

Report: Chat Application using TCP and UDP Socket Programming in C++

Name: Ismail Hafeez

Course: Computer Networks

Semester: Fall 2025

Project Title: Chat Application using TCP and UDP Sockets in C++

1. System Design

1.1 Overview

The developed system is a dual-mode **Chat Application** that enables multiple users to communicate using either **TCP (Transmission Control Protocol)** or **UDP (User Datagram Protocol)**.

The TCP mode provides **reliable, connection-oriented messaging**, while the UDP mode supports **faster, connectionless communication**.

The chat supports:

- **User registration and authentication**
- **Private and broadcast messages**
- **File transfer**
- **Status updates** (online/away/busy)
- **Message history and logging**
- Command-based interaction (/msg, /file, /status, /help, /exit)

1.2 Architecture

Component Description

Server	Central node that manages connected clients, authentication, and message routing. Logs all activity to a file.
Client	Allows users to send commands and interact with others. It communicates with the server using either TCP or UDP.

1.3 TCP Mode Design

- **Connection-Oriented:** Each client connects via a persistent socket (`connect()` / `accept()`).
- **Reliability:** Ensures ordered, error-free data delivery.
- **Concurrency:** The server uses `select()` to handle multiple clients simultaneously.
- **Persistence:** Each message and event is logged in `server.log`.

1.4 UDP Mode Design

- **Connectionless:** The server uses recvfrom() and sendto(); no explicit connections.
- **Lightweight:** Faster but less reliable—suitable for small chat messages.
- **Stateless Communication:** Each packet carries its own addressing info.
- **Simplified Authentication:** Server maps username to the latest IP:port of the client.

1.5 Comparison: TCP vs UDP

Feature	TCP	UDP
Connection	Connection-oriented	Connectionless
Reliability	Guaranteed delivery, ordered packets	No guarantee of delivery or order
Speed	Slower (due to acknowledgments)	Faster (no handshake)
Use Case	Secure, multi-client chat	Lightweight, broadcast chat
Implementation Complexity	High (multi-client handling, select/poll)	Moderate (simple packet routing)

2. Code Explanation

2.1 TCP Server

File: server.cpp

Function	Description
main()	Initializes socket, binds port, listens for clients, and uses select() for concurrent connections.
process_line()	Parses each client command (AUTH, REGISTER, MSG, STATUS, FILE, etc.).
broadcast()	Sends messages to all connected clients except sender.
send_private()	Delivers private message to a specific user.
append_log()	Logs each event and also displays it in the terminal.
handle_command()	Handles slash commands (/msg, /status, /history, /exit, etc.).

Data Structures:

- ClientInfo struct stores each client's username, socket fd, and status.
- std::map<int, ClientInfo> and std::map<std::string, int> manage active connections.

2.2 TCP Client

File: client.cpp

- Connects to the server via TCP (connect()).
 - Prompts user for REGISTER or AUTH.
 - Runs a receiver thread to listen for messages.
 - Parses slash commands to format requests for the server.
 - Supports graceful exit (/exit).
-

2.3 UDP Server

File: udp_server.cpp

Function	Description
main()	Creates UDP socket, binds it, and continuously listens for datagrams.
broadcast_msg()	Sends a message to all online users.
load_users() / save_user()	Handles persistent user data.
append_log()	Displays and records events.
Message Handlers	Handle REGISTER, AUTH, MSG, PRIV, FILE, STATUS, LEAVE.

2.4 UDP Client

File: udp_client.cpp

- Creates a UDP socket and connects logically by saving the server's address.
- Prompts for REGISTER or AUTH.
- After login, shows the same /menu as the TCP version.
- Sends and receives using sendto() and recvfrom() in parallel threads.
- Automatically sets status to *away* on exit.

