CSC3002S

NETWORKS 1

MNGLUT008 MDXROA001 MDXROA001

Introduction

This network application provides client-server communication using TCP sockets that allow P2P client-client connection using UD2P sockets. Clients can transmit and receive messages or files to and from each other.

Overview of the system functionality and features

The server runs on a host machine, allowing clients to connect via TCP sockets. The server listens for incoming connections from clients and maintains a list of connected clients and their statuses (online/offline). The client will specify the IP address and port number for the server they want to connect to.

Once connected to the server the clients can send requests abstracted by a numbered menu; 1 - to list the other clients connected to the server, 2 - to connect to available clients for p2p chats, 3 - to change their availability status and 4 - to exit the application and disconnect from the server.

When a client wants to connect to another they will be able to see a list of available clients — 'available' means that they are connected to the server and have an *<<ONLINE>>* status. Once a client selects the user they want to initiate a private chat with, the server will send a chat request. The other client can accept or deny this request. If they decide to accept this request, they will be prompted to enter their username and then once both clients have done so, they are placed in a chat room where they can send and receive messages/files.

The messages sent between clients via UDP are encrypted. The system leverages an encryption library for secure communication. A key is generated on the server upon establishing a UDP connection. The same encryption key is used between the clients.

Protocol Design and Specification

Protocol Design Breakdown:

This server uses a custom protocol for communication between itself and clients. The protocol involves various message formats and functionalities.

Client requests:

The server interprets messages received from clients based on specific message headers.

- ***L:*** Requests a list of all connected clients.
- ***C:*** Requests a list of available clients (those marked online).
- ***$Stts$:*** Requests a change in the client's online status.
- ***CON:*** Initiates a chat request with another specific client.

- ***Y:*** Indicates acceptance of a chat request from another client.
- ***N***: Indicates rejection of a chat request from another client.
- ***E:*** Signals disconnection from the server.

Server responses:

The server sends back specific responses based on the client's request:

- The client list (for L and C)
- Confirmation/rejection messages for chat requests (for Y and N)
- Information for initiating the peer-to-peer connection (upon successful chat request)
- UDP connection setup for direct peer-to-peer communication.
  - It provides each client with the other's IP address, port, and the temporary UDP port to be used for chat.

Server implementation

Listening for connections: The server listens for incoming TCP connection requests from clients on specific port 17280.

Client management: Upon accepting a client connection, the server stores the client's socket and IP address in a list (conClients). It also maintains a separate list (client_statuses) to track the online/offline status of each connected client.

Base Client implementation

Connecting to server: The client establishes a TCP connection with the server on the designated port.

Sending requests: The client socket sends messages to the server with the appropriate prefix to indicate the desired action (list clients, connect to another client, change status, etc.).

Receiving responses: The client receives responses from the server and interprets them based on the message content. It updates its internal state (client list, chat status) based on server responses.

Chat functionality:

When a client requests to connect with another (CON), the server forwards the request to the target client. If a chat request is accepted, the server shares necessary details (client names, addresses, port numbers, encryption key) and the client uses the provided information to establish a UDP connection with the other client using the temporary UDP port.

The client utilises the shared secret key for encryption and decryption of messages during the chat session.

Overall, this protocol facilitates communication between the server and clients, manages client connections, and enables secure peer-to-peer chat functionality.

Sending and Receiving files:

Socket Setup:

Creates a TCP socket using socket(AF_INET, SOCK_STREAM).

Binds the socket to a specific IP address and port (ip and 9999).

Listens for incoming connections with receiver_sock.listen(1).

Accepting Connection:

Accepts an incoming connection from a client using receiver_sock.accept() and retrieves the client's address

Receiving File Metadata:

Receives the file name and size from the client.

Decodes the received data using UTF-8 encoding.

Receiving File Data:

Opens a file in binary write mode using open(file_name, "wb").

Receives file data in chunks of 1024 bytes.

Writes received data to the file until the entire file is received.

Progress Tracking:

Utilises tqdm module import to display a progress bar indicating the file transfer progress.

Confirmation and Cleanup:

Sends a completion message to the client after file reception and closes the file sending socket and the receiver socket.

Lutho Client implementation

What makes my client implementation different is its visually appealing chat aesthetic which is reminiscent of popular messaging apps. Messages are neatly organised, with outgoing messages on the left and incoming messages on the right , these different types of messages are also colour coded differently. This enhances user familiarity and leverages the aesthetic usability effect as users tend to associate a more appealing user interface with better functionality. Additionally, in the main menu page, I made the offline/online state labels colour coded to green and red respectively to provide visual cues for system state.

