ChatConnect Application: Inter-Hostel Technical General Championship

Nupur Rajesh Unavekar

October 2023 - November 2023

Abstract

ChatConnect is a Python-based application developed for the Inter-Hostel Technical General Championship. The application facilitates file sharing and client communication within a network, utilizing a central server to manage interactions. This report details the design, implementation, and testing of the application, along with the challenges encountered and their solutions.

1 Introduction

The Inter-Hostel Technical General Championship required a robust application for real-time file sharing and communication among participants. ChatConnect was developed to meet this need, drawing inspiration from the PSP file-sharing framework. The primary goal was to create an application capable of handling multiple clients concurrently, ensuring secure and efficient communication.

2 Methodology

2.1 System Design

ChatConnect employs a client-server architecture, with a central server responsible for managing communication between clients. The server listens for incoming connections and relays messages and files between connected clients.

2.2 Communication Protocols

The application utilizes two primary protocols:

- **UDP (User Datagram Protocol)**: Used for control messages due to its low latency and connectionless nature.
- **TCP (Transmission Control Protocol)**: Employed for file transfers to ensure reliable and ordered delivery of data.

2.3 Threading Model

To manage multiple clients concurrently, the server spawns two threads for each client:

- **Receiver Thread**: Listens for incoming messages from the client.
- **Sender Thread**: Handles outgoing messages to other clients.

This lock-free threading model ensures smooth and efficient communication without bottlenecks.

2.4 User Interface

The user interface was developed using Kivy, a Python framework for multitouch applications. The UI allows users to send and receive messages, view the chat log, and manage file transfers in a user-friendly environment.

3 Implementation

The application was implemented in Python, leveraging the socket library for network communication. The server and client scripts were developed with modularity in mind, allowing for easy maintenance and scalability.

3.1 Server Implementation

The server script initializes a TCP socket to listen for incoming connections on a specified port. Upon receiving a connection request, the server spawns threads to handle client communication and file transfers.

3.2 Client Implementation

The client script connects to the server using TCP and communicates with the server using predefined control messages. The client UI, built with Kivy, provides an interface for sending messages and managing files.

4 Testing and Results

ChatConnect was tested locally with a setup of 5 clients and 1 server. The application successfully handled simultaneous connections and file transfers without any data loss or connection drops. The Kivy-based UI was also tested for usability, ensuring a smooth user experience.

4.1 Performance Testing

Performance was evaluated based on the latency of control messages and the speed of file transfers. The results indicated that UDP provided minimal latency for control messages, while TCP ensured reliable file transfers with minimal overhead.

5 Challenges and Solutions

5.1 Concurrency Management

Managing multiple clients concurrently was a significant challenge. This was addressed by implementing a threading model that allocated two threads per client, ensuring that each client could send and receive messages independently.

5.2 UI Integration

Integrating the Kivy UI with the underlying communication logic required careful management of asynchronous events. This was achieved by ensuring that UI updates and network communication occurred on separate threads, preventing blocking operations.

6 Conclusion

ChatConnect successfully met the requirements of the Inter-Hostel Technical General Championship, providing a reliable and user-friendly platform for file sharing and communication. The application demonstrated robust performance in a local network setup, with the potential for further expansion and scalability.

7 Future Work

Future enhancements could include the addition of encryption for secure communication, support for larger networks with more clients, and the development of mobile versions of the application.

References

- Python Documentation: `https://docs.python.org/3/`
- Kivy Documentation: `https://kivy.org/doc/stable/`