# VOIP CHAT PROGRAM

**October 11, 2019**

Gerrit Burger - 21261687
Jacques Huysamen - 20023669
Computer Science 354 - Networking

# Contents

# 1 Project Description

## 1.1 Introduction

Voice over IP or VoIP is a popular method for the delivery of voice communications over Internet Protocol networks. It enables users to have voice conversations over the Internet. IP packets are used to send voice data between users, instead of hardware switching circuits used in older telephony methods. Global telephone networks use VoIP to enable users to have voice conversations between users all over the world.

## 1.2 Problem statement

We were instructed to implement a VoIP system that would enable users to initiate voice conversations with other users over the local LAN. A client-server model was used to handle communication between multiple connected users. We used Python to code this project.

# 2 Project Features

## 2.1 Client Features

Required features included for the client:

- Ability to connect to a server to initialize communication.
- Disconnecting/reconnecting to a server without incident.
- Ability to display a list of all currently connected users, updated from the server.
- Unique nickname for each connected client.
- Ability to send text messages to users while in a call.
- Conference calls, with the ability to speak to multiple clients concurrently in a call.
- Conference channels, with the ability to leave and enter channels.
- The ability to send pre-recorded voice messages to other connected clients.
- The ability to call other connected clients on a private call.
- Calls use Real-time Voice transmission
- High voice quality.
- Voice transmission uses the UDP protocol.
- A GUI to enable the client to:
  - See text messages received from other clients.
  - See a list of currently connected users.
  - Select users and enter text to send text messages to other clients.
  - Record voice messages and select users to whom the voice message should be sent.

## 2.2 Server Features

Required features included for the server:

- Handle various requests from clients concurrently.
- Act as a mediator and proxy for the clients.
- Send updated user lists to all clients when clients disconnect or a new client connects.
- Ability to handle illegal actions from clients.
- A GUI that contains the following:
  - A windows to display all client activity
  - A area that displays a list of all the currently connected clients.

# 3 Program Description

## 3.1 Server

### 3.1.1 Initialization

When the server program is run and a new instance of a server is created the local machine's IP address is found and set as the server's IP address. The server's GUI will open and a thread is spawned that will wait for clients to connect to the server. A pool of ports is created ranging from 5000 to 10000, each time a client connects a unique port will be assigned to the client to be used to create a TCP socket between the client and the server. This port number will be used to identify clients. A pool of multicast IP addresses are also created, python's multicast IP range is 224.0.0.0 through 230.255.255.255, each time a group call or channel is created one of these IP addresses are assigned to the call or channel to enable clients to send and receive data to other clients in the call or channel. Each time a client connects to the server a thread is also spawned to handle voice notes for that client.

### 3.1.2 Client connection

When a new client connects to the server the server's connection thread will accept the clients' request to create a TCP socket. The TCP socket object is stored in a dictionary to be used again later by the server. A port number is popped from the array of ports and assigned to this client by storing the port number in a dictionary to enable the server to identify clients at a later stage. After the client has entered a username on the client's GUI the username is sent to the server. The server checks to see that the username is not in use, if it is in use the server notifies the client and the client will be prompted to enter another username. If the username is not in use the username is stored by the server and the client is notified that the connection was successful. When a new client is connected the server will send the updated user list to all currently connected users to update the list on their GUIs. Each time a client connects to the server a thread is spawned that will handle all activity of that user on the server's side.

### 3.1.3 Handling client requests

When a client wants to send a message or start a call the client will send a message to the server to be relayed to the recipient of the call or message. String identifiers were used to distinguish between different requests from clients. When the server receives a message from a client the server will identify the type of operation the client has requested. The client that wants to start the call or send the message will send an array of information to the server including the message to be sent or channel to be joined as well as the recipient of the call or message. The server will use this information to get the recipient's TCP socket and relay the message to the recipient. In the case of a call, the recipient will be prompted to accept or reject the call. The recipient's choice will then be sent back to the server, the server will then again relay this message back to the original sender of the request.

When a client wants to send a voice note to another client the sound is recorded on the client's side and sent to the server along with the recipient of the message. The client's voice note thread on the server will receive all the sound data and store it in an array. The recipient's TCP socket is retrieved from the dictionary using the recipient's username and the voice note is sent on this socket to the recipient.

## 3.2 Client

### 3.2.1 Initialization

When the client program is run the GUI will start and the client will be prompted to enter the IP address and port number of the server the client wishes to connect to. If these values are valid and the client can successfully connect to the server the client will be prompted to enter a username to be used. If the entered username is already in use by the other user connected to the server the server will notify the client and the client will be prompted to enter another username. Once a valid username has been entered the main GUI will open and all the needed sockets connections between the server and client will be initialized. A list of all the users currently connected to the server will be sent to the client and will be shown on the client's GUI.

### 3.2.2 Sending of text messages and voice notes

When a client wants to send a text message to another client and no text is entered or no recipient is selected the server will notify the client and an error message will be displayed. If a text is entered and a client is selected this information will be sent to the server. The server will relay the message to the recipient and the text message will be displayed on the GUI.

When a client wants to send a voice note, a recipient client must be selected before the voice note recording button is pressed. When the button is then pressed the sending client's voice will be recorded until the button is pressed again, when the button is pressed for the second time recording stops and the sound data is sent to the server. The server will then relay this data to the recipient and the recipient's GUI will notify the client that a new voice note has been received and display it on the voice note window where it can be played.

### 3.2.3 Calls and conference calls

When a one to one call in initialized, the client has to select another client from the online user list and press the call button. When the button is pressed a request is sent to the server, the server will replay this request to the recipient. The recipient client's GUI will display a pop-up message where the call can be accepted or rejected. The recipient client's choice to accept or reject the call is sent to the server and the original client who initiated the call is notified. If the call was accepted 2 threads are spawned on each client's side, one to record and send sound data and one to receive sound data and play it. When the call button is pressed again by one of the clients the server is notified and will replay the message to both clients to send the call, thus killing the threads and closing the socket connections.

