REPORT - AASTHA SHARMA (z5444661)

COMP3331 Assignment

Student Name: Aastha Sharma
Student ID: z5444661

For this assignment, I designed a simple but fully working discussion forum system using a client-server architecture. The server handles all control operations over UDP and uses TCP specifically for transferring files like uploads and downloads. Originally, the server was designed to be single-threaded, but later I extended it to support multiple concurrent clients properly by using threading and queues, so multiple people can interact with the server at the same time without stepping over each other.

The project is split into several files to keep everything organized.

- `server.py` manages the main UDP server loop, handles authentication, and dispatches client commands.
- `functions.py` contains the actual logic for all the commands like creating posts, uploading files, listing posts, etc.
- `client.py` deals with user interaction, sends requests over UDP, and sets up TCP connections for file transfers when needed.

Communication between the client and server is done using a very simple text-based protocol. For login, the client first sends the username, waits for an ACK from the server, then sends the password, and waits for another ACK. Only after that the server responds with either a success message ("Welcome!") or an error. Once authenticated, the client sends commands prefixed with the username, and the server parses them and executes the correct action. Commands follow formats like `CRT <threadTitle>`, `MSG <threadTitle> <message>`, `LST`, `RDT <threadTitle>`, and so on.

One thing I worked carefully on was making sure the system could **handle packet loss** during UDP transmission. To deal with that, I added a reliable sending mechanism where the client waits for an ACK after every UDP message it sends. If an ACK doesn't come back within a timeout, it automatically retries sending the message up to five times before giving up. I tested this using the provided lossy proxy that simulates random packet drops, and the system held up pretty well even with around 10% packet loss. Login, thread creation, messaging and other commands still go through most of the time without issues.

When it comes to concurrency, after a user logs in successfully, the server spawns a new thread specifically to handle that client. Each client thread listens to a private queue, and commands coming from that client are routed into their queue for processing. This way, multiple users can be logged in and performing actions at the same time, without interfering with each other. I will

not lie, it was a little tricky to manage the shared client state (like tracking which clients are active), but using dictionaries and proper checks made it manageable.

For the design decisions, I kept things pretty simple. Separating the server-side command logic into `functions.py` to keep the main server loop lighter and more readable.

There are a few limitations though. Although multiple clients can now log in and interact concurrently, the system doesn't yet support things like client timeouts or full session persistence after a crash. If a client disconnects, their session state is cleaned up. I also didn’t implement heavy-duty input validation beyond basic checks like not allowing empty usernames or passwords.

In terms of references, I mainly followed Python's official documentation for socket programming and basic file handling functions. I didn’t copy any external code for the core logic, everything is built from scratch based on concepts taught in lectures and tutorials, with occasional syntax checks from the Python docs.

Overall, I'm pretty happy with how it turned out. The forum system supports multiple users, file uploads and downloads work reliably over TCP, and packet loss handling makes the UDP communication stable even under slightly messy network conditions. It’s simple but covers everything the assignment asked for.