To implement this , I created a python file chatAesthetic.py that I imported in my client code and used as a utility. I defined several functions such as left() for colour coding outgoing messages and right() for colour coding incoming messages. I imported the **shutil** library for terminal size determination which I used to take care of all space padding involved and the colour coding.

Chulumanco Client implementation

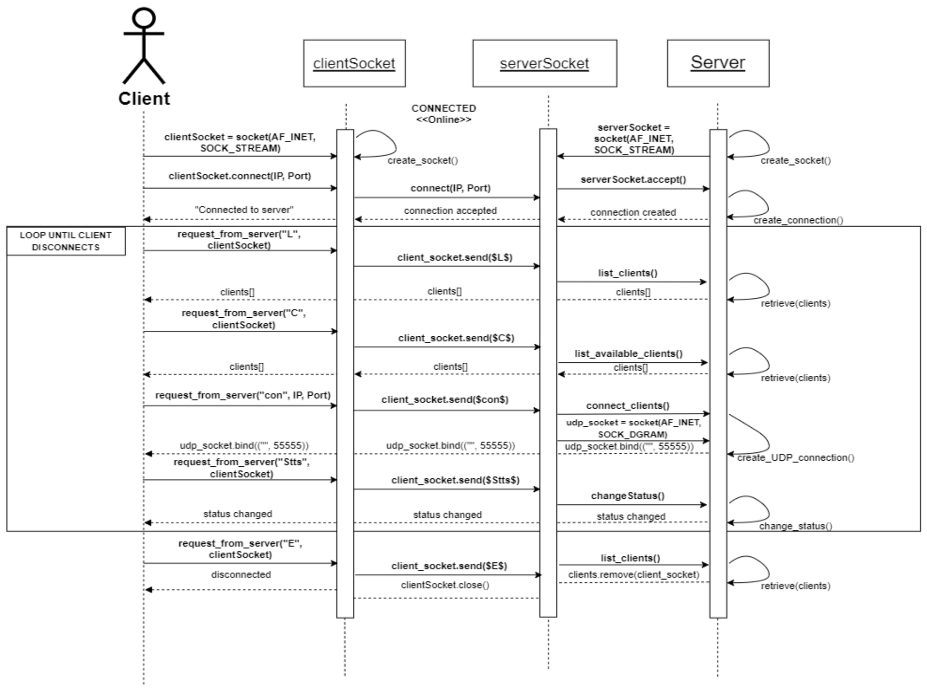
My client implementation is designed to make connecting with friends easy. Instead of technical terminology, the client uses straight forward commands like “List of available friends”, “Connect to a friend”, and “Change active friendship status”, this makes navigation easy and understandable. Moreover, terms like “Connect to a friend”, highlight a social aspect of the client. It’s designed to encourage interaction and make building connections feel effortless. I think this creates a welcoming atmosphere and helps users feel comfortable using the client.

Roanda Client implementation

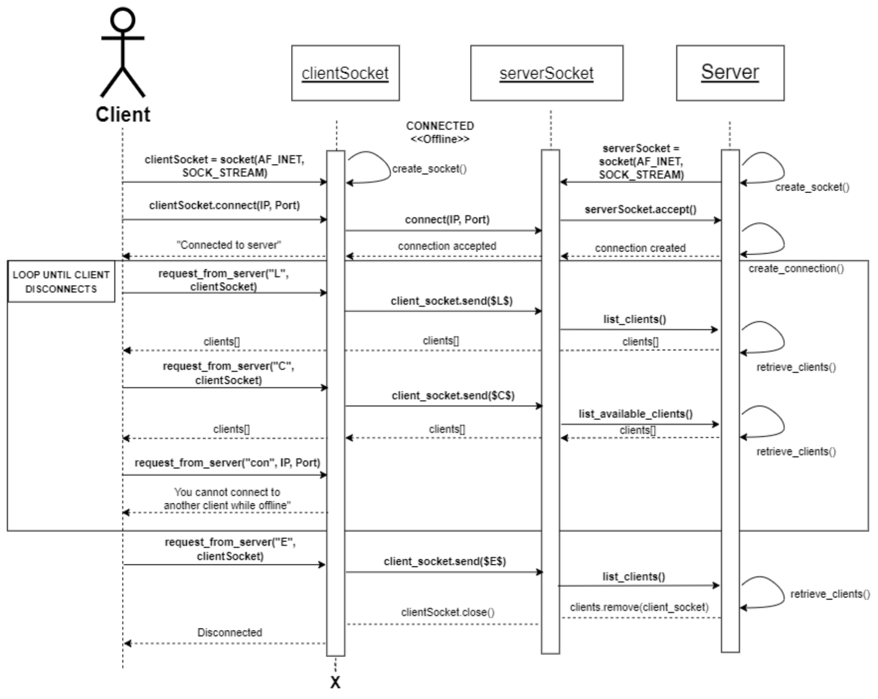
Maintains the logic of the base client implementation and provides encapsulation for more readable code. Terminal UI gives a more professional feel to the user and chat is minimalistic and visually appealing to enhance the clarity for use and abstract the protocol implementation from the user.