When a client wants to initiate a conference call one or more other clients are selected from the online user list and the conference call button is pressed. The server is notified of all the clients to connect to the conference call, the server will send a request to each client. The server assigns a multicast IP address for the conference call each client creates a TCP socket that binds to this IP address. Each client will spawn a thread that will record sound and send the sound over the multicast socket to all other clients in the conference call. For each other client in the conference call, a thread is spawned to receive sound data on the multicast socket, each thread will only play sound received from a single client to enable clients to hear sound from all clients connected to the conference call at the same time.

### 3.2.4 Chat channels

When a client creates a chat channel the server is notified of the name of the channel. The server assigns a multicast IP address to this channel and all connected clients are notified of the new channel. All the clients will update their channel list on the list of channels. Then when a client wants to join a channel the server is notified and a thread is spawned to record and send voice data to all other clients connected to the channel. When another client joins the channel the server is again notified and all clients connected to the channel are notified. A thread is again spawned to receive and play voice data for each client connected to the channel.

# 4 Data Structures

Since we used python all of the data structures are in the form of dictionaries because their key-value pairing makes it very easy to work with stored data and retrieving necessary data from them.

## 4.1 Data Structures

### 4.1.1 Server Data Structures

On the server-side of the project a variety of data structures were used to keep track of data about users, their sockets, addresses, etc. and many other data like voice notes, channels and their IPs, etc.

In regards to users and their data the following data structures were used: A dictionary with key values being a certain client's TCP socket and the values being their IP. A dictionary with key values being a certain client's IP and the values being their chosen nickname. A dictionary with key values being a certain client's TCP socket and the values being their chosen nickname. A dictionary with key values being a certain client's chosen nickname and the values being their IP. A dictionary with key values being a certain client's IP and the values being their port on which their sockets are connected.

In regards to conference channels and their data the following data structures were used: A dictionary to store a normal conference call's IP popped from a pool of generated IPs between 224.0.0.0 and 230.255.255.255 (Which allows multicasting over UDP sockets) and the selected users in the conference call. A dictionary to store all of the created conference channels with the key being their name and value being their multicasting IP popped from the pool of available IPs generated from the same pool as the previous dictionary. A dictionary storing all of the clients connected to a certain channel, the key being the channel name and the value being a list of connected users' IPs.

In regards to voice notes: When a client connects to the server a socket is also created through which voice notes are separately sent from the other data being sent to the clients.

A dictionary was created to link the voice note sockets and the port used for TCP sockets across the server.

### 4.1.2 Client Data Structures

On the client-side, we have two dictionaries and one list. The first dictionary is to store all of the IPs of the people either connected in a conference call with the client or the same channel as the client and a boolean to specify whether the client should be accepting sound from a certain client over the call, for example when a user leaves a channel the client should no longer receive any sound from that user. The other dictionary is used to store voice notes that must be added to the GUI, the dictionary thus connects the name of the voice note to the bytes of sound received. Each client also has a list of users currently in their conference call or channel.

# 5 Experiments

## 5.1 Voice quality by changing chunk size

### 5.1.1 Hypothesis

As the chunk size increases the quality of the sound recorded by the sender and received by the receiver will increase.

### 5.1.2 Method

In this experiment, chunk size refers to the number of bytes to be recorded by the sender's microphone to then be sent in a packet to the receiver. A sample audio clip of a siren was downloaded to be used in this experiment, 3 different chunk sizes were used to test the effect of the chunk size on the quality of the sound. For each chunk size, the audio clip was played externally on the sender's side and a call was initiated between the sender and the receiver. The sound data received by the receiving client was stored and written to a .wav file. Each of these .wav files were then used to create spectogram graphs displaying the sound frequency over time. Graphs were also created for the original sound file to be used to compare to the graphs of the received files.