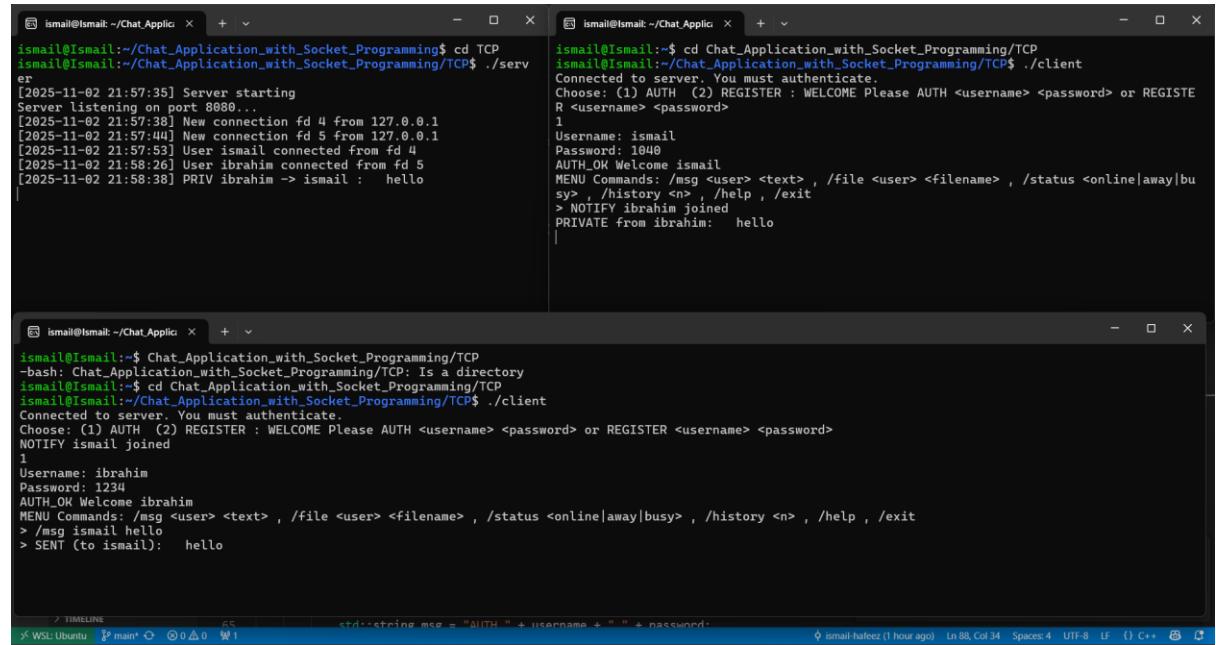
3. Challenges Faced

Challenge	Explanation	Resolution
Handling multiple clients	Managing concurrent connections in TCP without threads	Used select() for multiplexing.
Authentication in UDP	No connection state tracking	Mapped username → IP:port in a

Challenge	Explanation	Resolution
Reliable file transfer in UDP	UDP can lose packets	Added Base64 and confirmation messages (optional).
Syncing logs and terminal	Needed unified logging system	Modified append_log() to print and log simultaneously.
Maintaining consistent menus	Ensuring both TCP and UDP offer same user commands	Standardized /menu output on both.

4. Testing Results

TCP



The screenshot displays three terminal windows on a Linux desktop environment. The top-left window shows the server side starting up and accepting connections from two clients. The top-right window shows a client connecting and performing a registration. The bottom window shows another client connecting and sending a message to the first client.

```

ismail@Ismail:~/Chat_Application_with_Socket_Programming$ cd TCP
ismail@Ismail:~/Chat_Application_with_Socket_Programming/TCP$ ./server
[2025-11-02 21:57:35] Server starting
Server listening on port 8080...
[2025-11-02 21:57:38] New connection fd 4 from 127.0.0.1
[2025-11-02 21:57:44] New connection fd 5 from 127.0.0.1
[2025-11-02 21:57:53] User ismail connected from fd 4
[2025-11-02 21:58:26] User ibrahim connected from fd 5
[2025-11-02 21:58:38] PRIV ibrahim > ismail : hello
|
```



```

ismail@Ismail:~/Chat_Application_with_Socket_Programming$ cd Chat_Application_with_Socket_Programming/TCP
ismail@Ismail:~/Chat_Application_with_Socket_Programming/TCP$ ./client
Connected to server. You must authenticate.
Choose: (1) AUTH (2) REGISTER : WELCOME Please AUTH <username> <password> or REGISTER <username> <password>
1
Username: ismail
Password: 1040
AUTH_OK Welcome ismail
MENU Commands: /msg <user> <text> , /file <user> <filename> , /status <online|away|busy> , /history <n> , /help , /exit
> NOTIFY ibrahim joined
PRIVATE from ibrahim: hello
|
```



```

ismail@Ismail:~/Chat_Application_with_Socket_Programming$ Chat_Application_with_Socket_Programming/TCP
-bash: Chat_Application_with_Socket_Programming/TCP: Is a directory
ismail@Ismail:~/Chat_Application_with_Socket_Programming/TCP$ cd Chat_Application_with_Socket_Programming/TCP
ismail@Ismail:~/Chat_Application_with_Socket_Programming/TCP$ ./client
Connected to server. You must authenticate.
Choose: (1) AUTH (2) REGISTER : WELCOME Please AUTH <username> <password> or REGISTER <username> <password>
NOTIFY ismail joined
1
Username: ibrahim
Password: 1234
AUTH_OK Welcome ibrahim
MENU Commands: /msg <user> <text> , /file <user> <filename> , /status <online|away|busy> , /history <n> , /help , /exit
> /msg ismail hello
> SENT (to ismail): hello
|
```

