

OptiChat

Aniket Shirke	150100012
Huzefa Chasmai	15D170013
Samarjeet Sahoo	150100017
Uddeshya Upadhyay	15D110007

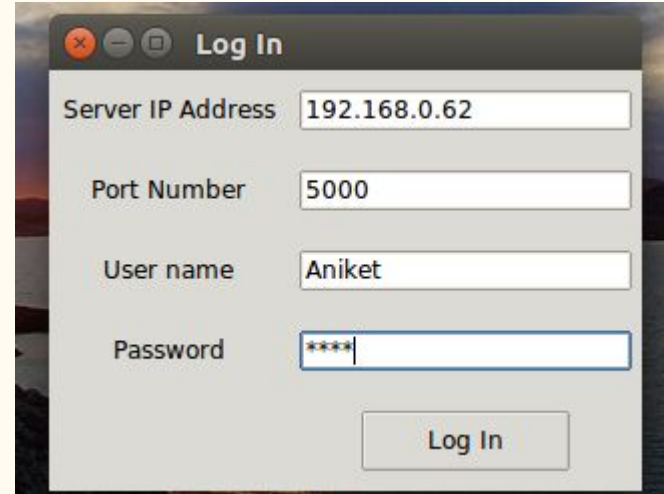
Description and Features

Our Chat application consists of a server running on Linux which can support multiple clients. A client can register for our application and use it to chat with other active clients selectively.

- Register to our application to get connected with your friends!
- We provide you with a really **cool GUI**
- Feeling bored to send the same message to multiple persons? We offer **Multicast messaging!**
- Why send only text messages, when we can provide you with the facility of sending **Multimedia files!**

Project Team Chat Application

OptiChat was built keeping in mind the uses and functionalities needed when working on a team/group project and one of our main inspirations was **Slack** which was used to built this very Project.



Architecture

- The connection which is being set up between the server and each client is a **TCP** connection
- *Tkinter* library is used as a platform incorporating the Protocol(next slide) to make the application more user friendly
- **Server**
 - The Server is hosted on a Linux machine
 - **Threading** has been implemented for managing individual clients.
- **Client**
 - On the client side, Tkinter , a GUI library in python is used for providing a cool user interface.
 - The complete code is object oriented and different views have been modularized which can be obtained by pressing appropriate buttons.

Protocol

- The messages which are getting sent to and fro server-client are encrypted using python's inbuilt **string encoding**
- The underlying format for each message which is being sent to the server from the client has the following format:

$$@<usr\ 1>\ @<usr\ 2>\ \dots\ @<usr\ n>\ :<message>$$

- For Broadcasting messages, only the message is sent directly to the server **without any '@'**
- For multimedia messages, the format is:

$$^@<usr\ 1>\ @<usr\ 2>\ \dots\ @<usr\ n>\ :<file\ name>:<base64\ encoded\ file\ content>$$

- Every time a new Client logs in, the server asks the client about its details, in particular send the string “**What is your name?**”, to store the mappings of the socket and the user ID.
- For broadcasting the list of active users, the server sends the list of active users appended after a ‘!’.
- For a Log in request, a ‘0’ is sent to the server to trigger the ‘*log in*’ procedure.
- For sending multimedia files, **base64 encoding** has been used to ensure the transfer of non ASCII characters from one client to another.

Server Implementation

- Implementing the server on python was **less verbose** as compared to C. The GUI provided by Tkinter helps in masking the underlying message format which is getting sent and parsed.
- For each client, an **individual thread** is set up!
- On initial log in, the client initially sends a '0' to the server else the server goes into the registration mode to register his user id into the database.
- The server asks the client for his *name* along with the *password* and authenticates him using the database.
- On success, the client is sent a *list of active users* and all the active users receive an *updated list of active users*.