### 5.1.3 Results

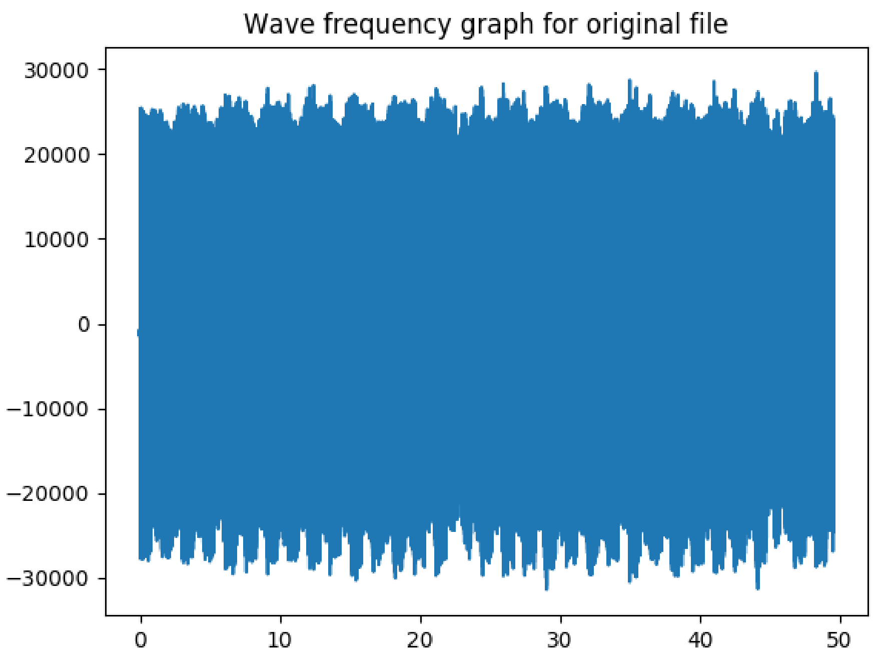


**Figure 5.1:** Spectogram graph for the original sound fille

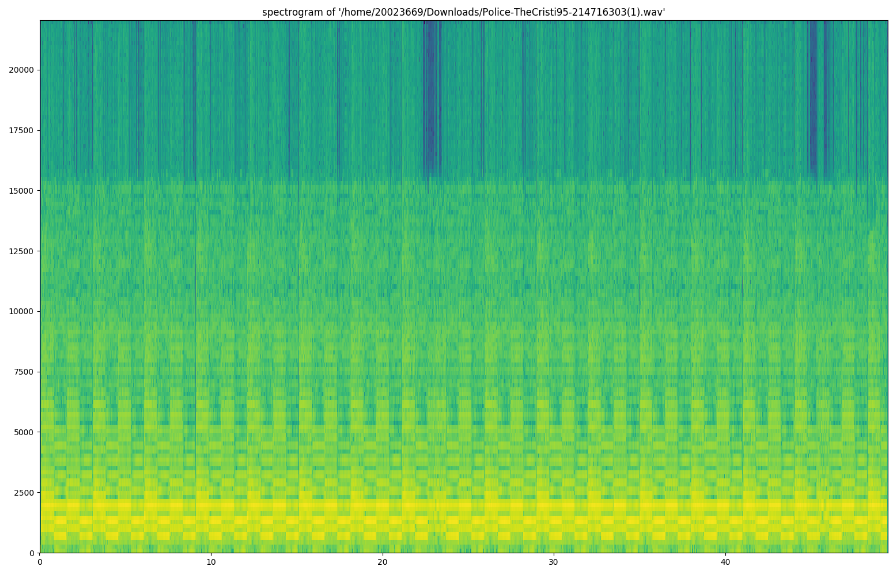


**Figure 5.2:** Spectogram graph for the original sound fille

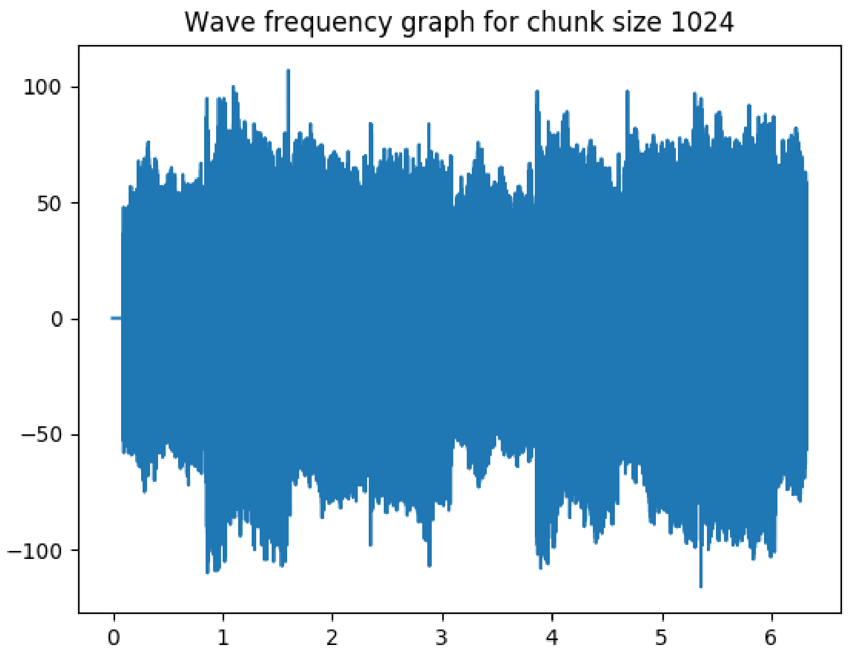


**Figure 5.3:** Spectogram graph for the received file with chunk size 1024

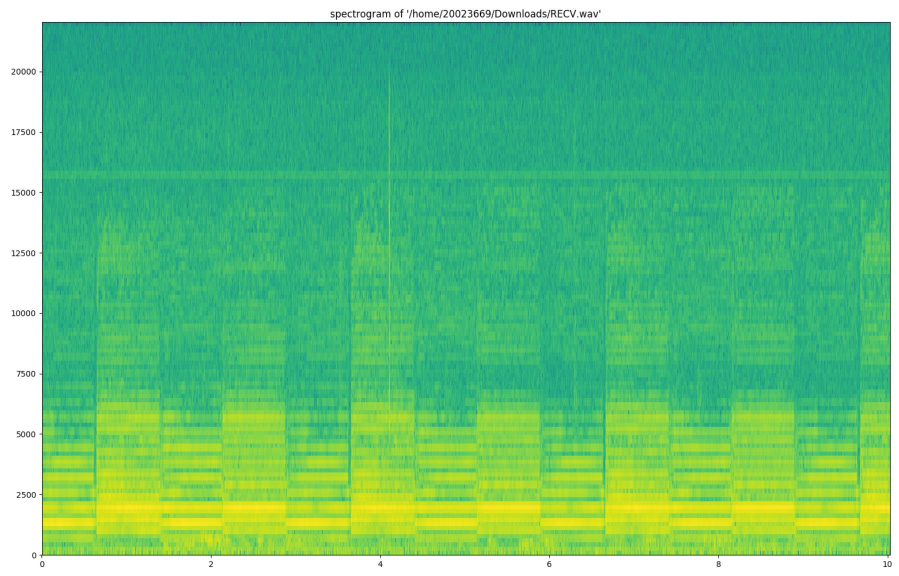


**Figure 5.4:** Spectogram graph for the received file with chunk size 1024

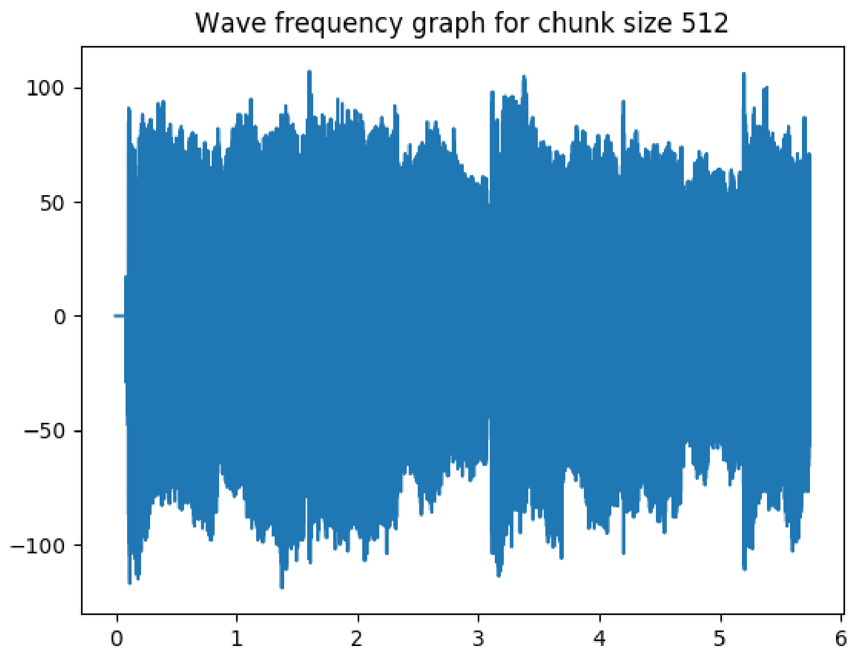


**Figure 5.5:** Spectogram graph for the received file with chunk size 512

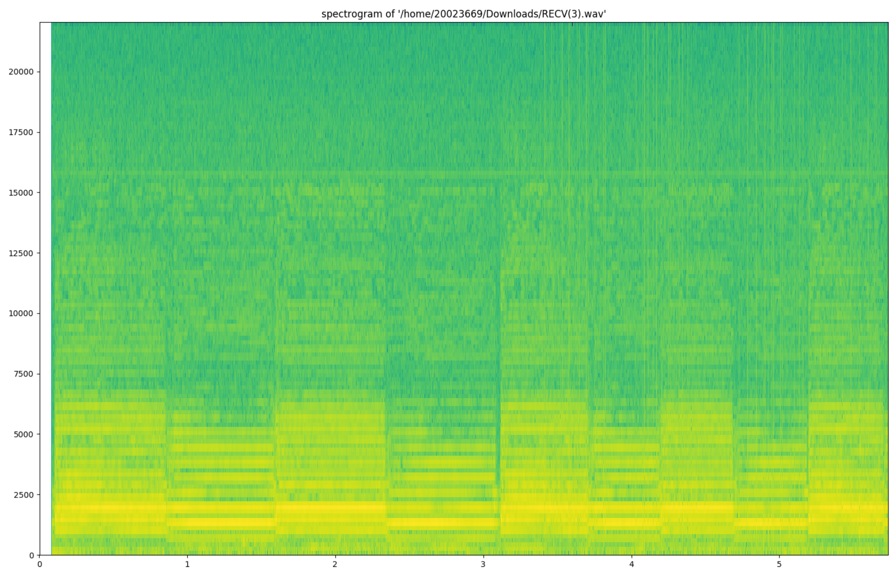


