CSC/ECE 573 - Spring20
Project Report

CHAT ROOM COMMUNICATION WITH ENCRYPTION

Submitted on:04/23/2020

Team:
Abhishek Gadireddy - agadire
Lakshmi Aishwarya Nellutla - lnellut
Lokesh Reddy Police - lpolice
Unith Mallavaram - ukmallav

Effort Breakdown

Component	Component weightage	agadire	lnellut	lpolice	ukmallav
High level design	0.1	40	5	5	50
Algorithm development	0.25	20	30	20	30
Coding	0.35	20	20	40	20
Debugging	0.2	25	25	25	25
Report writing	0.1	40	50	5	5
Per student aggregate contribution		25	25	25	25

Introduction

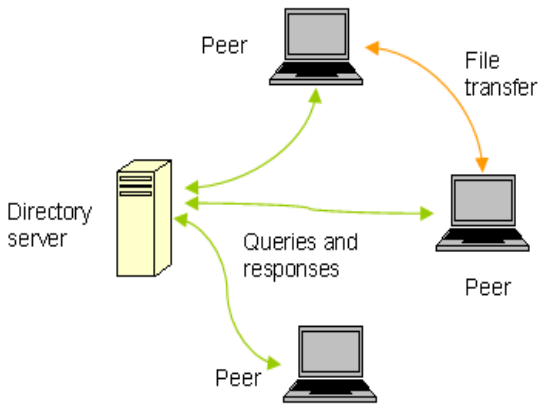
Objective:

- To create a chat room with the capabilities of exchanging encrypted messages as well as files among the peers in the room. The application should also let the users in the room to send encrypted data to selected peers.
- Implementation of this chat room is done with the support of file transfer and socket programming. To achieve low latency, we intend to use UDP as our transport layer protocol and reliability is implemented in the application using Go Back-N algorithm.

Information:

