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.

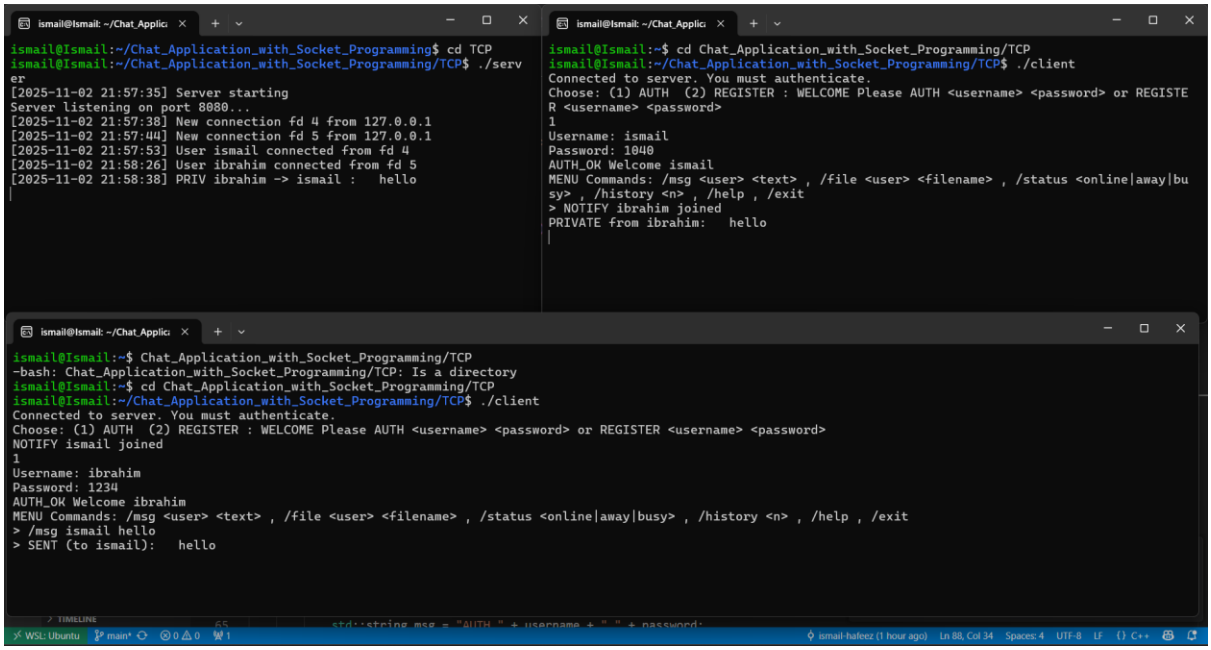
3. Challenges Faced

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

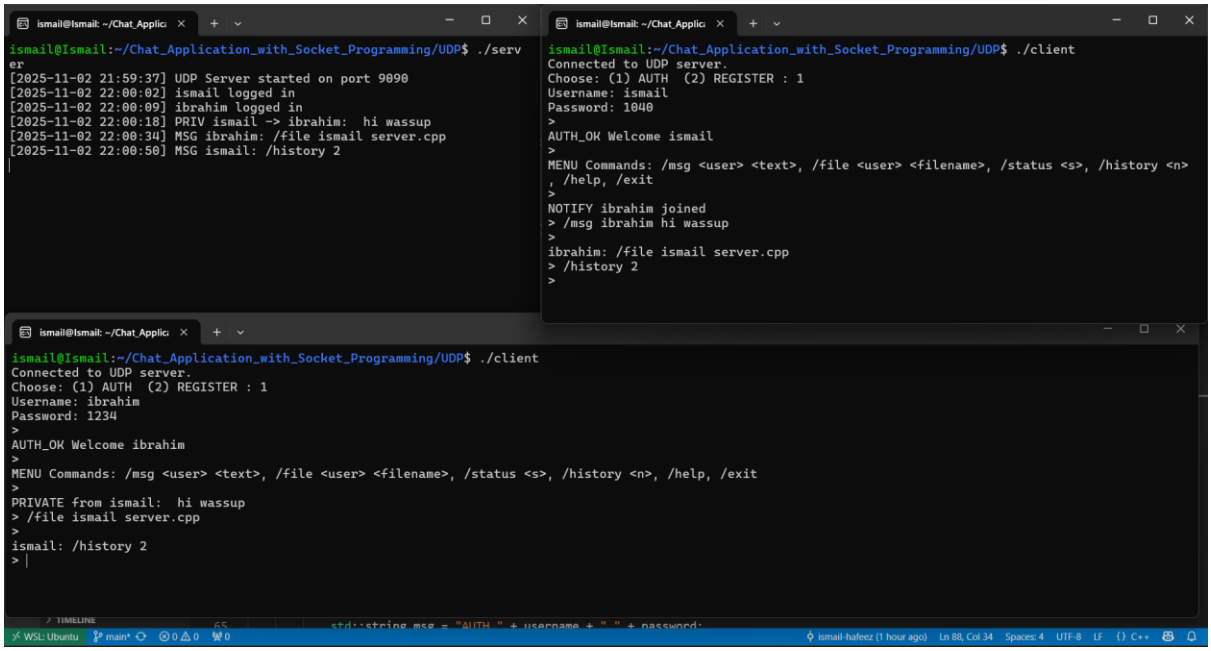
Challenge	Explanation	Resolution
		dictionary.
Reliable file transfer in UDP	UDP can lose packets	Added Base64 and confirmation messages (optional).
Syncing logs and terminal	Needed unified logging system	Modified append_log() to print and log simultaneously.
Maintaining consistent menus	Ensuring both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