**Figure 5.6:** Spectogram graph for the received file with chunk size 512

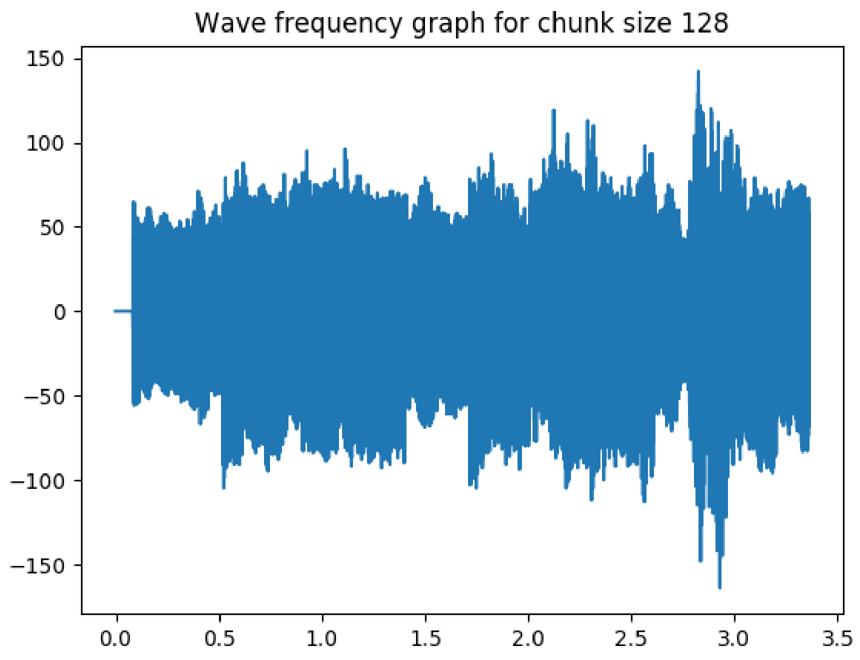


**Figure 5.7:** Spectogram graph for the received file with chunk size 128

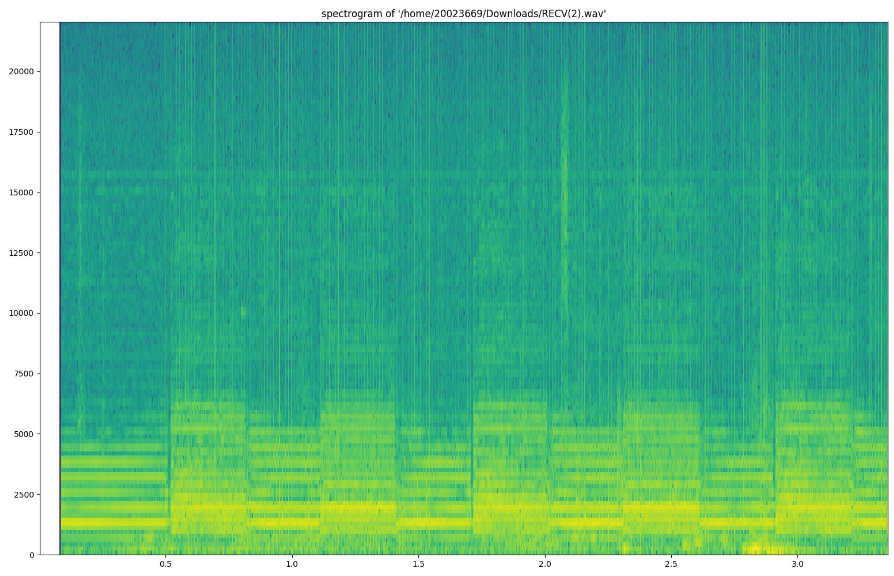


**Figure 5.8:** Spectogram graph for the received file with chunk size 128

### 5.1.4 Conclusion

As seen from the graphs above as the chunk increases the quality of the sound increases. The graphs of the received sound file with chunk size 1024 is the closest to the original file's graphs. The graphs of the smaller chunk size files were not able to distinguish very high and very low frequencies. The hypothesis can be accepted.

