CS310: Computer Networks
Assignment 1: Develop a network application using Socket programming

Group Members: Ranjan Naidu & Roska Takayawa
Student IDs: S11201181 & S11187423
Lab Time: Thursdays 12 - 2 pm
Campus: Laucala

Table of Contents

Introduction

Throughout centuries in many different countries communication has always played a very important role in connecting people and achieving their goals. With technology becoming more advanced as the year progresses, chat applications have become an essential part of our daily lives. In this report, we will be showing the source code prototype, usage, test report and a technical report illustrating how row TCP sockets are implemented in the source code provided. This report will also show what the interface of the application looks like, with an emphasis on any implementation issues and a conclusion of what we have achieved in this assignment and the difficulties that were faced while doing the assignment.

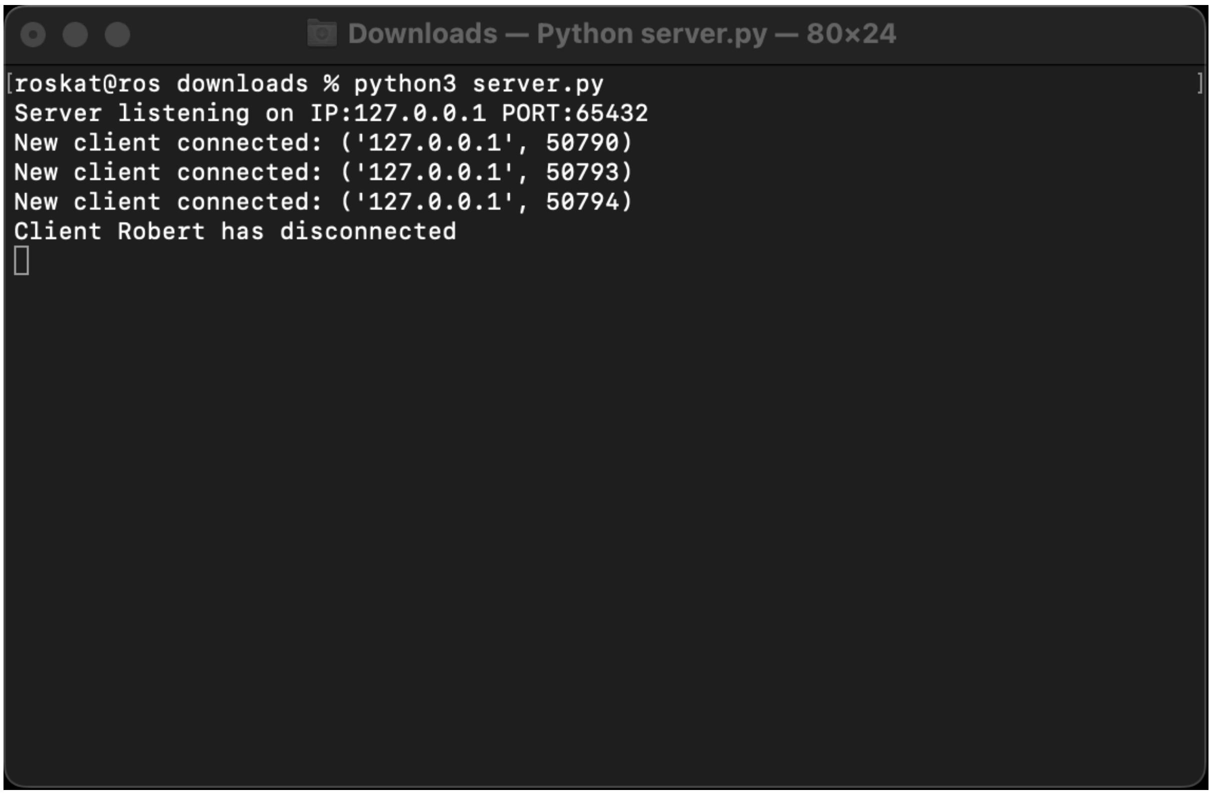
The prototype that is designed uses a hybrid architecture that adapts to both peer-to-peer and client-server architectures, displaying the important components of both network applications. The prototype also uses TCP socket connections and includes a client and a server, where users must authenticate before connecting to the server.

