to listen for voice notes and receive them. VoiceNoteSender.java creates voice notes of different lengths to send. Iterates through them to send and collect the data to be printed out.

Results:

Duration of voice note (seconds)	Time it took to transmit (seconds)
30	3.426
60	8.539
300	43.099
600	87.209

Conclusion: From the above results, we see that as the file size increases, the time it takes to successfully transmit and receive the voice notes increases. Therefore, the longer the voice note is, the longer it takes to send and receive it. In conclusion, we see that our hypothesis is correct.

Experiment 2: Impact of Audio Sampling Rate on Call Quality

Question: How does changing the audio sampling rate (Hz) affect the quality of a call?

Hypothesis: The larger the sampling rate is, the better the quality of the call will be.

Independent variable: The sample rate in Hertz (Hz).
Dependent variable: The quality of the call on a scale of 1-10.

Method: Used private call for this experiment. Changed the sampling rate after each call and rated the call.

Results:

Hertz (Hz)	**Person 1 rating (out of 10)**	**Person 2 rating (out of 10)**
8000	6	7
16000	7	8
24000	5	5.5
32000	5	5
64000	3	3

Conclusion: From the above results, we concluded that a sample rate of 16000Hz gave us the best call quality. For 8000Hz, the quality was clear, but we found 16000Hz to be clearer and had smoother audio. When we increased the sample rate from 24000Hz to 64000Hz, we experienced more drops in audio data and heard more frequent dead zones. The hypothesis is correct to a certain extent, as sound quality reaches an optimal value after which sound quality decreases.

Experiment 3: Effect of Jitter Buffer Size on Call Quality.

Question: How does the jitter buffer size affect audio quality?

Hypothesis: The larger the jitter buffer, the better the quality of the call.

Independent variable: The jitter buffer size (number of packets).
Dependent variable: The quality of the call on a scale of 1-10.

Method: Used private call for this experiment. Changed the jitter buffer size after each call and rated the call.

Results:

Jitter buffer size (number of packets)	Person 1 rating (out of 10)	Person 2 rating (out of 10)
3	7	8
5	8	9
8	8	9
10	8.5	9.5
20	7.5	8

Conclusion: From the above results we concluded that the jitter buffer size should be set to 10 packets as this yielded the best audio quality. As we increased the jitter buffer the audio quality improved but once it reached an optimal value of 10 packets, the quality of the call began to degrade. The hypothesis is correct to a certain extent as sound quality reaches an optimal value after which sound quality decreases.

Issues Encountered

- *Priyal:* I found it difficult to improve the voice quality of the group calls and correctly implement the mixing algorithm to get the audio of the group calls to play smoothly without everyone's voices sounding distorted.
- *Sulaiman*: My headset and microphone were not working properly. My group members said that when I am on a call with them, there is increased static. Even when I used a different headset, it improved the quality of the call, but there was still static in the background. So it might be an issue with my laptop, but I am not sure.

Design

Instead of having a direct connection between the client and the server, we implemented the ClientHandler as a medium or "middleman" to handle specific interactions between the clients. Thus, instead of the server handling IP addresses and establishing calls and messages, the ClientHandler does this instead.

Gideon:

- Chose to flip the text and event handlers of some buttons on the GUI (like the voicenote and call buttons) instead of having 2 separate buttons for starting and stopping.
- Private calls are started from within the whisper windows instead of from the client list.