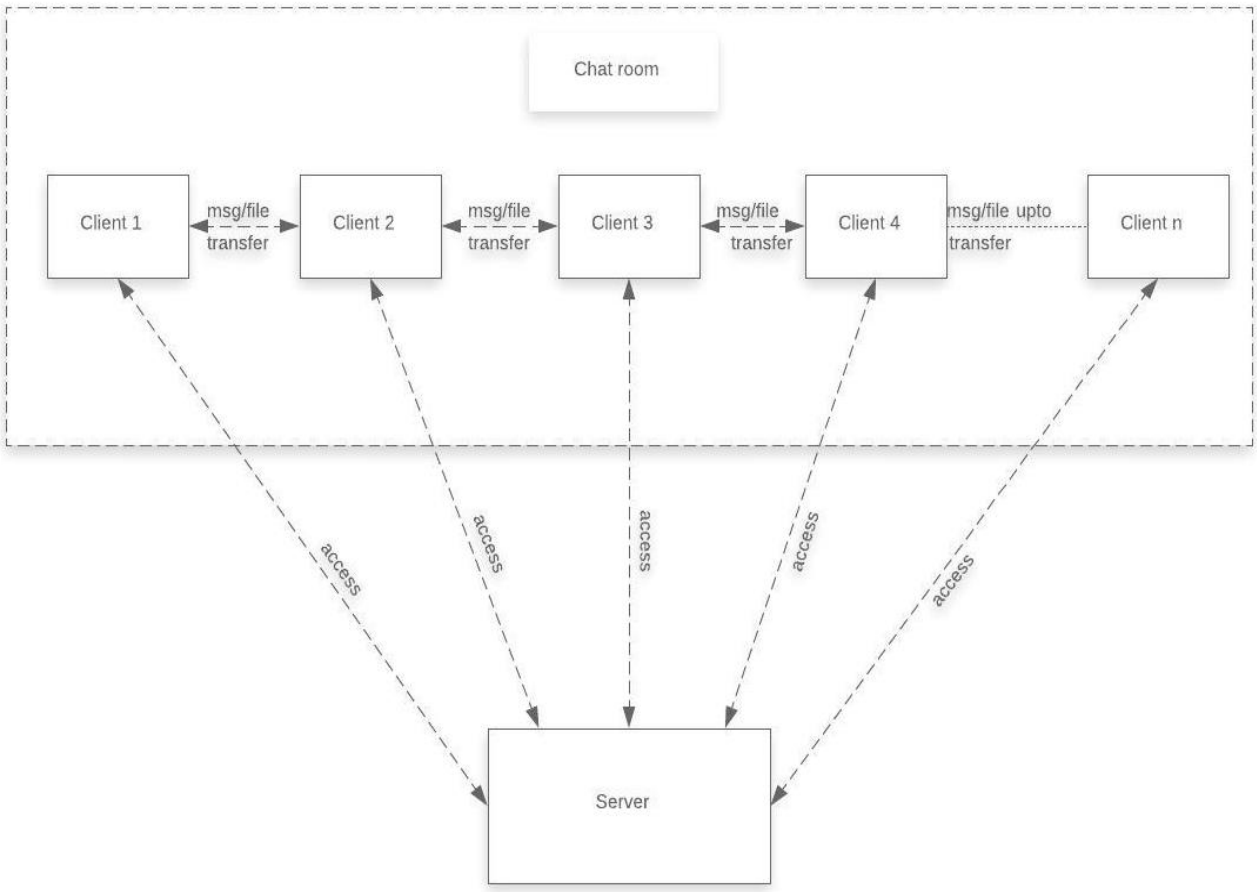
- A server is initiated by providing a password. The clients connect to a server and logs in to a chatroom by providing password. Server facilitates all message passing and informs the clients about peers already connected to the server, and existing clients get information about other clients. Multiple clients connect to the server which handles all clients in parallel.
- The following are the functionalities of chat room:
    - Broadcast messages - Sending a message to all the other clients in the chat room
    - Multicast messages - Sending a message to multiple clients
    - Unicast messages - Sending a message to a specific client in the chat room
    - Blockcast messages – Sending a message to all the clients excluding the client ids provided
    - Broadcast files - Sending a file to all the other clients in the chat room
    - Multicast files - Sending a file to multiple clients
    - Unicast files - Sending a file to a specific client in the chat room
    - Blockcast files – Sending a file to all the clients excluding the client ids provided
    - Kick a user out of the chat room by gaining at least two votes
    - Encryption of all communication including messages and files
    - Ensure reliable delivery using Go Back-N algorithm in the application
- Clients process the behavior of the chatroom by giving the appropriate commands for the above-mentioned functionalities based on the requirement.
- File transfer is a UDP based peer-to-peer transfer that does not involve the server for the sake of performance.
- UDP transfers are implemented with Go Back-N protocol for reliability where N is configurable.
- A peer can listen to user inputs and server messages at the same time over and above sending/receiving multiple files at the same time.
- A user input console for the server can kill a client if had a minimum of two votes.
- Server will be always running and listening on a pre-assigned set of TCP ports. Clients which request for file transfer will have files get transferred through UDP port. Throughput of the file is checked by varying N in Go Back N protocol to see effect of window sizes.

- Architecture model:

