

CS 3041
NETWORKING LAB

**SIMPLE CLIENT-SERVER CHAT
SYSTEM
(PYTHON)**

Group Assignment

Bhupesh Raj (CS11B006)

Ritvik Jaiswal (CS11B031)

INTRODUCTION

The project was on implementing a simple GUI based client-server chat system in Python that communicates via TCP. The reliability of TCP over UDP was required for the chat system to function.

IMPLEMENTATION

The project was implemented in Python owing to its simple syntax and extensive libraries. Tkinter library of Python was used for implementing the GUI.

TCP was used exclusively to maintain the integrity of the chat system, as packet loss (loss of messages) is taken care of by TCP.

SERVER

The server binds the TCP socket to port 8080 of the system. It then listens for connections on that port. Once a connection has been established, the server replies with a simple message acknowledging the connection. The client then sends its nickname to the server which stores it in a 'Dictionary' against the client's IP address. Only now does the actual chat system start.

The server maintains two threads for each of its clients – one for sending messages to the client and one for receiving and storing the messages from the client. The received messages are stored in a list which is shared between the threads of all the clients. Locking mechanism was used to ensure thread synchronization.

Messages between the client and server include the nickname of the sending client along with the actual message typed by the client. This provides for ease of displaying the messages on the client.

The server maintains an additional thread for each client. The sole purpose of this thread is to periodically send to all the clients, the list of nicknames of all connected clients. Since the list is updated in the server every time a client changes its nickname, it is periodically sent to everyone so that each client has an updated list of other online clients.

Connection with a client is closed on receiving a particular message '1' from the client. This message is not broadcast to other clients, it is merely to be informed that the client wishes to leave the chat, and to terminate the connection with that client.

CLIENT

All of the GUI is implemented on the client side. Python's Tkinter library is used for this purpose.

Buttons are provided to the client for changing the server's IP address and port, changing the nickname and to connect to the server.

Messages are displayed on a ScrolledText Field.

List of users is displayed on a List Box.

There is a canvas provided to the client to test out their drawing skills. There is also a reset button to reset the canvas.

Another List Box displays information about the server's IP address and port, and the client's nickname.

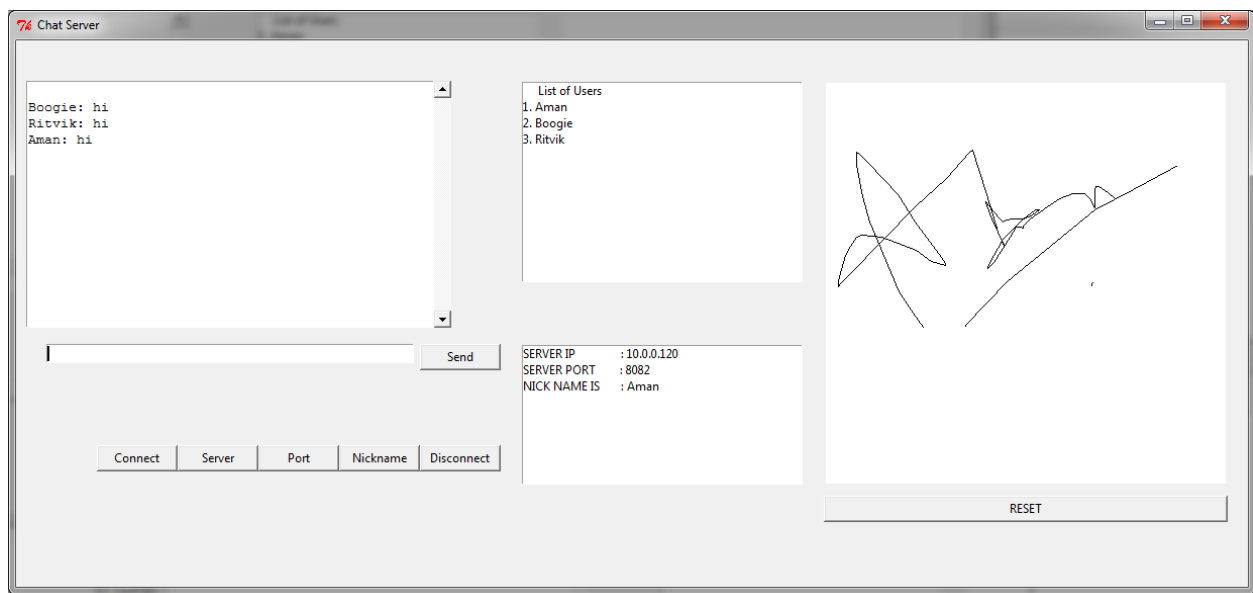
Before the client can start chatting with other clients, it has to send particular messages to the server informing the server about its nickname.