4. Testing Results

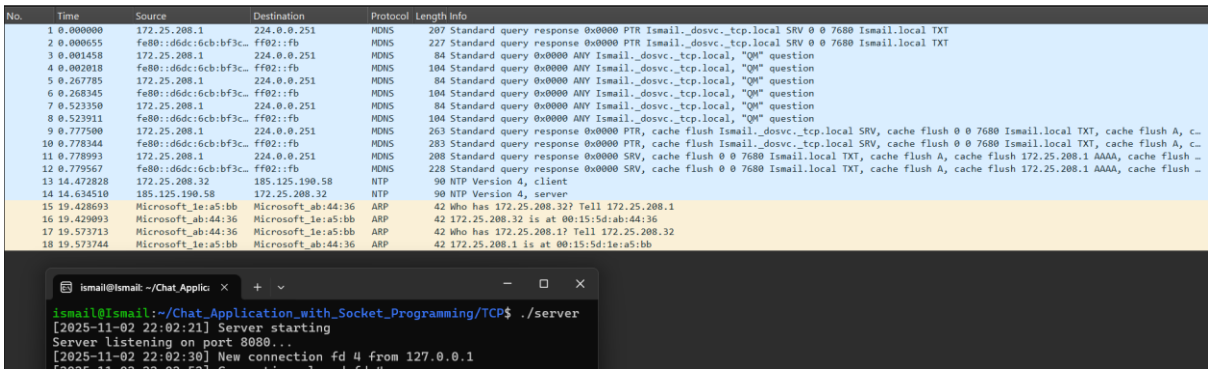
TCP

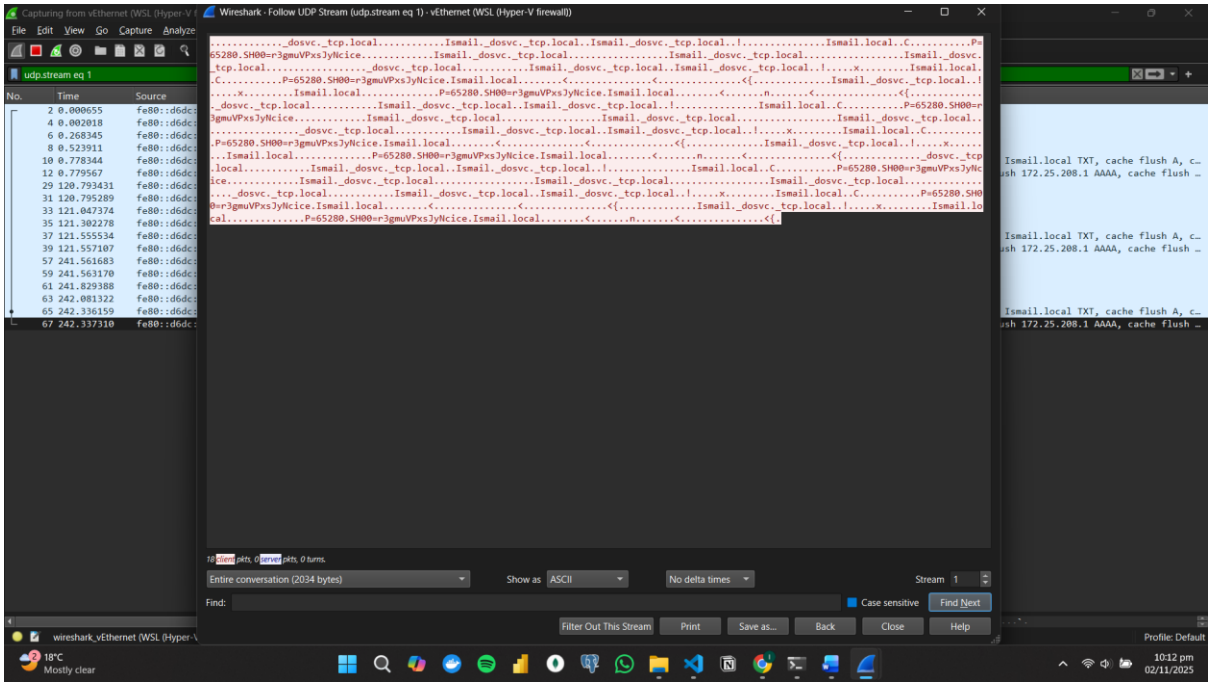


UDP



Wireshark




Wireshark - Follow UDP Stream (udp.stream eq 1) - vEthernet (WSL (Hyper-V firewall))
Capturing from vEthernet (WSL (Hyper-V f
File Edit View Go Capture Analyze
udp.stream eq 1
No. Time Source
2 0.000655 fe80::d6dc:
4 0.002018 fe80::d6dc:
6 0.268345 fe80::d6dc:
8 0.523911 fe80::d6dc:
10 0.778344 fe80::d6dc:
12 0.779567 fe80::d6dc:
29 120.793431 fe80::d6dc:
31 120.795289 fe80::d6dc:
33 121.047374 fe80::d6dc:
35 121.302278 fe80::d6dc:
37 121.555534 fe80::d6dc:
39 121.557107 fe80::d6dc:
57 241.561683 fe80::d6dc:
59 241.563170 fe80::d6dc:
61 241.829388 fe80::d6dc:
63 242.081322 fe80::d6dc:
65 242.336159 fe80::d6dc:
67 242.337310 fe80::d6dc:
Ismail.local TXT, cache flush A, c…
ush 172.25.208.1 AAAA, cache flush …
Ismail.local TXT, cache flush A, c…
ush 172.25.208.1 AAAA, cache flush …
Ismail.local TXT, cache flush A, c…
ush 172.25.208.1 AAAA, cache flush …
18 client pkts, 0 server pkts, 0 turns.
Entire conversation (2034 bytes)
Show as ASCII
No delta times
Stream 1
Find:
Case sensitive
Find Next
Filter Out This Stream
Print
Save as...
Back
Close
Help
wireshark_vEthernet (WSL (Hyper-V
Profile: Default
18°C
Mostly clear
10:12 pm
02/11/2025