Computer Science 354
Group 35:
Project 4 Report

Lyndon Fooy – 21028540
Rivaldo Ponnadu – 20691815

Report Overview
In this project we were tasked with creating a single-server-multiple-client VOIP communication program.
In this report we share how we developed and implemented the program as well as what the program can and cannot do as well as a brief analysis of the data structures and designs implemented throughout development.

Unimplemented Features
We have developed a fully functional system across local clients. However, when we ran functionality tests across the Hamachi service we encountered major issues. As a result, calls and voicenotes across Hamachi are not functional. We did encountered minor sporadic successes while testing call functionality however it is far from reliable enough to say the feature was implemented successfully.

Additional Features Implemented
Clients can be added or removed from conference calls at any point in the call. As well as each call group having its own dedicated chat. Clients can interact via this medium whether or not they are part of the group call. Voice notes can be paused while playing and resumed from where they were paused.

Description of Files
We have used 7 java classes to create our program.

1. Client.java – The class handling client-side interactions such as initiating calls, messaging users or chats as well as the recording and listening of voice notes.
2. ClientGUI.java – The GUI layer handling the interaction between the user and the Client class.
3. ClientHandler.java – Handles interactions between the Client class and the server. Implemented to be run as a thread by the server with one handler per client on the server.
4. LoginGUI.java – Used to gather appropriate server information from the user to establish a connection to the server.
5. Server.java – The class managing the server handling the interactions between the various connected clients. This includes the passing of messages, establishing call connections and data lines for voicenotes to be sent over.
6. ServerGUI.java – A helpful GUI to display all relevant server information in a presentable manner for a user.

Program Description

The VoIP application we created is java based and makes use of the Swing library for error handling and the javaFX library for the GUI implementation. The program works by starting a server after which an instance of the client class must be run. The client will need to specify the server IP, the port number to connect on and provide a unique username. The client will be accepted if the server exists and the chosen username has not been taken already. After the client is accepted the server runs a client handler thread to handle interactions between the client and server. The client may then access the functionalities of the program. This includes sending messages which can be seen on the messages pane. Users can also send private messages (whispers) here. From the voicenotes pane users can send and listen to received voicenotes and finally from the calls pane users can participate in group calls or chats. There are 256 possible groups to join on the server all numerically indexed from 0 to 255.

Experiments

- Creating a conference call
- Sending messages while in call
- Playing a voicenote
- Sending global, group chat and whisper messages while on call
- Sending multiple voicenotes

Issues Encountered

We encountered issues with the program's functionality over the Hamachi client as all the functionality can be utilised between two or more local clients. When attempting to send voicenotes over the Hamachi network the receiver would not be able to access the voicenote even after they had received the data for it often freezing their client in the process. As for calls over Hamachi we found limited success in that we could sometimes hear each other for a short period of time in the call. We were unable to rectify either of these problems.

Significant Data Structures

We made use of the Enum structure to create a lexicon of commands for communication between the client and server. This helps ensure the server cannot receive invalid commands. A linked list is also used to hold information about the groups on the server. Finally, we used the hash set to record the usernames of clients who have connected to the server as a means of ensuring there are no two users with the same name.

Compilation

Using terminal locate the group-35 file of the project and enter the command "make". This will create all relevant java classes. Removing the created classes can be done by typing the "make clean" command into a terminal in the group-35 file.

Execution

Using the terminal locate the group-35 file and compile the classes using the commands explained in Compilation. Then run an instance of the server using the command “make server”. The server GUI should then pop up. In a separate terminal also in the group-35 file type the command “make client”

Design

Our program was written in Java and we used a Makefile to compile and run both the client
and the server. We implemented JavaFX for both the server and the client GUIs. In order to separate the sending and receiving of voicenotes, as well as calls, we used so that the client could be in a call and still send and receive messages. When running either the server or the client, details such as the hostname, port number and username needed to be specified. A user could leave out the hostname and port number, in which case the hostname would default to localhost and the port would default to 1234, but the username was required.

Libraries

We used the Java sound library to handle recording and playback of sounds. Clips were used for voice notes, and thus they can be replayed. Multicast sockets were used for calls,
with a TargetDataLine for playback and a SourceDataLine for recording.

Execution

To allow a client to connect to the server, the server must be started before the clients are
started. The following two commands need to be run in two separate terminals:

In terminal 1:

$ make server

This will start the server and listen on port 1234 for any clients wanting to connect or send
messages.

In terminal 2

$ make client

This will start a new client and prompt the client to log in.