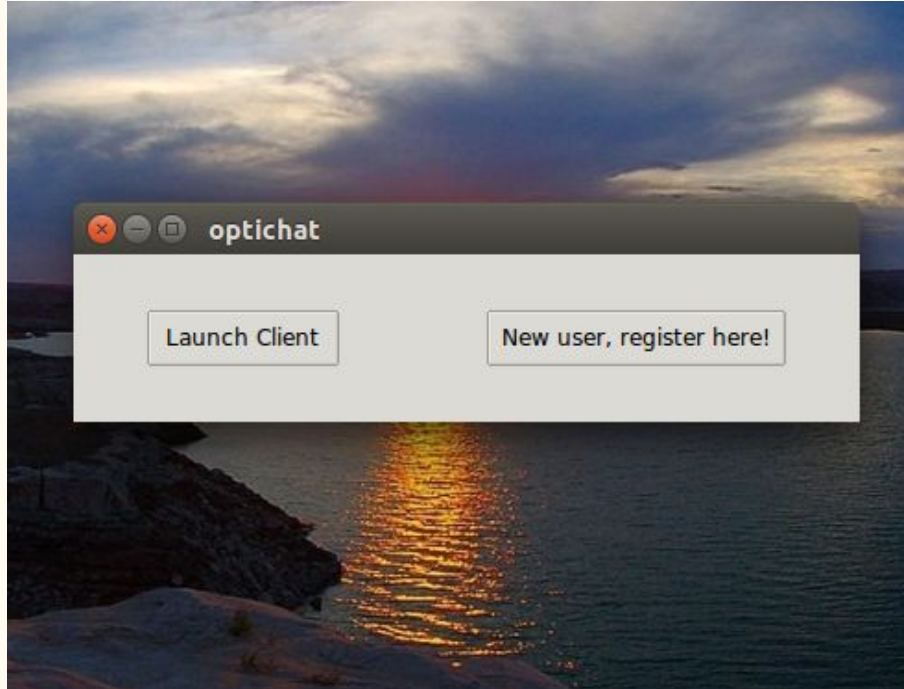
- The list of active users is broadcasted on **triggered updates**: when a new client *joins in* and when a client goes *offline*
- The server also supports **broadcasting** and **multicasting** of messages.
- “*Netifaces*”, a python library, is used for identifying the IP address of the server (and not hardcoding it). The only thing required is the port number for the initial hosting of the server. Note that it is assumed that the server machine is connected to the network via a **Wifi network interface**.
- The receive buffer space is set to **10 MB** (for supporting transfer of multimedia files)
- The server is interpreted by **python3**

Client Implementation

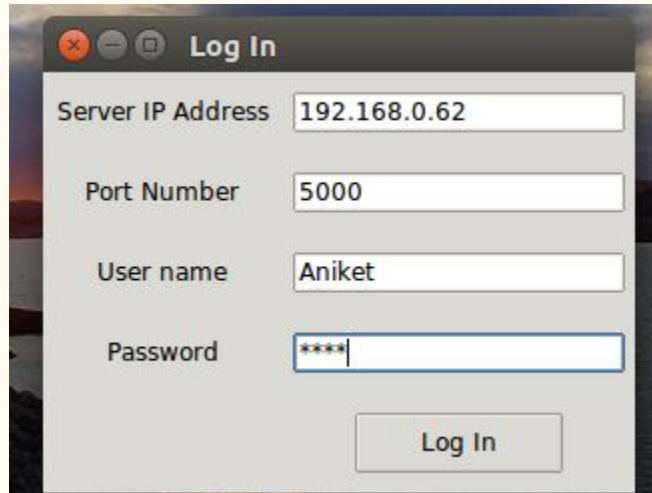
- The Client side is based on *Tkinter*. There is a single class '*Application*' and the flow of the code is developed by defining functions in this class.
- The initial view provides the client with an option to register or log in. If the client opts to register, the *registration menu* is displayed and on registration, the log in screen appears.
- The Client logs in by providing the **Server IP address**, the **Port number**, **User ID** and **Password**. If the Log in fails, an *authentication error* is projected and an option to *try again* is provided, else the main chat box is opened.
- On the left side of the text box, a **list of active users** is maintained along with adjacent **checkboxes**. The **chat history** covers the majority of the screen. On the bottom part, there is the text box and options to send message as well as multimedia