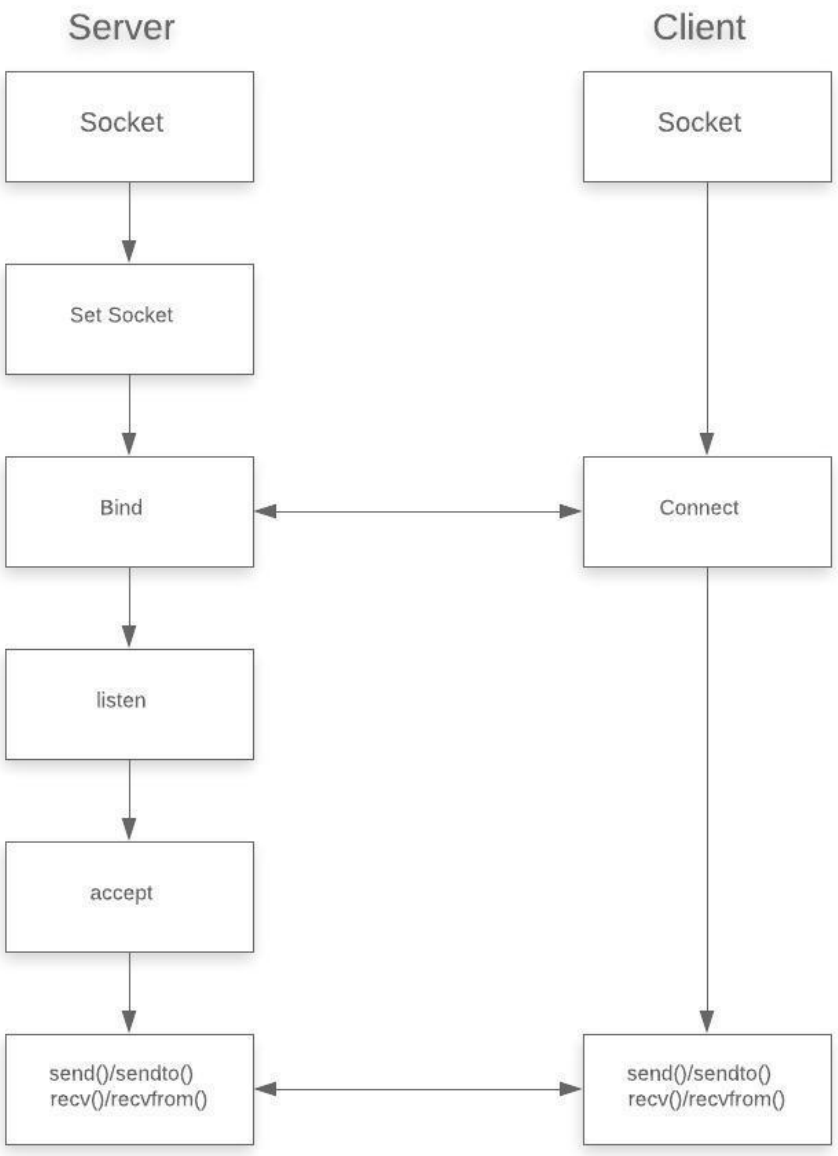


Design

Block Diagram of the system:

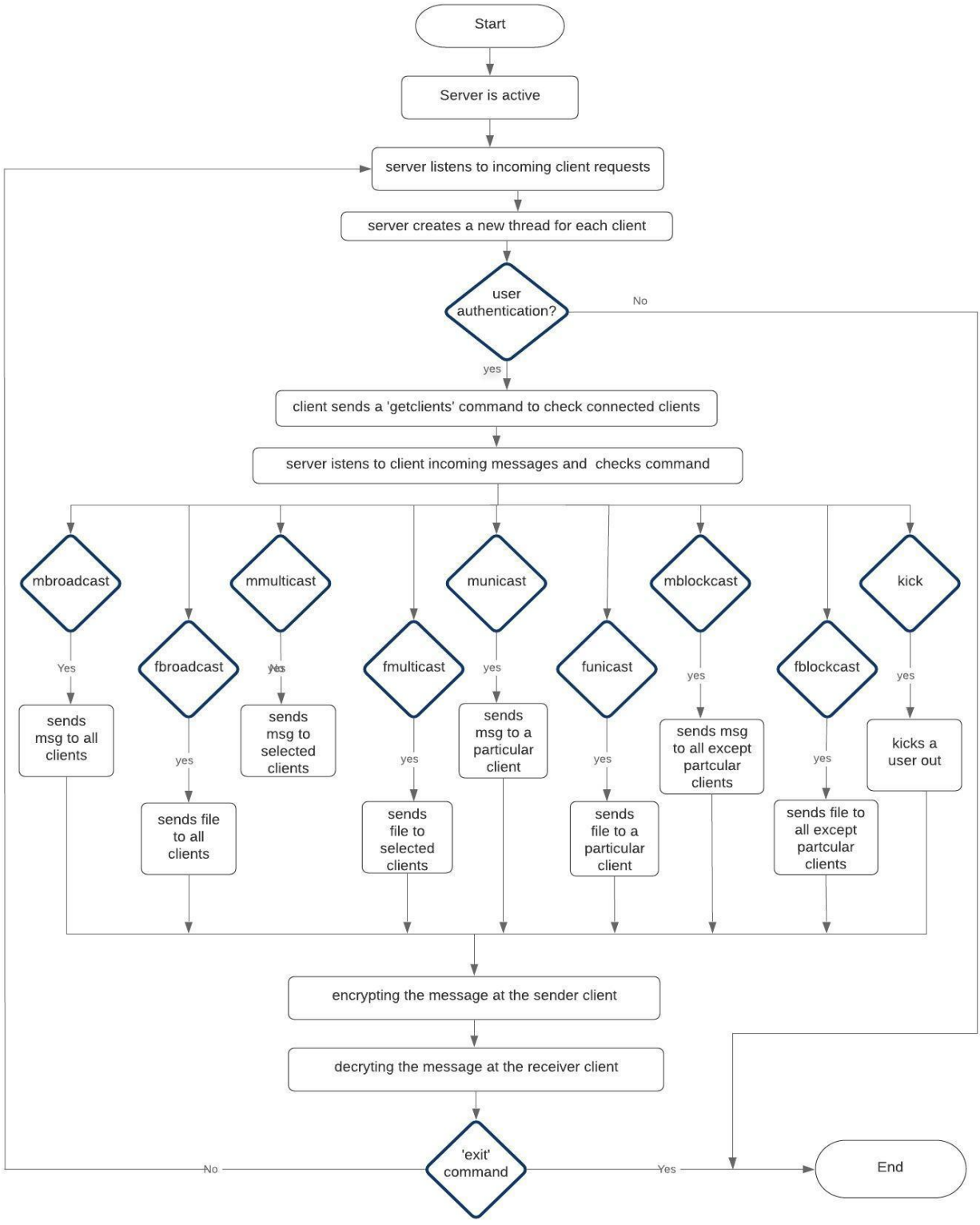


Server-Client Algorithm:



Note: TCP connection uses send() and recv() functions whereas UDP connection uses sendto() and recvfrom() functions

Program Algorithm:



Flow Chart depicting the detailed conversation between clients in a chatroom. TCP connection is used for sending messages and UDP connection is used for files.

Implementation

Programming language and tools used: Python 3

Python libraries are used to implement socket programming.

Configuration:

- All the devices are connected to the same local network.
- The IP address of the server is the IP address of the device on which it is running. It is obtained by giving ipconfig command.
- Firstly, run the server program by giving python3 server.py <ip_address> <port_number> <password>
- Run the clients using the command python3 client.py <ip_address> <port_number> <password>

Contents of the Source Code files:

The following is the brief description of files:

1. **Userdefinedfunctions.py:**
   This file has all the basic functions we use throughout our program for implementing message and file transfer functionalities. It contains the send and receive functions for encoding and decoding the data.

2. **Server.py:**
   This file runs the server application. It implements the TCP server on which clients connect in order to create a chat room and start using its functionalities. It has one class defined, the server class which represents all the server related information. It has a list of client objects who are instantiated by the server. Server creates multiple threads for the clients. Each client is run on a separate thread created by the server. For removing a client, we suspend the thread attribute for that client. It contains the functions to remove clients, list chat room names, binds to a port number, and listen on a TCP socket, data broadcasts after getting an input from the user.
   run() is the main thread of the server. It goes online and starts listening for an incoming TCP connection from a client. When a new client connects, it starts running with a new thread. So, each client connects to a new thread instance of the server and hence, we can have parallel clients. We implement this by the concept of multithreading to improve performance of the server. The port number on which the server binds is the TCP port number. Encryption of messages is also done before sending the data to the client.