## 5.2 Making and Receiving consecutive calls

### 5.2.1 Hypothesis

Consecutive calls will not influence the sound quality or cause the program to terminate.

### 5.2.2 Method

Upon creating the server and connecting two clients, calls were initiated and closed consecutively for 1, 5 and 10 times and then replicated with three clients, where they randomly called each other and terminated the calls a few seconds after receiving the call and answering.

For two clients, let's call them A and B, A always called B. Firstly A called B normally, then A called B 5 times where the call was closed immediately closed after receiving but left on for 5 seconds on the last call and the same for calling 10 times.

For the three clients calling the same procedure as above was followed, but there was no order in which one client called another.

### 5.2.3 Results

The reason for leaving keeping the call open on the last iteration of the calls was to see whether the call's quality was effected, for example threads not closing correctly upon disconnecting from a call which would result in the client receiving distorted sound, echos, etc. or receiving the same sound from a client more than once.

Upon running the experiment for two clients it was clear that there is no effect on the sound quality of the calls received by the clients or the stability of the program when calling consecutively. The same goes for when more than two clients called one another consecutively.

### 5.2.4 Conclusion

The program is stable when clients call each other consecutively. Hence, the hypothesis is accepted.

## 5.3 The stability of conference calls

### 5.3.1 Hypothesis

The program will be stable when multiple users are in a conference call and will remain stable after terminating the conference call.

### 5.3.2 Method

Multiple clients connected to the server (around 5). After the clients connected a conference call was initiated between all of them, the call remains open for a certain amount of time, around 5 seconds, where clients speak to each other on the call.

The call is terminated and a conference call with all the clients is then initiated again to test the stability of the sound that each client receives and to test the stability of the program to ensure that it does not terminate unexpectedly.

### 5.3.3 Results

The conference call between the clients worked perfectly, with minimal delay in the sound between clients, distortions, echoing or dead zones.

After terminating the conference call the program remained stable.

Running the conference call consecutively did not affect the stability of the program or the quality of the group calls, the delay remained minimal and so did the distortion, there was no echoing and dead zones as well, meaning the threads closed without any problems and also did not duplicate.

### 5.3.4 Conclusion

Upon the end of the experiment, it is clear that conference calls do not affect the stability of the program or the quality of the sound received over conference calls. Hence, the hypothesis can be accepted.

## 5.4 Stability of conference channels

### 5.4.1 Hypothesis

Multiple conference calls, leaving and joining channels will not affect the stability of the program or the quality of the sound received over channels.

### 5.4.2 Method

Multiple clients connected to the server and multiple channels were created. After the creation of the channels the multiple clients joined and left different channels consecutively all at the same time.

Every time clients join channels the speak to the other clients connected to their channel to test whether the quality of the sound over the channel remained stable, simultaneously stress testing the program to see if the program remains stable after joining and leaving channels.

After each client joined and left each channel all of the channels were deleted. Once more a channel was created and all of the clients connected to the channel to test the stability of the program and quality of the sound over the channel. Then the channel was deleted and all of the clients terminated their program.

### 5.4.3 Results

The creation of multiple channels did not terminate the program and the program remained stable.

When a client joined or left a channel the program remained stable and the quality of the sound received over the channels was not affected.

After deleting all of the channels and creating a single channel once more, all of the clients connected to that channel and the program remained stable, the quality of the sound received over the channel also was not affected. Then the clients terminated their programs without any errors.

### 5.4.4 Conclusion

Having multiple conference channels, joining, leaving and deleting them did not affect the stability of the program or affect the quality of the sound received over the channels. Hence, the hypothesis can be accepted.

## 5.5 The effect of sending, receiving and playing voice notes on the stability of the program

### 5.5.1 Hypothesis

Sending, receiving and playing voice notes happen concurrently with private calls, conference calls and sending messages and will not affect the stability of the program or the quality received over calls.

### 5.5.2 Method

First, three clients connected to the server, two clients initiated a private call and then the third client proceeded to send voice notes to both of the other two clients that are in a private call. Then each of the two clients in the private call listened to the voice notes while in their private call.

Secondly a conference call was initiated between two of the three clients and the third client sent a voice note to both of the clients connected to the conference call. The two clients in the conference call proceeded to listen to the voice notes while in the conference call.

Thirdly two clients sent multiple voice notes to one another while also sending messages to one another while receiving and playing the received voice notes.

### 5.5.3 Results

The two clients in the private call received the voice notes sent from the third client without any interrupts to the private call or any effect on the stability of the program or the quality of the sound received over the private call. The clients in the private call was also able to listen to the voice notes while concurrently being in their call without affecting the sound received from one another.

The effect of sending, receiving and listening to voice notes while in the conference call was the same as mentioned above (i.e. not effect to the stability of the program or the quality of the sound received over the conference call).

When two clients proceeded to send multiple voice notes to each other while simultaneously listening to voice notes and send messages to one another there was no effect to the stability of the program and messages and voice notes was able to be sent and received concurrently without influencing the quality of the sound playing from a voice note.

### 5.5.4 Conclusion

There was no effect of sending, receiving or playing voice notes while concurrently sending and receiving messages and voice notes on the stability of the program or the quality of the sound sent over a voice note or a voice note being played.

