Hybrid Centralized and Peer-to-Peer Chat System using Sockets

Nguyen Manh Khoi-BI13219
Nguyen Viet Minh Khoi-BI13221
Le Quang Huy-BI13190
Tran Manh Long-BI13267
Tran Hung Thinh-BI12428
Tran Hong Nhat-BA12143

December 31, 2024

1 Introduction

Communication systems are essential in nowaday world. The goal of this project is to build a chat system which can combine the benefits of centralized server and the flexibility of peer-to-peer (P2P) interactions. Centralized models provide easy management and messages broadcasting, while P2P models can make direct communication and reduced server dependency.

2 Methodology

The hybrid system using the following methodologies:

- **Centralized Communication**: Users connect to a central server for authentication and broadcasting messages to other connected clients.
- **Peer-to-Peer Communication**: Users can make a direct connections with peers for private messaging, without the central server.
- **Multithreading**: Both the server and clients use multithreading to handle multiple simultaneous connections efficiently.

The system architecture consists of two main components:

1. **Central Server**: Manages client connections, broadcasts messages, and provides a list of active users for P2P connections.
2. **Clients**: Communicate with the central server and establish P2P connections for private chats.

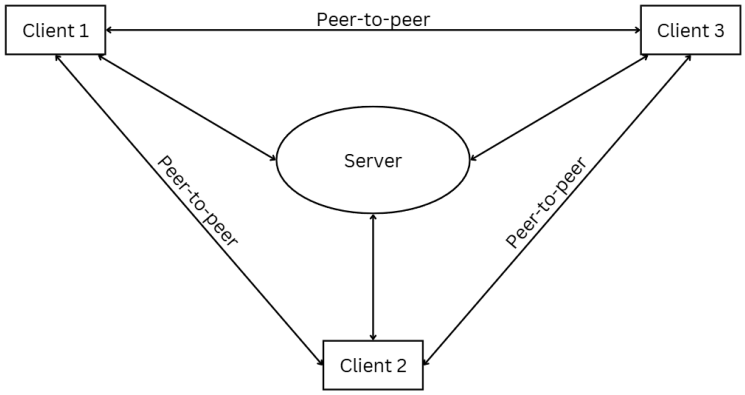

Figure 1: System Architecture Diagram

3 Implementation

The project is implemented in Python, using the socket and threading libraries for network communication. The central server and client functionalities are seperated in respective classes.

3.1 Central Server

The central server handles client connections, manages a list of active clients, and broadcasts messages. The following code snippet illustrates the server's core functionality:

```
class CentralServer:
    def __init__(self, host='127.0.0.1', port=12345):
```

```
        self.server_socket = socket.socket(socket.
            AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind((host, port))
        self.server_socket.listen(5)  # Maximum 5
            connections
        self.clients = {}  # Dictionary to store client
            connections

    def handle_client(self, client_socket,
        client_address):
        try:
            username = client_socket.recv(1024).decode()
            self.clients[username] = client_socket
            print(f"{username} connected from {
                client_address}")

            while True:
                message = client_socket.recv(1024).
                    decode()
                if message == 'PEER':
                    client_socket.send(str(list(self.
                        clients.keys())).encode())
                else:
                    self.broadcast(message, username)
        except:
            print(f"{username}disconnected.")
            del self.clients[username]
        finally:
            client_socket.close()

    def broadcast(self, message, username):
        for user, conn in self.clients.items():
            if user != username:
                try:
                    conn.send(f"{username}:{message}".
                        encode())
                except:
                    continue

    def start(self):
```

```
        print("Central server started...")
        while True:
            client_socket, client_address = self.
                server_socket.accept()
            threading.Thread(target=self.handle_client,
                args=(client_socket, client_address)).
                start()
```

Listing 1: Central Server

3.2 Peer-to-Peer Client

Clients communicate with the central server and manage direct connections with peers. Key functionalities include sending messages, receiving messages, and initiating P2P communication.

```
class PeerClient:
    def __init__(self, username, central_host='127.0.0.1
        ', central_port=12345):
        self.username = username
        self.central_socket = socket.socket(socket.
            AF_INET, socket.SOCK_STREAM)
        self.central_socket.connect((central_host,
            central_port))
        self.central_socket.send(username.encode())
        self.p2p_port = None  # Placeholder for P2P port

    def receive_messages(self):
        while True:
            try:
                message = self.central_socket.recv(1024)
                    .decode()
                print(message)
            except:
                print("Disconnected from server.")
                break

    def send_message(self, message):
        self.central_socket.send(message.encode())