3. **Client.py:**
   This file holds all the client related information. It is initialized with the IP address of the client and the socket at the client which is used to communicate with the other clients. It has functions to connect to a server, create a chatroom by implementing authentication functions and send messages to the server. One point of importance is that that the first client is the one who creates the chatroom, can specify using the password functions. It displays a central menu to the user from where a client can select various options made available to the user. These options enable the user to customize the implementation configurations itself. Client.py file implements the peer to peer message transfer and file transfer option in the client side of the program. It runs on the device on which we wish to implement the peer to peer file and message transfer. It has one class, called the Client, which functionally stores all the information associated with a peer. It has functions to listen for file request queries and the main thread, run(), which checks for the keyword ‘getusers’ in the chats a peer sends (henceforth referred to as the sender) to the receiver. On receiving this command, it displays all the

clients that are currently connected to the server. It also checks on the commands for broadcasting, multicasting, unicasting and block casting messages and files. If any command in files such as fboradcast, fmulticast, funicast, fblockcast is present in the message from clients, it is considered as the stimulus to initiate the file transfer. In case of file transfer,we initiate a UDP connection by calling the udpclient.py function. Encryption of data is done at the sender side. Decryption of data is done at the receiving client side since we get the encrypted data.

4. **Client_udp.py:**
   It contains the functions used by the requester, to initiate and respond to the UDP file transfer connection, that comes in from the server_udp. It has six functions: recv_file, udp_receive, send_data,rcv_data,encryptmessage,decrypt message and send_data1. The functions such as send_data, send_data1, encryptmessage and decryptmessage are common to client_udp.py and server_udp.py. This is established by creating a UDP connection and binds to the UDP port numbers. recv_file and udp_recv functions are used to receive incoming files in the form of packets by using Go Back N algorithm. Send_data function is used to initiate the sending of file. Encryption is used to encrypt the data at the sender client side and Decryption is used to decrypt the encrypted data at the receiving client side. Client_udp function is called for the functionality of file transfer among connected peers.

5. **Server_udp.py:**
   It contains the functions which are utilized by the sending client to initiate a UDP connection to the requesting client. It has six functions: send_data which is responsible for sending of data messages and calling send_file function if a file is to be sent. Send_file function is meant for the transfer of files among the connected peers by breaking down the information into packets and implements Go Back N protocol for the same to ensure reliability. rcv_data function is used to receive all the incoming requests from the clients that are connected to the server. encryptmessage function is used for encrypting the message or file. Decryptmessage is used for decrypting the received data either message or file.