The main feature of this prototype is to allow users to send and receive text messages directly from any other user using the application at any time. Additionally, users will also be able to privately message other users within the group chat and send messages simultaneously within the chat.

Important Emphasis

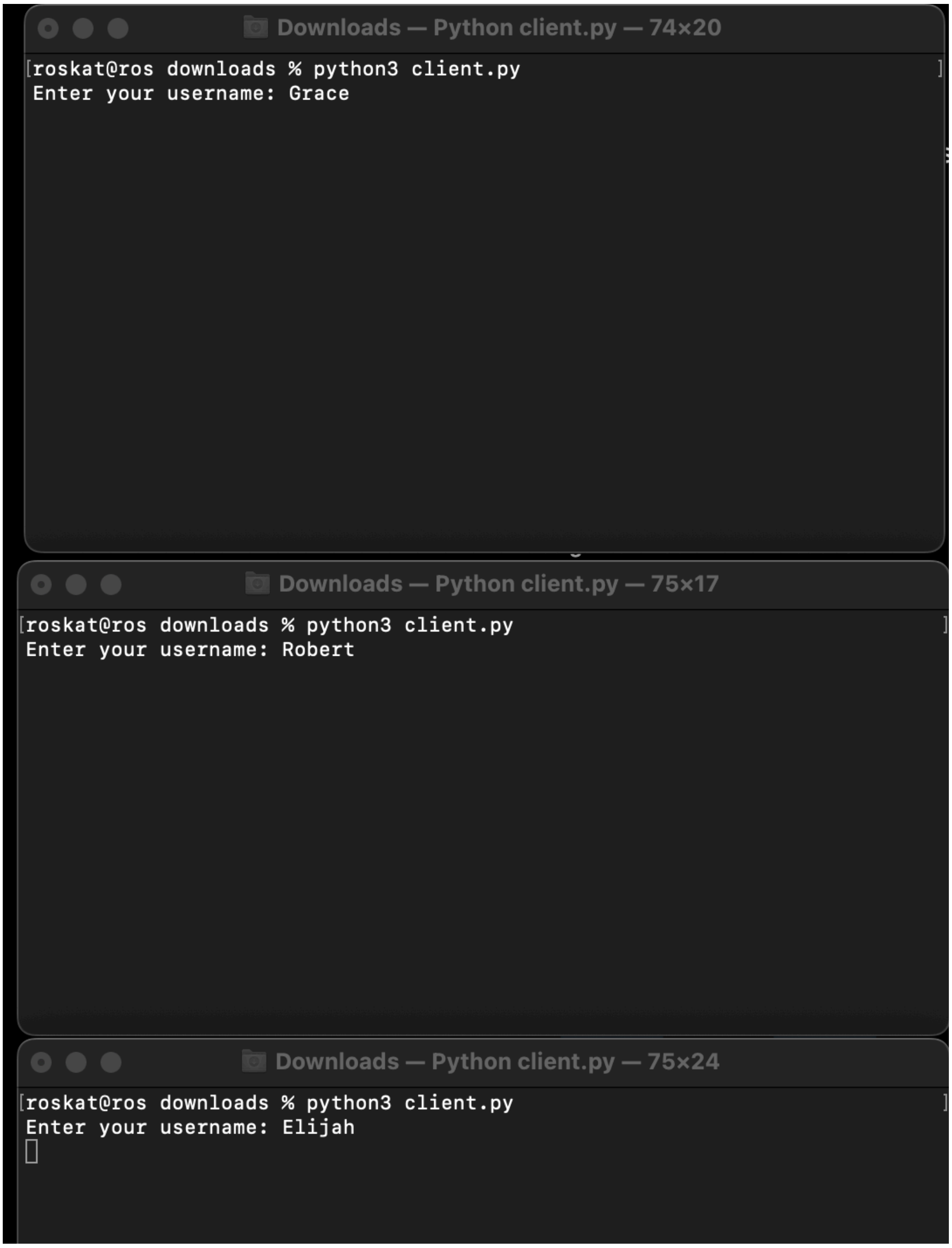
Screenshots of how to work the application

