Graphic Era Hill University
DEHRADUN • BHIMTAL • HALDWANI

PROJECT AND TEAM INFORMATION

Project Title

(Try to choose a catchy title. Max 20 words).

StreamLine Chat – Seamless conversations through multithreaded TCP connectivity.

Student/Team Information

Team Name:	TBA
Team member 1 (Team Lead) (Name, Student ID, Email, Picture):	Akshit –220111979 Akshitsharma02003@gmail.com 
Team member 2 (Name, Student ID, Email, Picture):	Bisht , Shobhit – 220111631 shobhitbisht2703@gmail.com 

Team member 3 (Name, Student ID, Email, Picture):	Rawat , Tanish – 220111278 tanishrawat10@gmail.com 
Team member 4 (Name, Student ID, Email, Picture):	Chauhan, RajVardhan Singh- 220111066 rajvardhansevenb@gmail.com 

PROJECT PROGRESS DESCRIPTION

Project Abstract

(Brief restatement of your project's main goal. Max 300 words).

StreamLine Chat is a C++-based real-time chat application that utilizes multithreaded TCP connectivity to enable seamless bidirectional communication between multiple clients and a central server. The system prioritizes responsiveness and low-latency message handling, leveraging the WinSock2 API on Windows. We have implemented a clean separation of responsibilities within the codebase: socket management, thread execution, and user input/output are modularized for efficiency and extensibility. A key addition to our infrastructure is the integration of a custom virtual private network (VPN) using Tailscale Tailnet, allowing all chat clients and the server to communicate using consistent, private IP addresses—ideal for testing and secure messaging across remote devices. This design forms a reliable foundation for scalable communication tools while remaining educational for networking and concurrency learning.

Updated Project Approach and Architecture

(Describe your current approach, including system design, communication protocols, libraries used, etc. Max 300 words).

Our architecture includes a dedicated server and multiple clients, connected over a Tailscale VPN to ensure seamless communication even in remote environments. The server listens for incoming connections and launches a new thread for each client, ensuring non-blocking operations. Each client maintains two threads—one for receiving and another for sending messages—using C++ std::thread for concurrency.

Key technologies:

- **C++** for core logic
- **WinSock2** for socket-level networking
- **Tailscale Tailnet** for creating a personal virtual IP network (VPN)
- **std::mutex** for synchronization
- **Threading** to prevent UI freeze and ensure smooth I/O operations

The use of a virtual private network guarantees consistent and discoverable IPs for all devices, streamlining connection and testing phases. The system is currently console-based but modular enough to support GUI layers in the future.

Tasks Completed

(Describe the main tasks that have been assigned and already completed. Max 250 words).

Task Completed	Team Member
WinSock2 Initialization and Cleanup	Akshit, Shobhit
Server Socket Binding and Listening	Akshit, Shobhit
Multithreaded Client Communication Setup	Tanish, RajVardhan Singh
Chat Message Broadcasting Logic (Server-side)	Shobhit, Akshit
Console Interface with Real-time Updates	Tanish, RajVardhan Singh
Personal VPN Setup using Tailscale	Akshit

Challenges/Roadblocks

(Describe the challenges that you have faced or are facing so far and how you plan to solve them. Max 300 words).

IP Address Discovery Across Networks: One of the biggest hurdles was ensuring all clients could reach the server outside of a LAN setup. This was solved by configuring a private virtual network using Tailscale Tailnet.

Thread Synchronization Bugs: During concurrent message handling, race conditions occasionally caused corrupted outputs or deadlocks. This was mitigated using careful mutex management for the client and console streams.

Cross-platform Limitations: Since WinSock2 is Windows-specific, portability to Linux/macOS is deferred. Future versions may adopt Boost.Asio or similar cross-platform libraries.

Disconnection Detection: Handling unexpected disconnects and cleaning up threads/sockets was challenging. This has been mostly resolved with robust error-checking on recv() and send().

Tasks Pending

(Describe the main tasks that you still need to complete. Max 250 words).

Task Pending	Team Member (to complete the task)
GUI implementation (optional using Qt/ImGui)	RajVardhan Singh
Encryption layer using OpenSSL	Shobhit Bisht
Reconnection Logic and Client Auto-Retry	Akshit Sharma
Server-side command parser for admin tools	Tanish Rawat
Linux/macOS compatibility via Boost.Asio	Team (future goal)

Project Outcome/Deliverables

(Describe what are the key outcomes / deliverables of the project. Max 200 words).

A multithreaded TCP chat client/server in C++
Real-time message broadcasting with no UI lag
Use of Tailscale VPN for virtual networking
Modular, clean code structure with educational value
Codebase prepared for GUI and security upgrades
Extensive documentation on concurrency, sockets, and VPN setup

Progress Overview

(Summarize how much of the project is done, what's behind schedule, what's ahead of schedule. Max 200 words.)

We have successfully completed the core server and client functionalities, including multithreaded communication, console I/O, and a virtual private network setup. Message broadcasting and concurrent user interaction have been thoroughly tested and validated. Some advanced features like encryption and GUI remain pending. The project is on schedule for core deliverables, with optional enhancements still under consideration.

Codebase Information

(Repository link, branch, and information about important commits.)

Repository: *(If hosted online, insert GitHub/GitLab link here)*
Branch: main
Important Commits:

- Init multithreaded chat server – Server-side socket and threading
- Client-side console and sender/receiver threads – Complete TCP client
- Add Tailscale IP handling and VPN support – Integrated private virtual network

Testing and Validation Status

(Provide information about any tests conducted)

Test Type	Status (Pass/Fail)	Notes
Local LAN Connection Test	PASS	Works on same Wi-Fi network
Cross-Network via Tailnet	PASS	Clients on VPN connect successfully
Concurrent User Messaging	PASS	No delays with 3+ users simultaneously
Disconnection Handling	PASS	Graceful cleanup on server and client sides
High Message Volume Stress	PASS	System stable under rapid message inputs

Deliverables Progress

(Summarize the current status of all key project deliverables mentioned earlier. Indicate whether each deliverable is completed, in progress, or pending.)

Server-side TCP listener	Completed
Client multithreaded messaging	Completed
Console I/O and mutex sync	Completed
Tailnet VPN integration	Completed
Encryption implementation	Completed
Cross-Device compatibility	Completed
GUI frontend (Qt/ImGui)	In Progress
Cross-platform compatibility	Pending