- To send a message, the client *ticks in the checkboxes* to whom the message is intended to and hits the send button.
- For sending multimedia, a **file dialog** pops up and the client can choose any file to transfer. (File contents are encoded using **base64 encoding**)
- The chat history is maintained in the standard form, the left hand side indicate the *received messages* while the right hand side are the ones which were *sent*.
- The client is interpreted by **python**

Screenshots : OptiChat



Log in Step

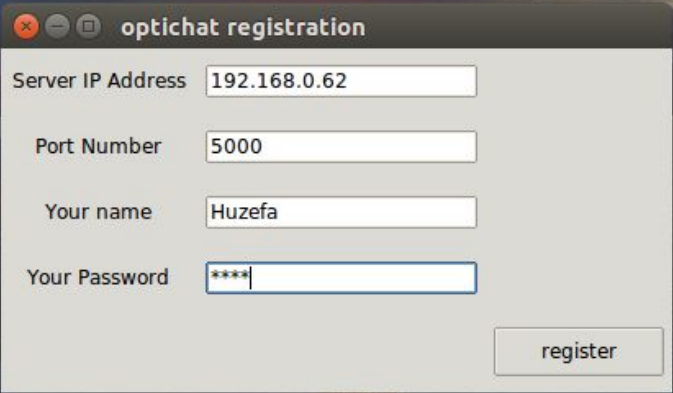


A screenshot of a "Log In" dialog box. The dialog has a title bar with standard window controls (close, minimize, maximize) and the text "Log In". Inside the dialog, there are four input fields arranged vertically, each with a label to its left:

- Server IP Address: 192.168.0.62
- Port Number: 5000
- User name: Aniket
- Password: ****

At the bottom right of the dialog is a button labeled "Log In".