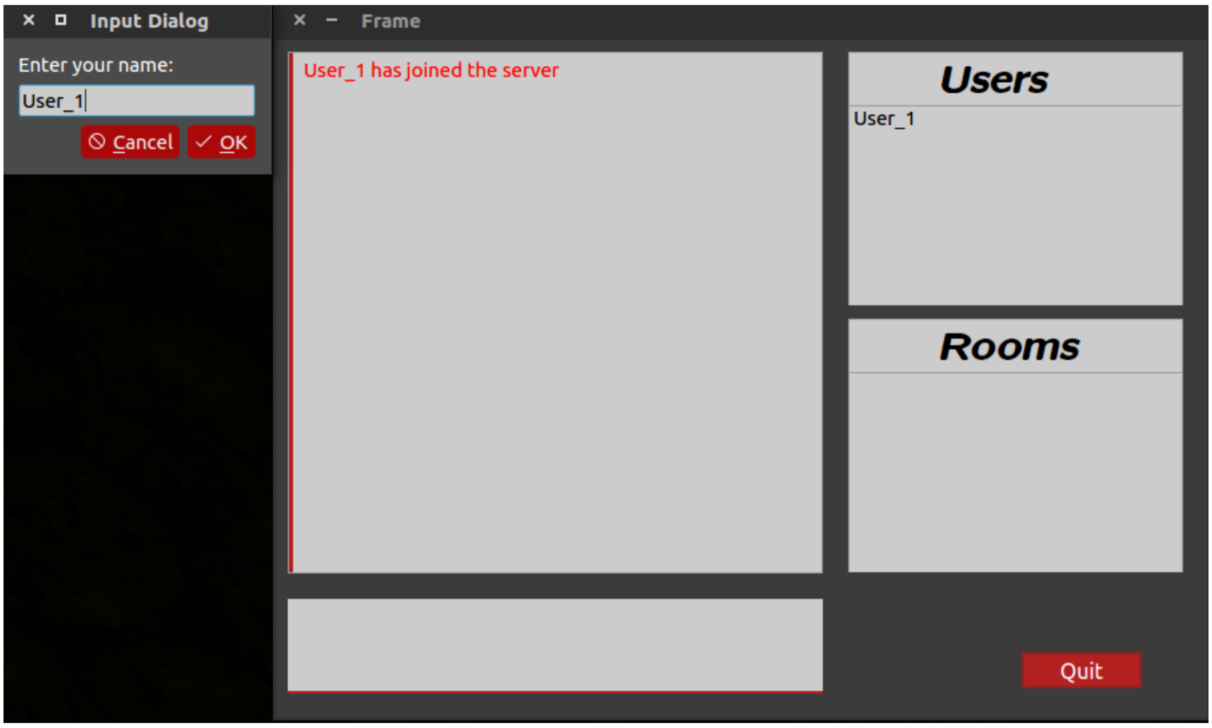
## 5.6 Connecting of client with the same username

### 5.6.1 Hypothesis

If a client tries to connect to a server with a username that is already used by another client connected to the server, the client will be notified that the username is already in use.

### 5.6.2 Method

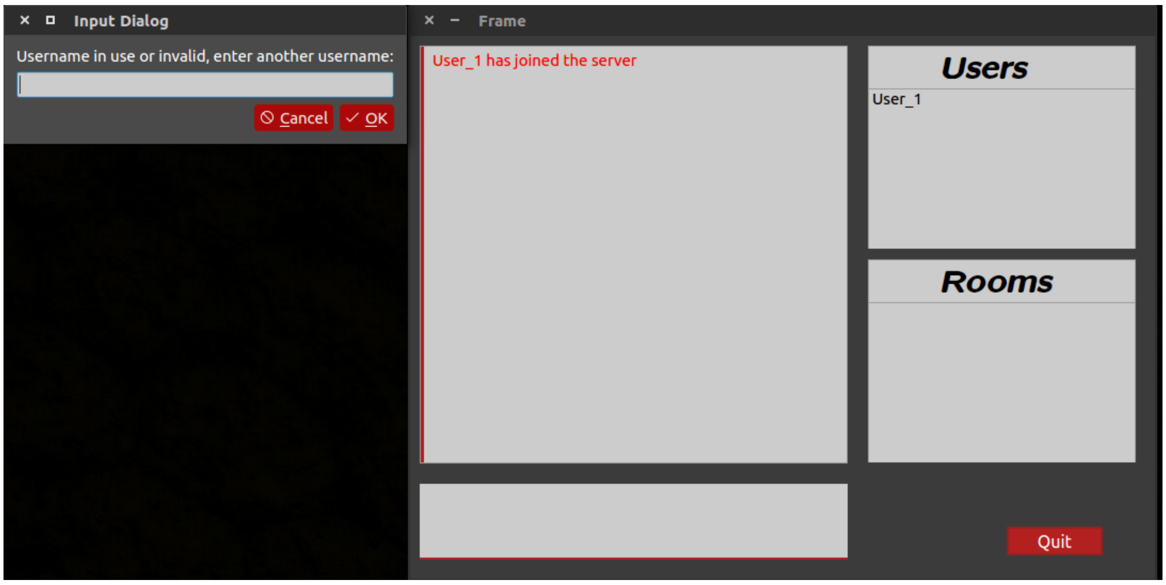
A server instance is created and a client is connected with the username User_1. Another client is then connected to the same server and the new client tries to use the same username User_1, the results from the server are recorded and a valid username is then entered by the new client.