The screenshot below shows a terminal window running the Server.py file, and shows what IP and Port it has, and shows when a new client has joined or when a user exits



```
roskat@ros downloads % python3 server.py
Server listening on IP:127.0.0.1 PORT:65432
New client connected: ('127.0.0.1', 50790)
New client connected: ('127.0.0.1', 50793)
New client connected: ('127.0.0.1', 50794)
Client Robert has disconnected
```

Screenshot of making the clients for the chat room


Downloads — Python client.py — 74×20

```
roskat@ros downloads % python3 client.py
Enter your username: Grace
```

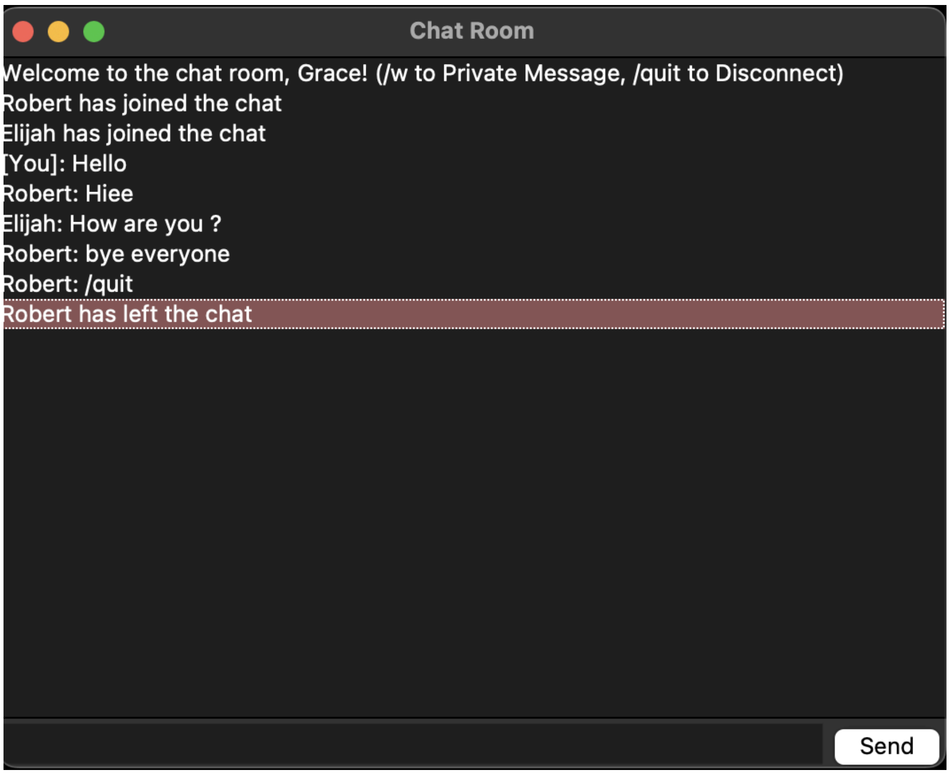
Downloads — Python client.py — 75×17

```
roskat@ros downloads % python3 client.py
Enter your username: Robert
```