Registration

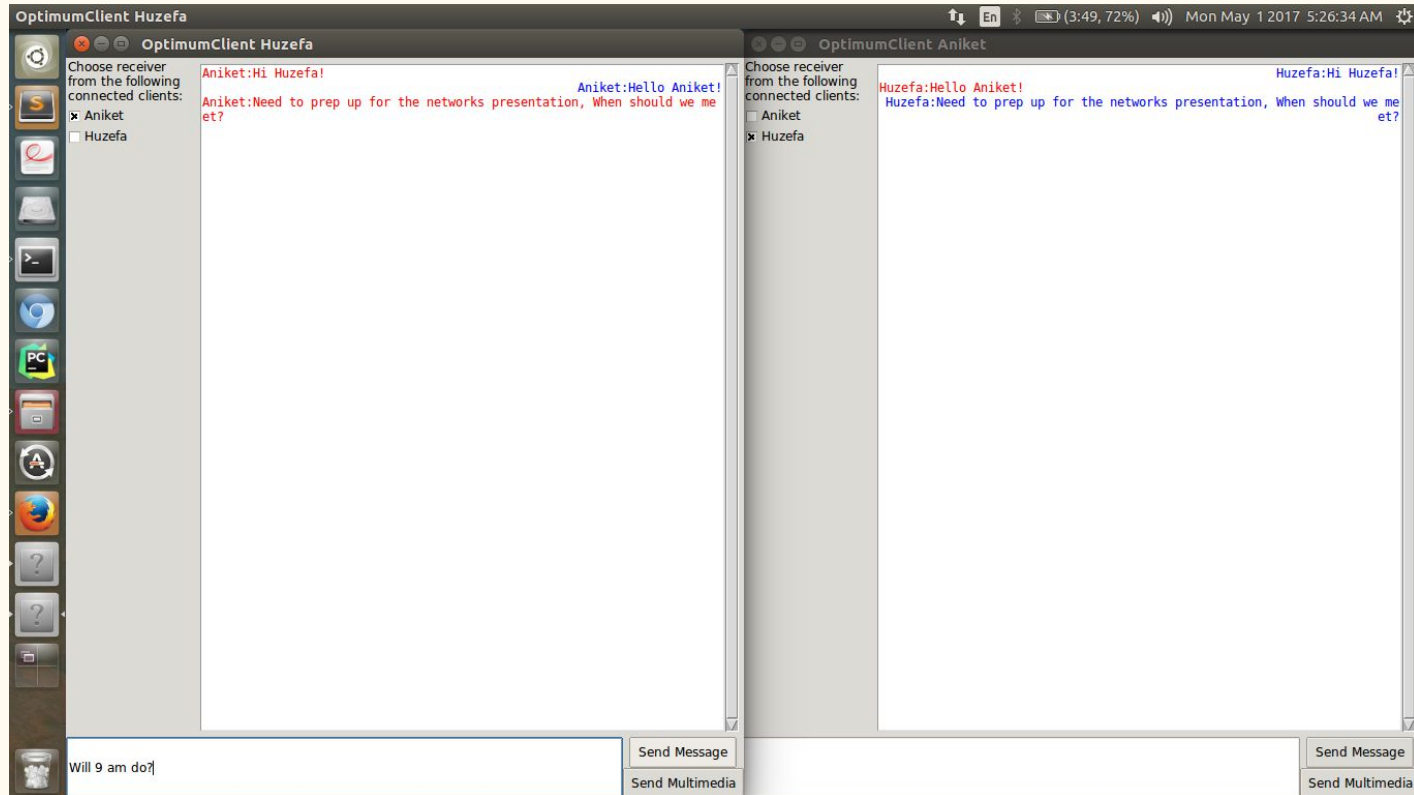


A screenshot of a registration dialog box titled "optichat registration" overlaid on a scenic background of a lake at sunset. The dialog box contains four input fields and a "register" button.

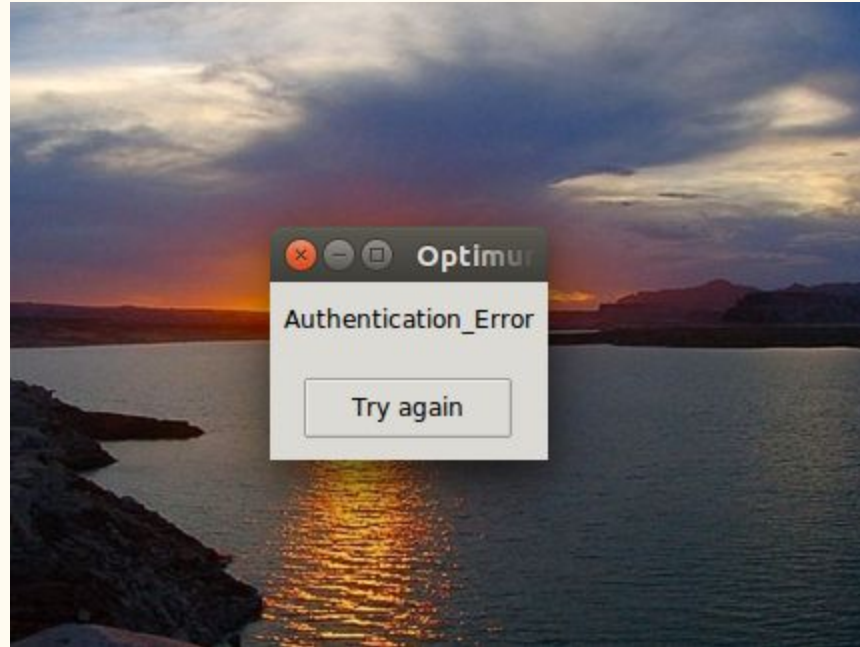
Field Label	Value
Server IP Address	192.168.0.62
Port Number	5000
Your name	Huzefa
Your Password	****