User_1 has joined the server

User_1

**Figure 5.9:** The second client is connected and prompted to enter a username



User_1 has joined the server

User_1

**Figure 5.10:** The client is notified that the username is already in use

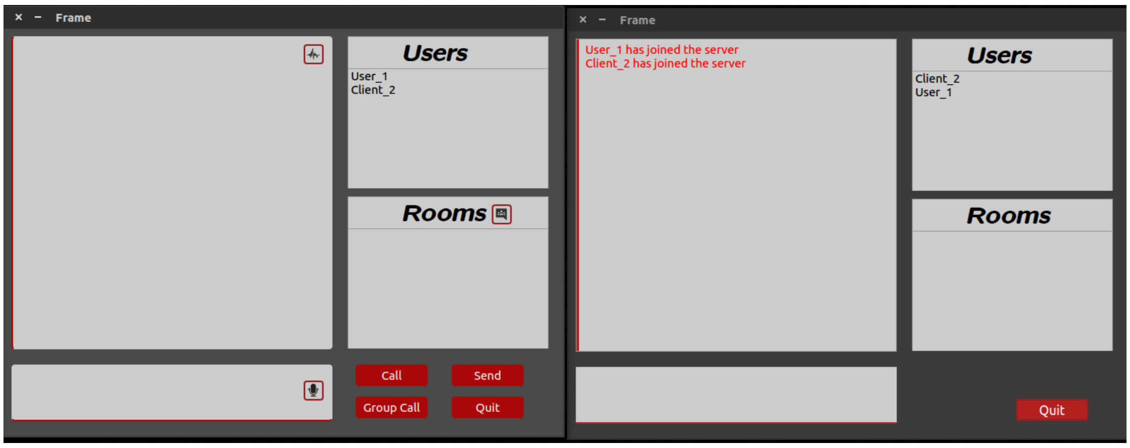

User_1 has joined the server
Client_2 has joined the server

**Figure 5.11:** The client enters a valid username and successfully connects to the server

### 5.6.3 Conclusion

As seen above clients can not connect to the same server with the same username, clients are notified that the username is already in use and prompted to enter another username. The hypothesis can be accepted.

## 5.7 Client functionality

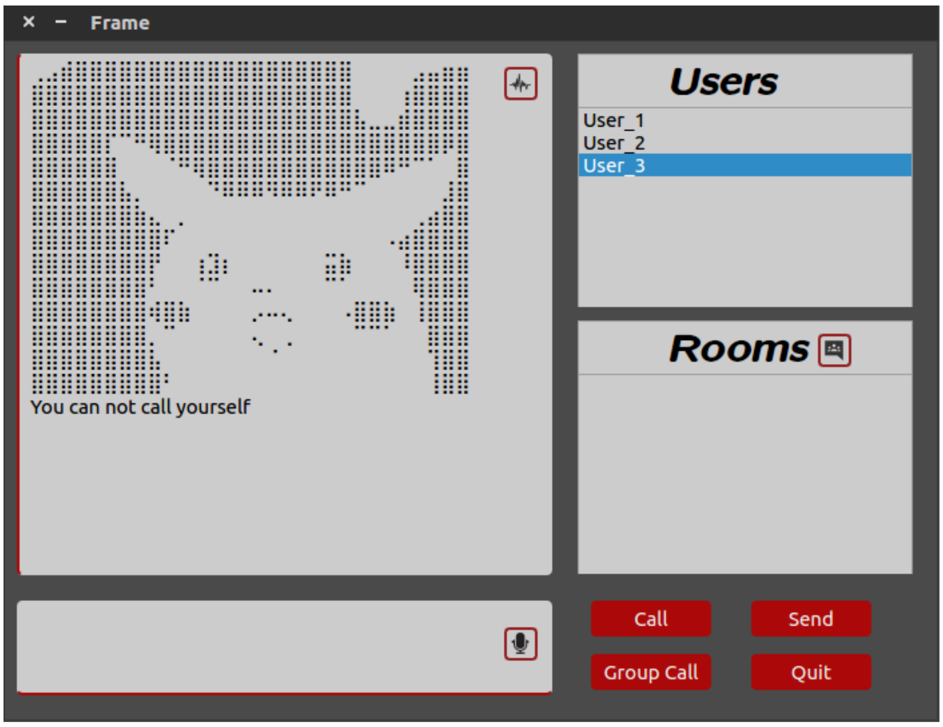
### 5.7.1 Hypothesis

When a client tries to execute an illegal operation the program will handle it accordingly and notify the client of the error.

### 5.7.2 Method

A few illegal operations were executed by a client and the results were observed.

### 5.7.3 Results



You can not call yourself

**Figure 5.12:** The client is notified that it is not possible to call yourself

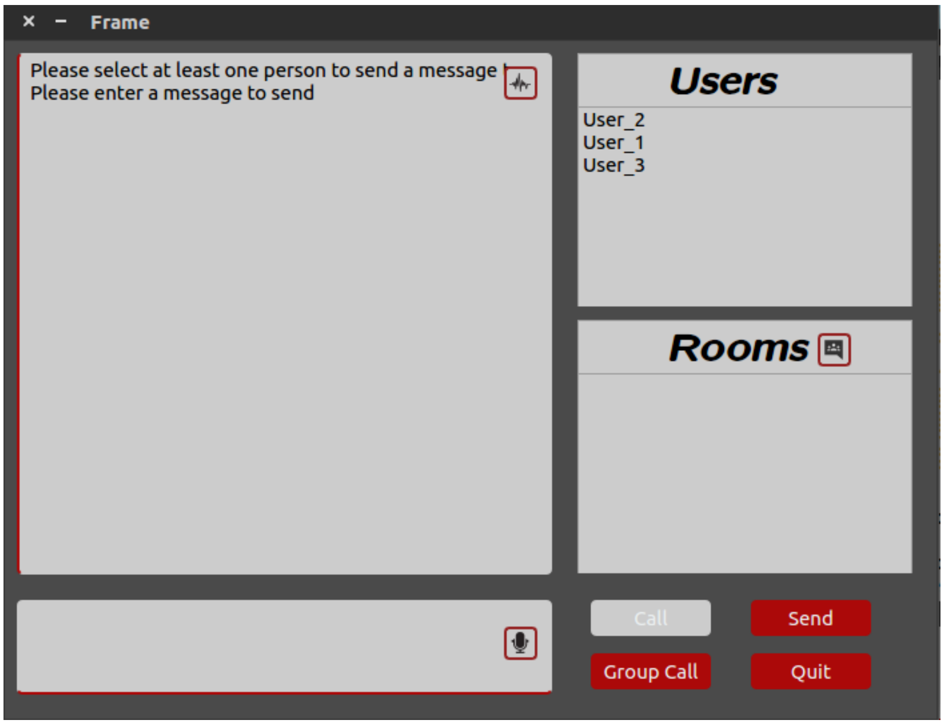

Please select at least one person to send a message t
Please enter a message to send