Sequence Diagrams

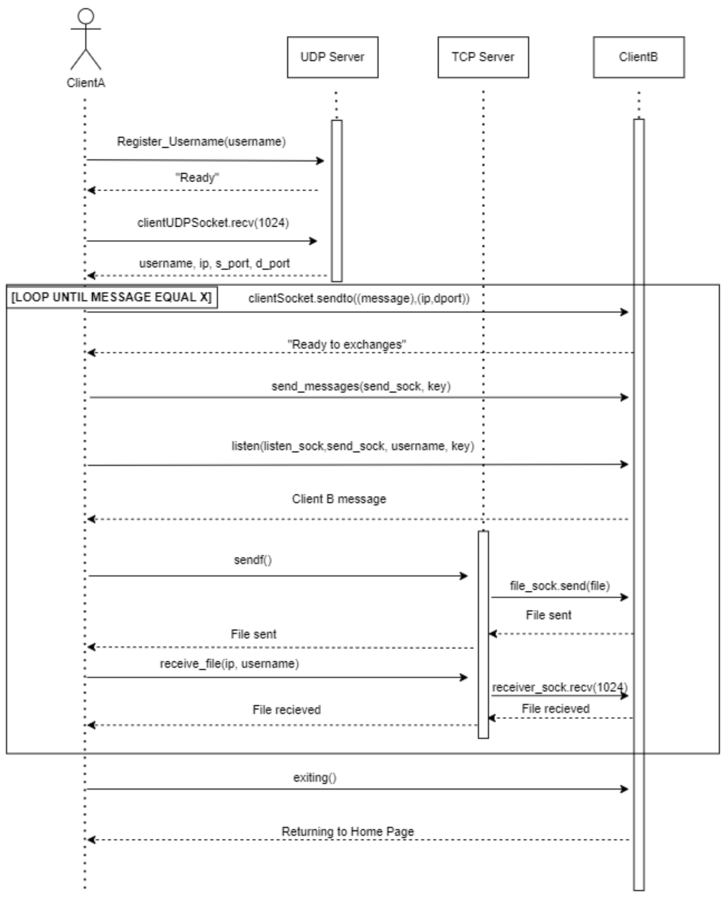
Client - Server: <<ONLINE>> state



Client - Server: <<OFFLINE>> state

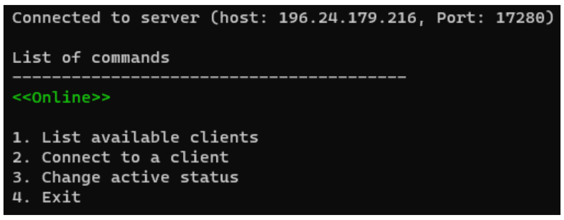


Client - Client: Chatting


ClientA
UDP Server
TCP Server
ClientB
Register_Username(username)
"Ready"
clientUDPSocket.recv(1024)
username, ip, s_port, d_port
[LOOP UNTIL MESSAGE EQUAL X]
clientSocket.sendto((message),(ip,dport))
"Ready to exchanges"
send_messages(send_sock, key)
listen(listen_sock,send_sock, username, key)
Client B message
sendf()
file_sock.send(file)
File sent
File sent
receive_file(ip, username)
receiver_sock.recv(1024)
File recieved
File recieved
exiting()
Returning to Home Page