The list box containing the list of connected clients is refreshed periodically, with data obtained from the server.

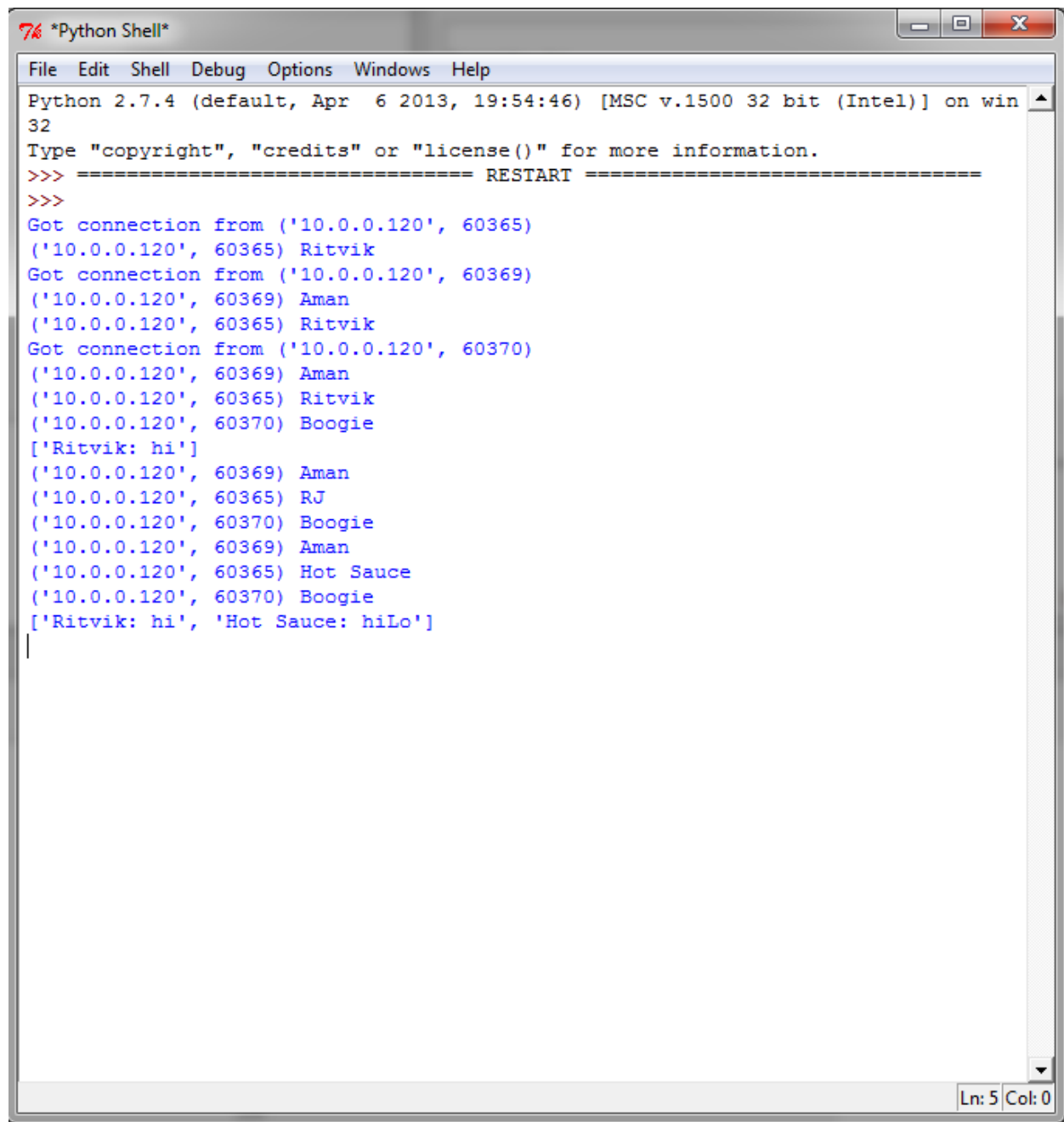
If the client wishes to disconnect from the server, it sends a message '1', after which the socket between the client and the server is closed.