**Figure 5.13:** The client is notified to select a user to send the text message to, and to enter text

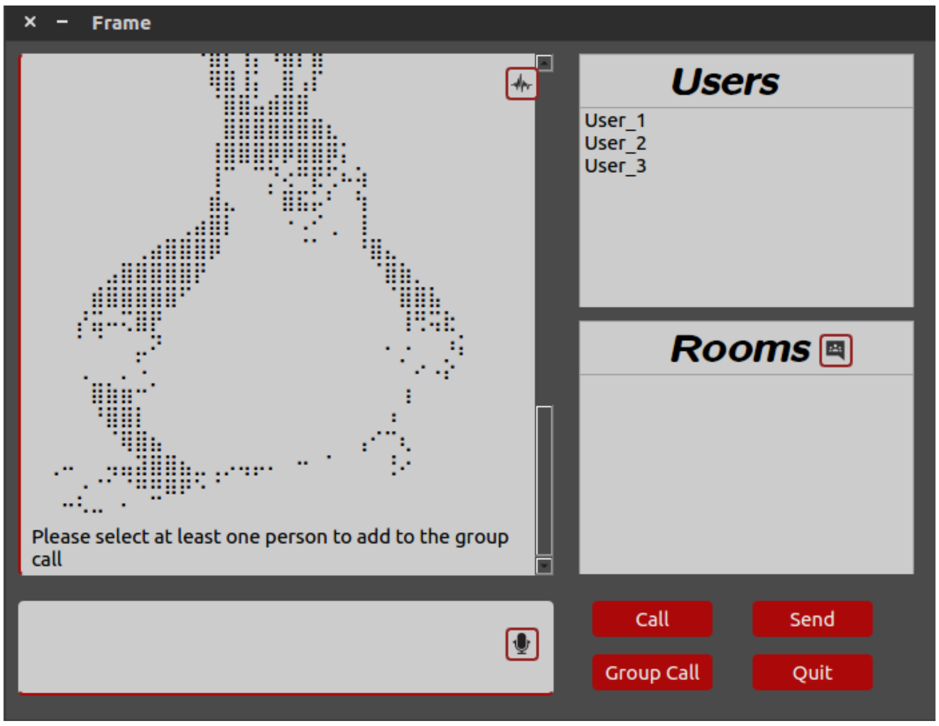
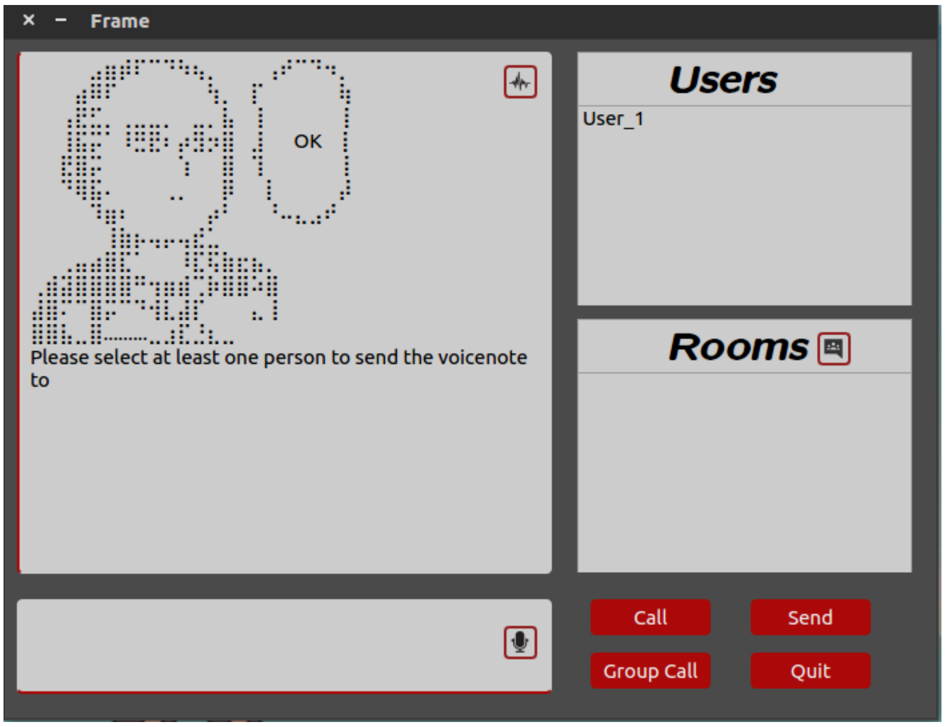

Please select at least one person to add to the group call

**Figure 5.14:** The client is notified that at least one person must be selected to start a group call



Please select at least one person to send the voicenote to

**Figure 5.15:** The client is notified that a user should be selected to whom the voicenote should be sent

### 5.7.4 Conclusion

As seen above all of the illegal operations are handled and the client is notified of the errors. The hypothesis can be accepted.

# 6 Conclusion

From the experiments performed the following was clear:

- The voice quality is heavily affected by a change in the chunk sizes, bigger chunk sizes resulting in better sound and smaller chunk sizes resulting in distorted sound.
- Sound quality and program stability is not affected by making or receiving consecutive calls.
- Conference calls do not affect the stability of the program and consecutive conference calls do not affect the quality of sound received over a conference call.
- Same as conference calls, conference channels also do not affect the stability of the program or the quality of sound received over channels after creating multiple channels or joining and leaving multiple times.
- Client functionalities such as trying to use duplicate usernames, calling yourself, not selecting anyone to call, etc. are managed by the program and do not affect the stability of the program.