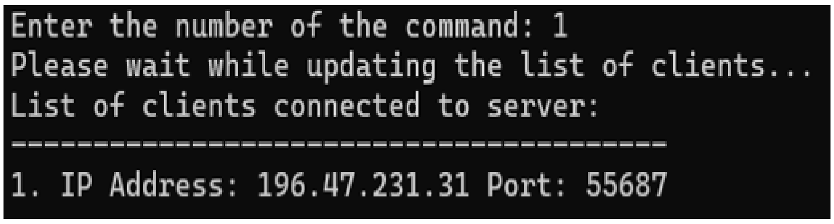
Examples and screenshots

MAIN MENU



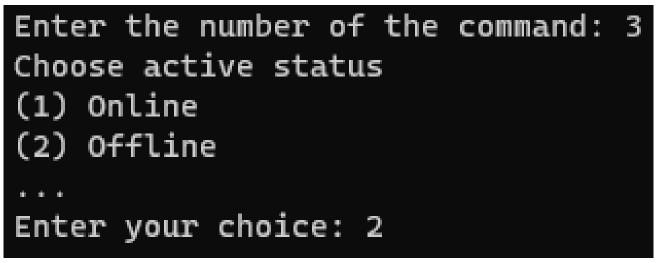
```
Connected to server (host: 196.24.179.216, Port: 17280)

List of commands
----------------------------------------
<<Online>>

1. List available clients
2. Connect to a client
3. Change active status
4. Exit
```

Connected to server



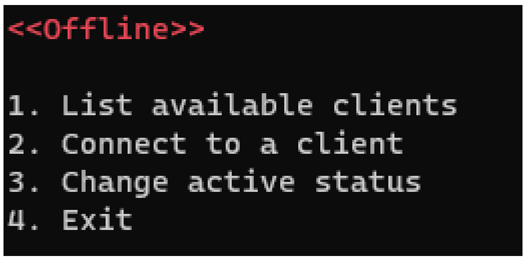
```
Enter the number of the command: 1
Please wait while updating the list of clients...
List of clients connected to server:
----------------------------------------
1. IP Address: 196.47.231.31 Port: 55687
```

Clicked 1. To view list of clients.



```
Enter the number of the command: 3
Choose active status
(1) Online
(2) Offline
...
Enter your choice: 2
```

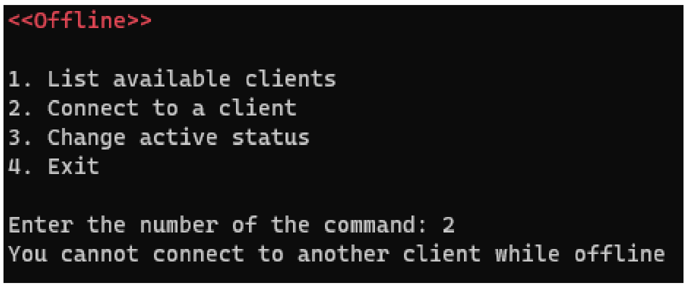
Clicked 3. To change active status to Offline.



```
<<Offline>>

1. List available clients
2. Connect to a client
3. Change active status
4. Exit
```

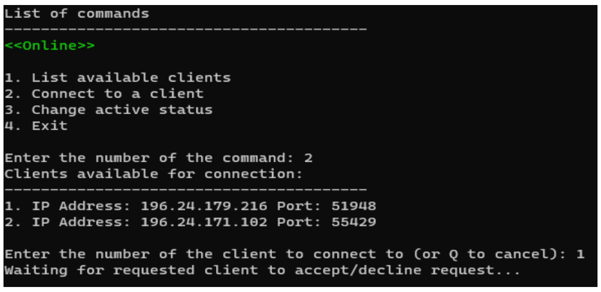
Changed status to “<<Offline>>”

CONNECTING TO CLIENTS



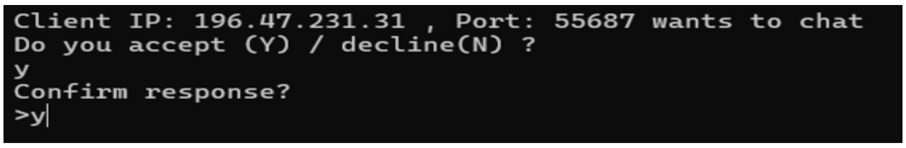
```
<<Offline>>

1. List available clients
2. Connect to a client
3. Change active status
4. Exit

Enter the number of the command: 2
You cannot connect to another client while offline
```