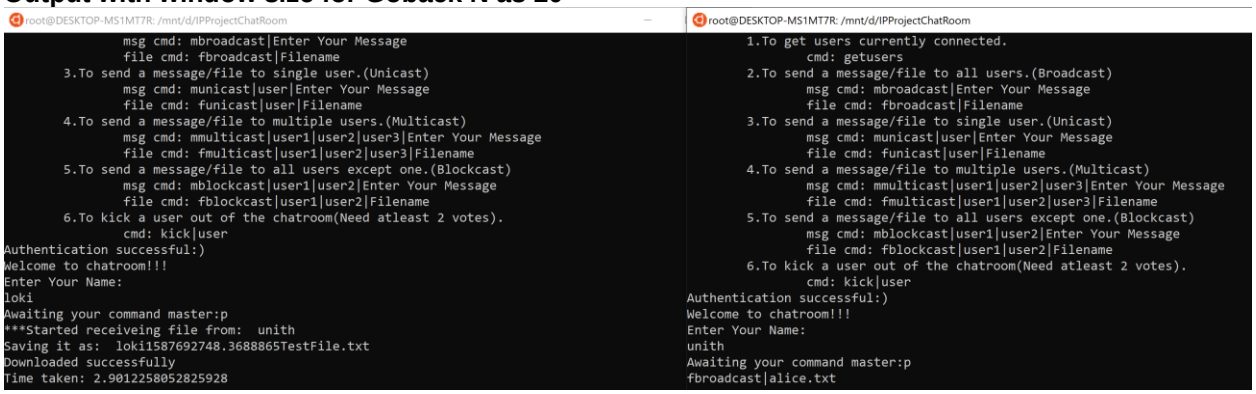
Results and Discussion

In case of Broadcast and Multicast sending files parallely using multithreading significantly reduced transfer time compared with sequentially sending of files to each user.
Unicast user1 – t seconds
Unicast user2 – t seconds
Unicast user3 – t seconds
Total time = t + t + t = 3t

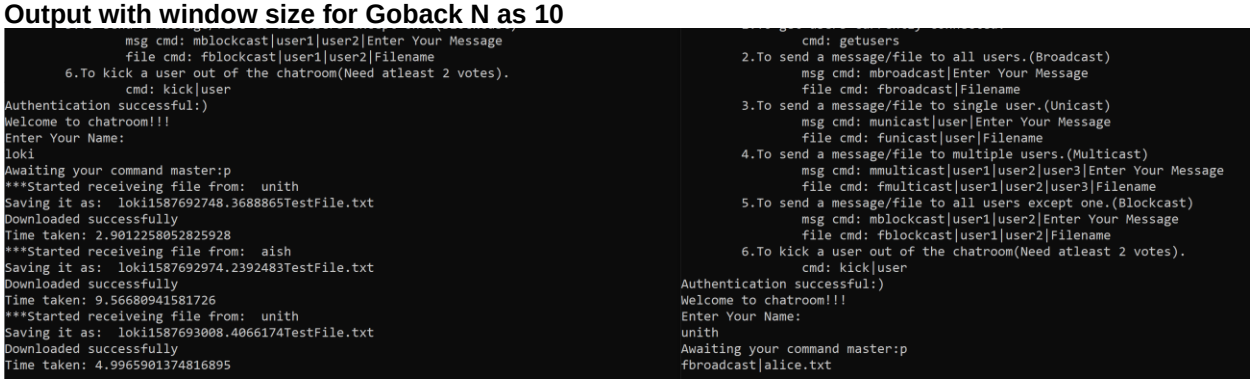
Broadcast or Multicast (User1 , User2, User 3)
Total time = t seconds

One key observation is that, as expected, by increasing the window size, time taken for file transfer is reduced. The following screenshots depict this point.