register

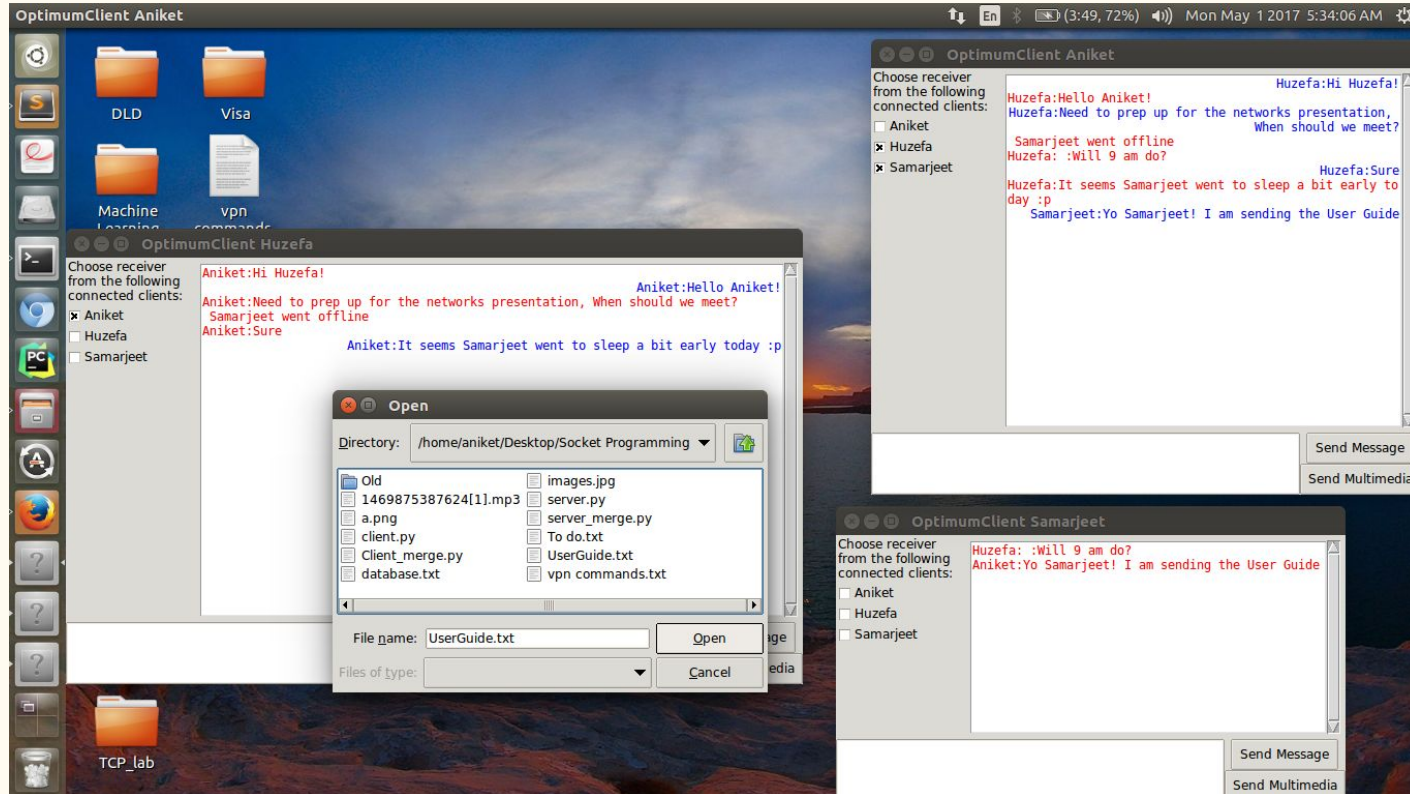
Sending messages



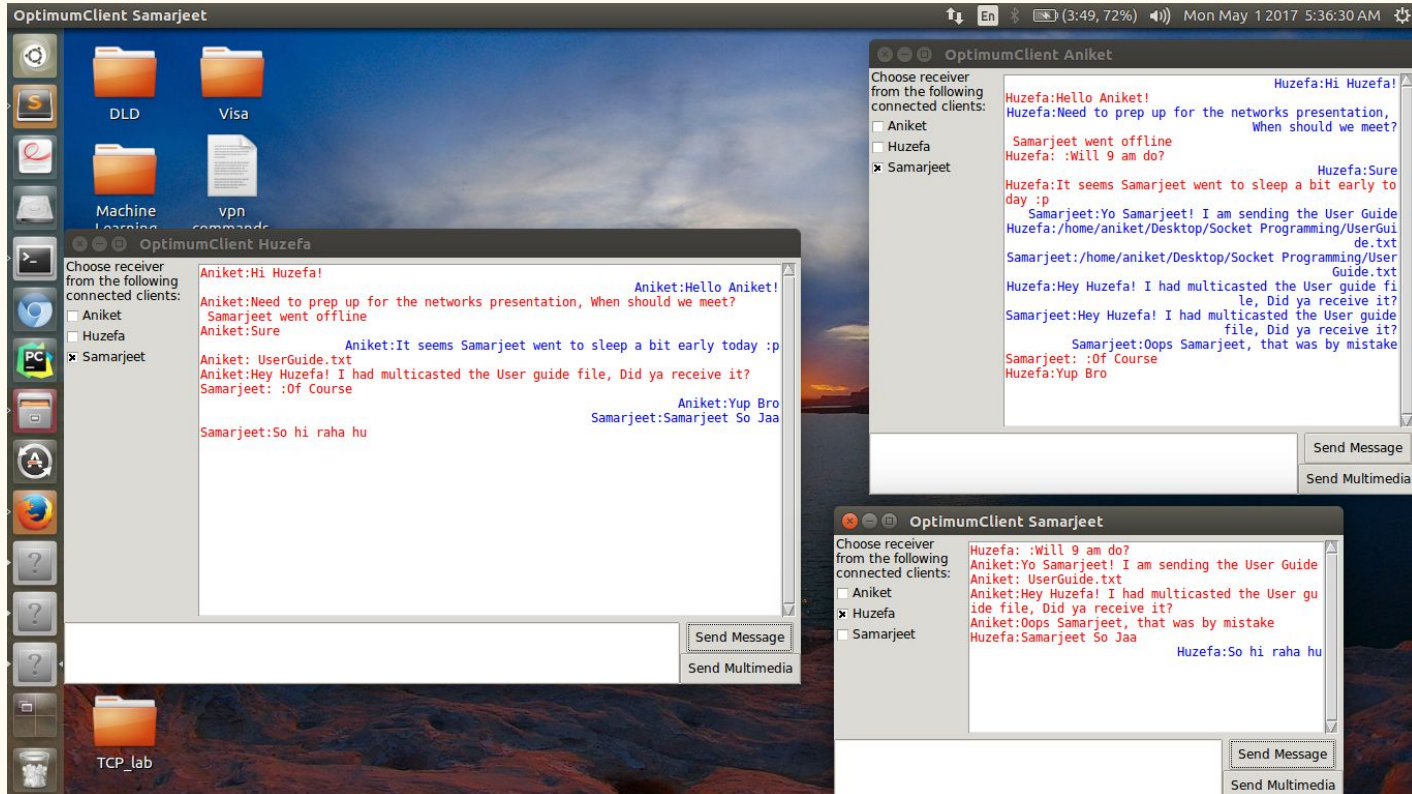
Authentication error



Sending Multimedia Files



Overall chat environment



Scope for Improvement

- The GUI could have been made more visually appealing, more colorful probably.
- The max file size for actual transfer is $\sim 100\text{KB}$. This could be improved.
- We have tried our best to smoothly handle all errors and bugs, but again, we are humans and there may just be an odd undiscovered bug popping up.

Work Division

The main part of the code has more or less equal contribution from each team member still the general initiatives for the various parts are:

Basic Structure, Protocol Designing: Samarjeet and Huzefa

Adding threading structure: Uddeshya

GUI: Huzefa, Samarjeet, Uddeshya, Aniket

Multimedia functionality : Aniket

Report, Presentation, Guide: Aniket, Huzefa, Samarjeet

References:

- <http://www.binarytides.com/code-chat-application-server-client-sockets-python/> (Basic structure of socket programming and generic chat application)
- <https://github.com/5hubh4m/ChatUp/blob/master/chatup.py> (GUI)
- <https://neerajkhandelwal.wordpress.com/2012/02/16/socket-programming-handling-multiclients/> (Threading)
- <http://google.com/> (Well, for almost every doubt)
- <http://stackoverflow.com/> (Doubts)

Thank You

**We still have a long way to go to beat WhatsApp!!
Appreciate any suggestions**