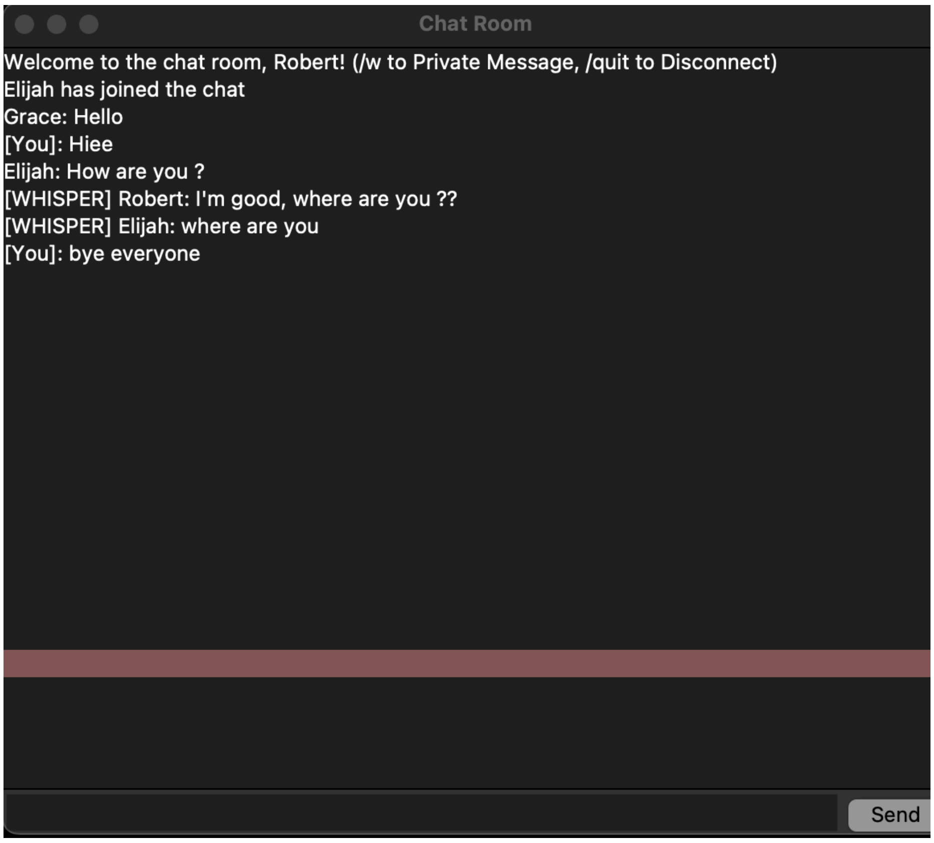
Downloads — Python client.py — 75×24

```
roskat@ros downloads % python3 client.py
Enter your username: Elijah
```

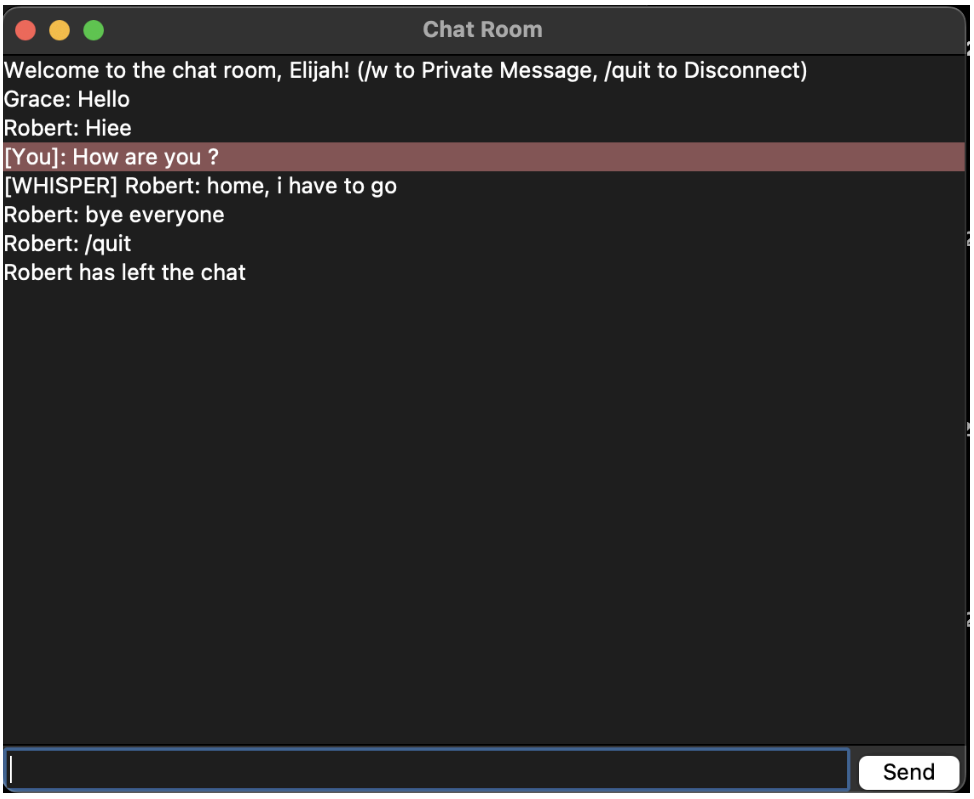
Our user interface and users interacting both through public chat and private messaging show what happens when a user quits and exits the chat room



```
Welcome to the chat room, Grace! (/w to Private Message, /quit to Disconnect)
Robert has joined the chat
Elijah has joined the chat
[You]: Hello
Robert: Hiee
Elijah: How are you ?
Robert: bye everyone
Robert: /quit
Robert has left the chat
```



```
Welcome to the chat room, Robert! (/w to Private Message, /quit to Disconnect)
Elijah has joined the chat
Grace: Hello
[You]: Hiee
Elijah: How are you ?
[WHISPER] Robert: I'm good, where are you ??
[WHISPER] Elijah: where are you
[You]: bye everyone
```



```
Welcome to the chat room, Elijah! (/w to Private Message, /quit to Disconnect)
Grace: Hello
Robert: Hiee
[You]: How are you ?
[WHISPER] Robert: home, i have to go
Robert: bye everyone
Robert: /quit
Robert has left the chat
```