Attempting to connect clients while offline



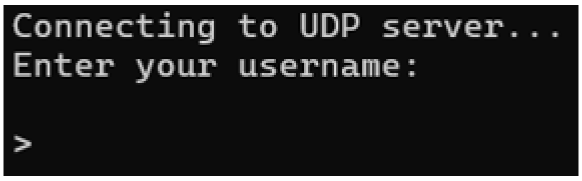
```
List of commands
----------------------------------------
<<Online>>

1. List available clients
2. Connect to a client
3. Change active status
4. Exit

Enter the number of the command: 2
Clients available for connection:
----------------------------------------
1. IP Address: 196.24.179.216 Port: 51948
2. IP Address: 196.24.171.102 Port: 55429

Enter the number of the client to connect to (or Q to cancel): 1
Waiting for requested client to accept/decline request...
```

Connecting to clients while online



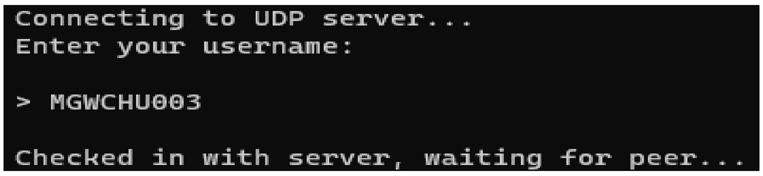
```
Client IP: 196.47.231.31 , Port: 55687 wants to chat
Do you accept (Y) / decline(N) ?
y
Confirm response?
>y
```

Client receiving chat request



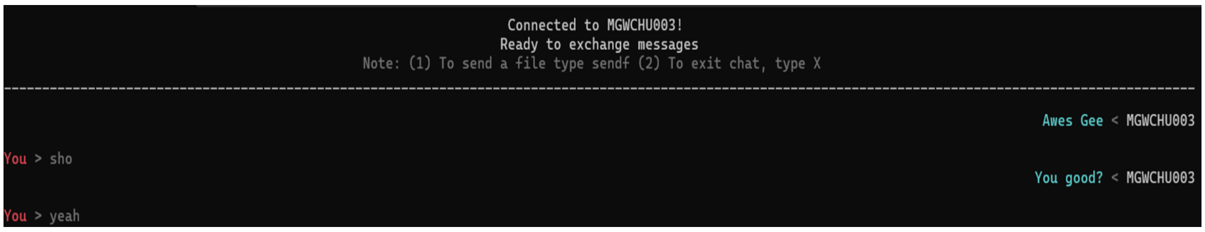
```
Connecting to UDP server...
Enter your username:

>
```

Client accepted chat request and prompted to enter username



```
Connecting to UDP server...
Enter your username:

> MGWCHU003

Checked in with server, waiting for peer...
```

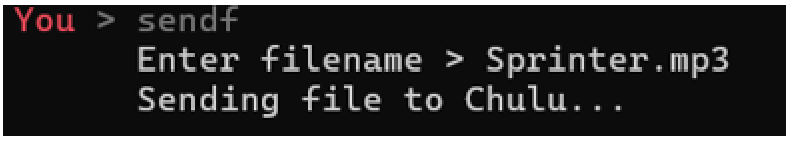
Client entered username and waits for peer to connect



```
                              Connected to MGWCHU003!
                              Ready to exchange messages
              Note: (1) To send a file type sendf (2) To exit chat, type X
------------------------------------------------------------------------------------

                                                           Awes Gee < MGWCHU003
You > sho
                                                           You good? < MGWCHU003

You > yeah
```

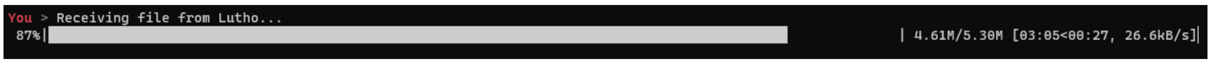
Clients are connected and chatting via UDP

FILE SENDING



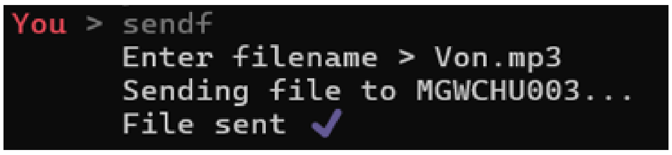
```
You > sendf
      Enter filename > Sprinter.mp3
      Sending file to Chulu...
```

Sendf ~ client X sending file “Sprinter.mp3” to client Y via TCP



```
You > Receiving file from Lutho...
 87%|████████████████████████████████████                | 4.61M/5.30M [03:05<00:27, 26.6kB/s]
```

client Y receiving file “Sprinter.mp3” from client X



```
You > sendf
      Enter filename > Von.mp3
      Sending file to MGWCHU003...
      File sent ✓
```

File sent