UDP

```

ismail@Ismail:~/Chat_Applic... $ ./server
[2025-11-02 21:59:37] UDP Server started on port 9090
[2025-11-02 22:00:02] ismail logged in
[2025-11-02 22:00:09] ibrahim logged in
[2025-11-02 22:00:18] PRIV ismail -> ibrahim: hi wassup
[2025-11-02 22:00:34] MSG ibrahim: /file ismail server.cpp
[2025-11-02 22:00:56] MSG ismail: /history 2
| 

ismail@Ismail:~/Chat_Applic... $ ./client
Connected to UDP server.
Choose: (1) AUTH (2) REGISTER : 1
Username: ismail
Password: 1040
>
AUTH_OK Welcome ismail
>
MENU Commands: /msg <user> <text>, /file <user> <filename>, /status <s>, /history <n>
, /help, /exit
>
NOTIFY ibrahim joined
> /msg ibrahim hi wassup
>
ibrahim: /file ismail server.cpp
> /history 2
> | 

ismail@Ismail:~/Chat_Applic... $ ./client
Connected to UDP server.
Choose: (1) AUTH (2) REGISTER : 1
Username: ibrahim
Password: 1234
>
AUTH_OK Welcome ibrahim
>
MENU Commands: /msg <user> <text>, /file <user> <filename>, /status <s>, /history <n>, /help, /exit
>
PRIVATE from ismail: hi wassup
> /file ismail server.cpp
>
ismail: /history 2
> | 

```

The screenshot shows three terminal windows. The top-left window is the UDP server log, showing connections from 'ismail' and 'ibrahim'. The top-right window is a client session where 'ismail' logs in and sends a message to 'ibrahim'. The bottom-left window is another client session where 'ibrahim' logs in and receives a message from 'ismail'. The bottom-right window shows the terminal status bar with WSL:Ubuntu, main, and other system information.

Wireshark

No.	Time	Source	Destination	Protocol	Length Info
1	0.000000	172.25.208.1	224.0.0.251	NDNS	207 Standard query response 0x0000 PTR Ismail._dovsc._tcp.local SRV 0 0 7680 Ismail.local TXT
2	0.000655	fe80::d6dc:6cb:bf3c:ff02::fb	224.0.0.251	NDNS	277 Standard query response 0x0000 PTR Ismail._dovsc._tcp.local SRV 0 0 7680 Ismail.local TXT
3	0.001458	172.25.208.1	224.0.0.251	NDNS	84 Standard query 0x0000 ANY Ismail._dovsc._tcp.local, "Q" question
4	0.002018	fe80::d6dc:6cb:bf3c:ff02::fb	224.0.0.251	NDNS	104 Standard query 0x0000 ANY Ismail._dovsc._tcp.local, "Q" question
5	0.267785	172.25.208.1	224.0.0.251	NDNS	84 Standard query 0x0000 ANY Ismail._dovsc._tcp.local, "Q" question
6	0.268345	fe80::d6dc:6cb:bf3c:ff02::fb	224.0.0.251	NDNS	104 Standard query 0x0000 ANY Ismail._dovsc._tcp.local, "Q" question
7	0.523350	172.25.208.1	224.0.0.251	NDNS	84 Standard query 0x0000 ANY Ismail._dovsc._tcp.local, "Q" question
8	0.523911	fe80::d6dc:6cb:bf3c:ff02::fb	224.0.0.251	NDNS	104 Standard query 0x0000 ANY Ismail._dovsc._tcp.local, "Q" question
9	0.677580	172.25.208.1	224.0.0.251	NDNS	263 Standard query response 0x0000 PTR cache flush Ismail._dovsc._tcp.local SRV, cache flush A, c...
10	0.778444	fe80::d6dc:6cb:bf3c:ff02::fb	224.0.0.251	NDNS	203 Standard query response 0x0000 PTR cache flush Ismail._dovsc._tcp.local SRV, cache flush A, c...
11	0.778993	172.25.208.1	224.0.0.251	NDNS	208 Standard query response 0x0000 PTR cache flush Ismail._dovsc._tcp.local SRV, cache flush 0 0 7680 Ismail.local TXT, cache flush A, cache flush ...
12	0.779567	fe80::d6dc:6cb:bf3c:ff02::fb	224.0.0.251	NDNS	228 Standard query response 0x0000 PTR cache flush 0 0 7680 Ismail.local TXT, cache flush A, cache flush ...
13	14.472828	172.25.208.32	185.125.190.58	NTP	90 NTP Version 4, client
14	14.634518	185.125.190.58	172.25.208.32	NTP	90 NTP Version 4, server
15	19.428693	Microsoft_1e:a5:bb	Microsoft_ab:44:36	ARP	42 Who has 172.25.208.32? Tell 172.25.208.1
16	19.429093	Microsoft_ab:44:36	Microsoft_1e:a5:bb	ARP	42 172.25.208.32 is at 00:15:5d:ab:44:36
17	19.573713	Microsoft_ab:44:36	Microsoft_1e:a5:bb	ARP	42 Who has 172.25.208.17 Tell 172.25.208.32
18	19.573744	Microsoft_1e:a5:bb	Microsoft_ab:44:36	ARP	42 172.25.208.1 is at 00:15:5d:1e:a5:bb

The screenshot shows a Wireshark capture of DNS traffic. It includes responses from a Microsoft host (185.125.190.58) and multiple requests from a client (172.25.208.1). The traffic includes standard queries, PTR responses, and NTP requests. The bottom window shows the TCP server log starting and accepting a connection from 127.0.0.1.

```

ismail@Ismail:~/Chat_Applic... $ ./server
[2025-11-02 22:02:21] Server starting
Server listening on port 8080...
[2025-11-02 22:02:30] New connection fd 4 from 127.0.0.1

```