Technical report on how row TCP sockets are implemented in the prototype

A TCP (Transmission Control Protocol) is a reliable, connection-oriented protocol that is used to provide a reliable and ordered data stream between networks/applications on different hosts. This will discuss more about how TCP sockets are implemented in the code given. Since it is a connection-oriented protocol, this means that a connection is established between two hosts before data transfer can occur.

With the chat room application, the server listens for incoming client connections and establishes a TCP socket connection with each client that connects.

Given below is the TCP socket implementation:

First, the server creates a socket using the socket() method provided by the Python socket module:

```
import socket
# create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Secondly, the server binds the socket to a specific IP address and port number using the bind() method:

```
# bind the socket to a public host and a well-known port
server_socket.bind((host, port))
```

Thirdly, the server listens for incoming client connections using the listen() met

```
# listen for incoming connections
server_socket.listen(5)
```

When a client connects, the server accepts the connection using the accept() method, which returns a new socket object that is used for communication with the client.

```
# establish a connection
client_socket, addr = server_socket.accept()
```

The server can then send and receive data from the client using the send() and recv() methods of the client_socket object.

```
# receive data from the client
data = client_socket.recv(1024)

# send data to the client
client_socket.send("Hello, client!")
```

The client also creates a TCP socket using the socket() method and connects to the server using the connect() method.

```
import socket

# create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# connect to the server
client_socket.connect((host, port))
```

The client can then send and receive data from the server using the send() and recv() methods of the client_socket object.

```
# send data to the server
client_socket.send("Hello, server!")

# receive data from the server
data = client_socket.recv(1024)
```

Implementation Issues

The implementation issues that can be faced while making a chat application:

- Compatibility Issues: The chat application could encounter compatibility issues with different devices and operating systems and may not be able to run on certain ones.

- Error Handling: The application must be able to handle errors such as server crashes and client disconnections. For this issue to be addressed, the implementation should include robust error-handling procedures and provide appropriate error messages to the user, which has yet to be implemented.

- Security vulnerabilities: The chat application requires users to authenticate before connecting to the servers. To address this issue, there must be an implementation of robust authentication procedures and the use of secure encryption methods.

- Scalability: The application must be able to handle a large number of users, and as the number of users increases, the performance of the application could/degrade. For this issue to be solved, there needs to be an implementation of a scalable architecture and to use of efficient data structures to manage user data.

Conclusion

To conclude this report, the development of this chat application was both a challenging and rewarding experience. Throughout the project, we have learned various programming concepts, such as socket programming, client-server, and peer-to-peer architecture.

The resources that were used to help guide us through this assignment were the use of Google and YouTube tutorials.

The process of getting this project done was the error handling for client disconnection, getting the private messaging to work, ensuring there was secure and reliable communication between users and time management as we had upcoming tests the entire week and had to prepare for them whilst also working on this project.

However, despite all the challenges mentioned in the previous paragraph, the final prototype was a success and demonstrated that it can fully function in both client-server and peer-to-peer network applications, allowing users to send and receive text messages both privately and publicly.

Contribution

Student ID and Name	Contribution
Ranjan Naidu (S11201181)	50.00%
Roska Takayawa (S11187423)	50.00%