COMP9331 Assignment Report

z5611182
Ishu Rajput

Introduction

This report explains how we built a basic online discussion forum as part of the COMP9331 assignment. The system is designed like a real-world forum where users can log in, create threads, post messages, and even upload or download files. It works using a client-server model, where one server talks to multiple clients. We used **UDP** for handling most of the user commands because it's faster, and **TCP** for file transfers because it's more reliable.

Program Design

The program implements a client-server forum system using a combination of UDP and TCP protocols. The design follows a request-response pattern where clients send commands to the server, and the server processes these commands and sends back responses.

Server Design

The `ForumServer` class handles all server-side operations. It maintains state for authenticated users, active threads, and pending authentication. The server listens on the same port for both UDP commands and TCP file transfers.

Client Design

The `ForumClient` class provides a command-line interface for users. It handles user authentication and command processing. It establishes UDP connections for commands and TCP connections for file transfers.

Data Structures

Server Data Structures

- `active_users`: A set to track currently logged-in users
- `threads`: A dictionary mapping thread titles to their creators
- `request_queue`: A queue for handling client requests asynchronously
- `pending_auth`: A dictionary to track users in the authentication process

File Storage

- User credentials are stored in a plain text file (`credentials.txt`)
- Each thread is stored as a separate file with the thread title as the filename
- The first line of a thread file contains the username of the creator
- Messages are stored with a number, username, and content
- Uploaded files are stored with a naming convention of {`thread_title`}-{`filename`}

Application Layer Protocol

The system implements a custom application layer protocol over UDP for commands and TCP for file transfers.

Authentication Protocol

- Client sends `"AUTH:{username}"`
- Server responds with:
  - `"PASSWORD:"` (for existing users)
  - `"NEWUSER:"` (for new users)
  - `"ERROR:"` (if username is already logged in)
- Client sends password
- Server validates and responds with `"SUCCESS:"` or `"ERROR:"`

Command Protocol

- All commands follow a similar pattern:
  - Client sends a command with arguments
  - Server processes the command and sends a response starting with `"SUCCESS:"` or `"ERROR:"`
- The protocol supports these commands:
  - `CRT`: Create a new thread
  - `LST`: List all active threads
  - `MSG`: Post a message to a thread
  - `DLT`: Delete a message from a thread
  - `EDT`: Edit a message in a thread
  - `RDT`: Read all messages in a thread
  - `RMV`: Remove an entire thread (creator only)
  - `XIT`: Log out from the system
  - `UPD`: Upload a file to a thread (TCP)
  - `DWN`: Download a file from a thread (TCP)

File Transfer Protocol

For file transfers (`UPD` and `DWN` commands):

- Client sends command via UDP
- Server approves via UDP if conditions are met
- TCP connection is established for the file transfer
- Data is transferred in chunks
- Connection is closed after transfer
- Server sends final confirmation via UDP

Trade-offs Considered

UDP vs. TCP

- UDP for commands: Chosen for its simplicity and lower overhead for small messages
- TCP for file transfers: Chosen for reliability and built-in flow control for larger data transfers

Thread Storage

- Individual files for threads: Simple to implement and maintain
- Trade-off: Less efficient than a database but easier to implement and debug

Authentication

- Plain text credentials: Simple but not secure
- Trade-off: Security sacrificed for simplicity

Potential Issues

- **Security Concerns**: Passwords are stored in plain text, no encryption, vulnerable to spoofing and replay attacks.
- **Error Handling**: Some errors like packet loss during UDP and TCP protocol (e.g., during file transfers) may not be caught properly.
- **Socket Management**: TCP socket reuse could cause conflicts during simultaneous file transfers.
- **Resource Management**: No limits on thread/message/file size; risk of exhaustion.
- **Authentication Weaknesses**: No session timeout.