SCREENSHOTS

CLIENT



SERVER

A screenshot of a Windows-style application window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the output of a Python script. The script starts with a header for Python 2.7.4, followed by a prompt to type "copyright", "credits", or "license()". Then, it prints a separator line "===== RESTART =====". The script then enters a loop where it receives connections from '10.0.0.120' at various ports (60365, 60369, 60370). For each connection, it prints the IP and port, followed by the client's name (Ritvik, Aman, RJ, Boogie, Hot Sauce). It then prints a list of messages: ['Ritvik: hi'], ['Aman', 'RJ', 'Boogie', 'Aman', 'Hot Sauce', 'Boogie'], and finally ['Ritvik: hi', 'Hot Sauce: hiLo']. The status bar at the bottom right shows "Ln: 5 Col: 0".

```
Python 2.7.4 (default, Apr  6 2013, 19:54:46) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Got connection from ('10.0.0.120', 60365)
('10.0.0.120', 60365) Ritvik
Got connection from ('10.0.0.120', 60369)
('10.0.0.120', 60369) Aman
('10.0.0.120', 60365) Ritvik
Got connection from ('10.0.0.120', 60370)
('10.0.0.120', 60369) Aman
('10.0.0.120', 60365) Ritvik
('10.0.0.120', 60370) Boogie
['Ritvik: hi']
('10.0.0.120', 60369) Aman
('10.0.0.120', 60365) RJ
('10.0.0.120', 60370) Boogie
('10.0.0.120', 60369) Aman
('10.0.0.120', 60365) Hot Sauce
('10.0.0.120', 60370) Boogie
['Ritvik: hi', 'Hot Sauce: hiLo']
```

CHALLENGES FACED

The initial plan was to share the canvas between the clients, enabling another level of message passing. Sharing the canvas would mean that any changes on the canvas of one client would be reflected on canvases of all other clients on the network.

It is impossible to share real time changes on the canvas as that would mean continuous passing of the canvas from each client to the server which would then have to forward the canvas to each of the other clients.

Instead, it was planned to update the canvas every few seconds on each of the clients. The client would send the image on the canvas to the server every few seconds and the server would immediately forward it to all other clients. This did not work as the images were too large. Moreover, these images were interpreted by the server as normal text messages, which were subsequently forwarded to every client and displayed on their text fields as text.

The idea to share the canvas was dropped, and instead the canvas was made local to each client.

FUTURE IMPROVEMENTS

The current application does not support private chats. Messages sent by one client are received by every other client. Private chats would allow clients to send messages to only a select few of the other clients.

Canvas is currently not shared between clients. It is local to each server. If a shared canvas is supported, it will make the application more interactive. The concept could be extended to sending emoticons to other users.

REFERENCES

The following sites were referred to implement the basic client-server functionality and the GUI –

Python tutorial – Tutorials Point

www.tutorialspoint.com/python/

What's Tkinter? – Effbot

effbot.org/tkinterbook/tkinter-whats-tkinter.htm