Output with window size for Goback N as 20

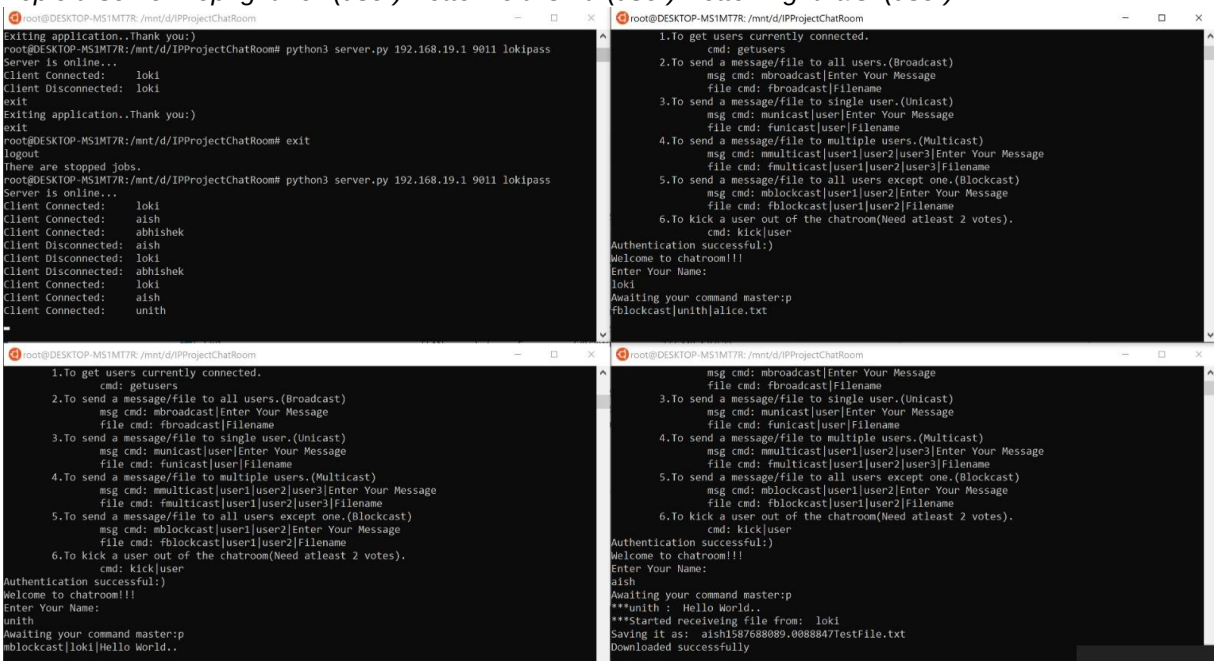


Output with window size for Goback N as 10



Blockcast: Sender: loki, Blocked:unith, Receiver: aish

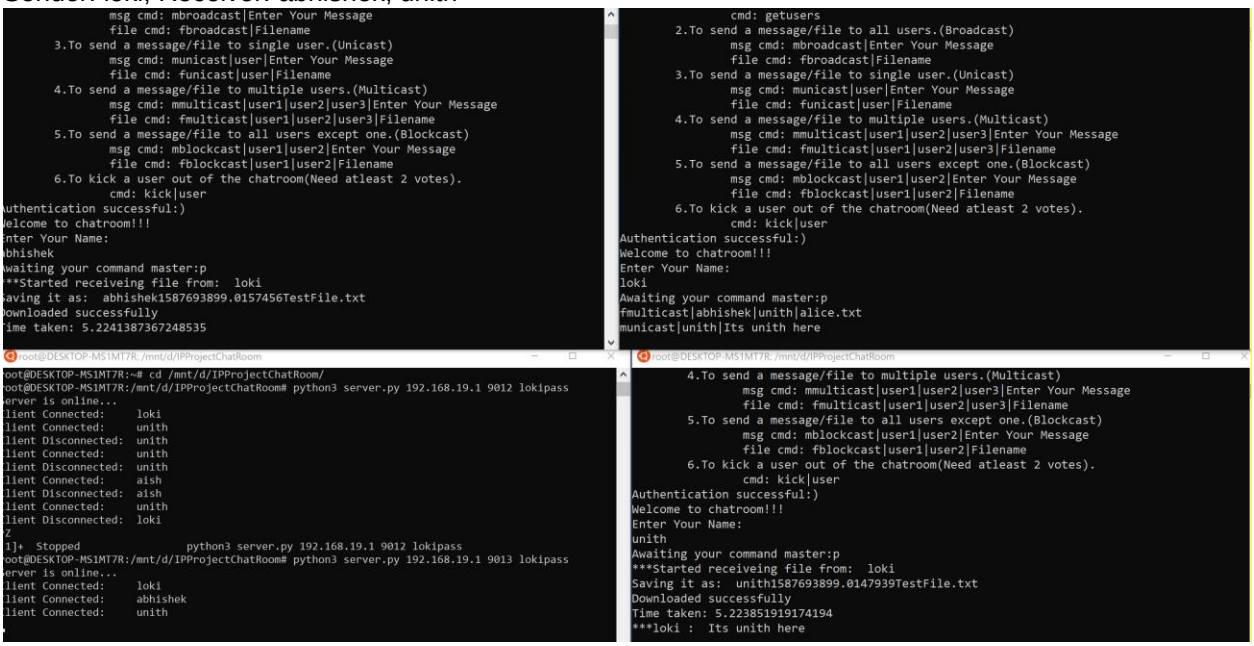
Topleft: server Topright: loki(user) Bottomleft: Unith(user) Bottomright: aish(user)



Multicast and Unicast output:

Topleft: Abhishek(user) Topright: loki(user) Bottomleft: Server Bottomright: unith(user)

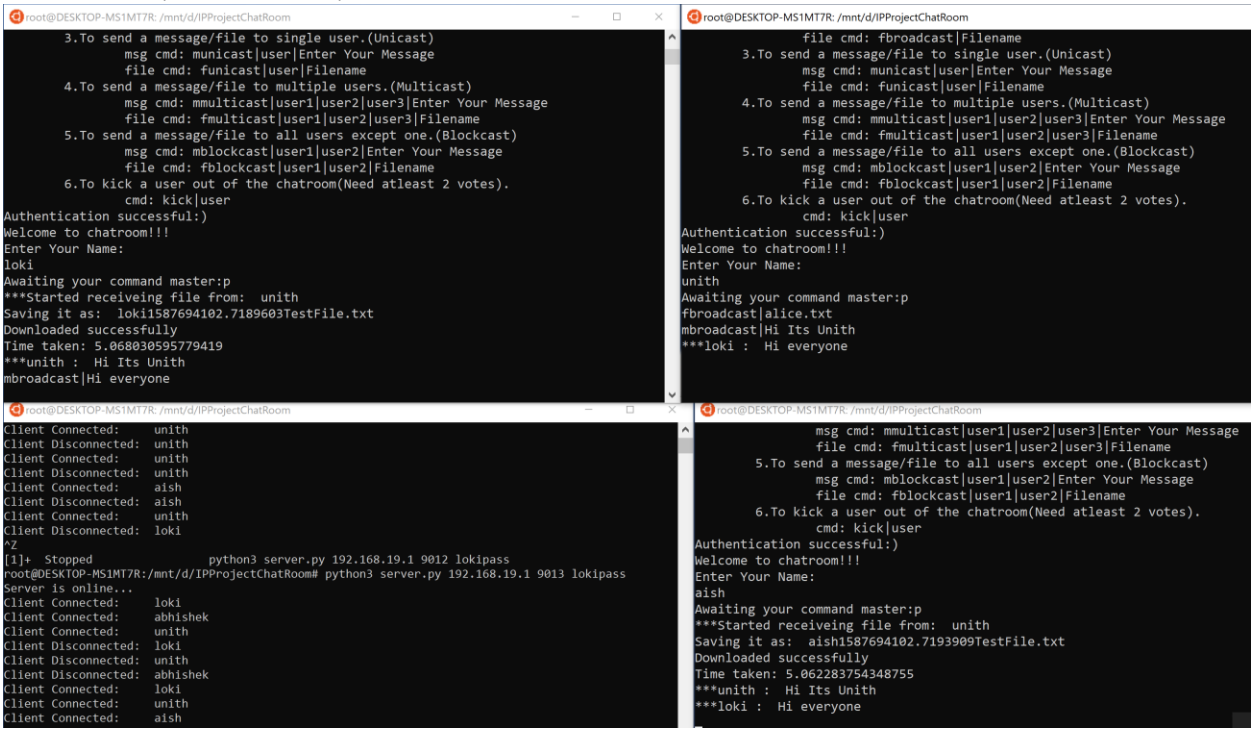
Sender: loki, Receiver: abhishek, unith



Broadcast

Topleft: loki(user) Topright: unith(user) Bottomleft: Server Bottomright: aish(user)

Sender: unith, Receiver: loki, aish



Related work and References

- Computer Networking: A Top-Down Approach, Seventh Edition by Pearson Kurose, Ross
- https://www.tutorialspoint.com/python/python_multithreading.htm
- https://docs.python.org/3/howto/sockets.html
- https://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python
- https://ieeexplore.ieee.org/document/5486192
- https://streamsets.com/documentation/datacollector/latest/help/datacollector/UserGuide/Origins/UDPMulti.html
- https://stackoverflow.com/