```

```
21      def peer_to_peer(self, peer_host, peer_port, message
           =None):
22          try:
23              peer_socket = socket.socket(socket.AF_INET,
                   socket.SOCK_STREAM)
24              peer_socket.connect((peer_host, peer_port))
25              if message:
26                  peer_socket.send(message.encode())
27              threading.Thread(target=self.handle_peer,
                   args=(peer_socket,)).start()
28          except Exception as e:
29              print(f"Error connecting to peer: {e}")
30          # Establish P2P connection and send messages
31
32      def start_peer_listener(self, host='127.0.0.1', port
           =None):
33          if port is None:
34              port = int(input("Enter port for P2P
                   listener (default is 54321): ") or 54321)
35          self.p2p_port = port
36          listener_socket = socket.socket(socket.AF_INET,
               socket.SOCK_STREAM)
37          listener_socket.bind((host, port))
38          listener_socket.listen(1)
39          print(f"Listening for P2P connections on {host
               }:{port}...")
40          while True:
41              peer_socket, peer_address = listener_socket.
                   accept()
42              print(f"Connected to peer at {peer_address}"
                   )
43              threading.Thread(target=self.handle_peer,
                   args=(peer_socket,)).start()
44          # Listen for incoming P2P connections
```

Listing 2: Peer-to-Peer Client

After enter username, cient will be asked for a port for P2P connection(54321 by default). The other client can P2P communicate by this port and IP.

4 Results

The hybrid chat system was deployed successfully:

- Multiple clients successfully connected to the central server and sent messages.

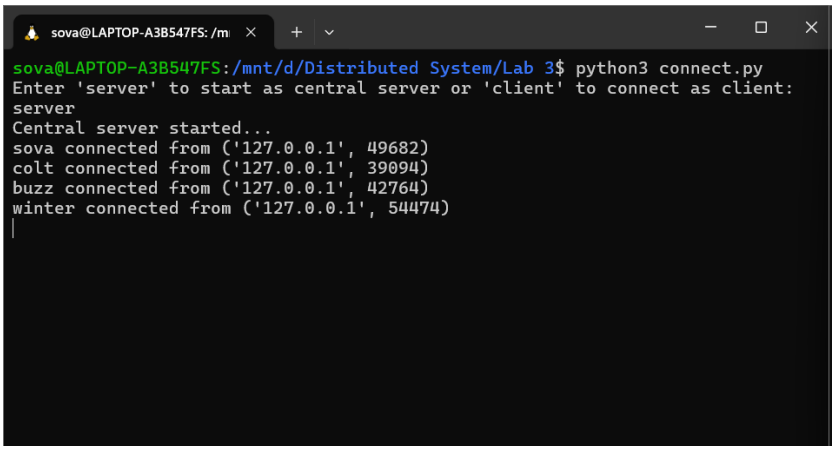

Figure 2: Clients are connected.

- Peer-to-peer communication was established, allowing private message between users.

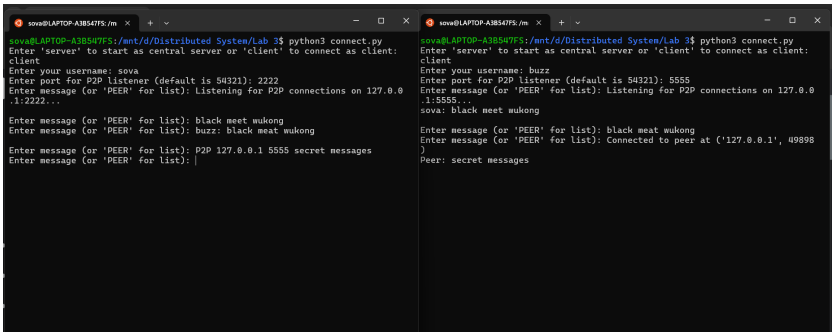

Figure 3: Private peer-to-peer communication demonstration.

- The system demonstrated scalability by handling concurrent client connections efficiently.

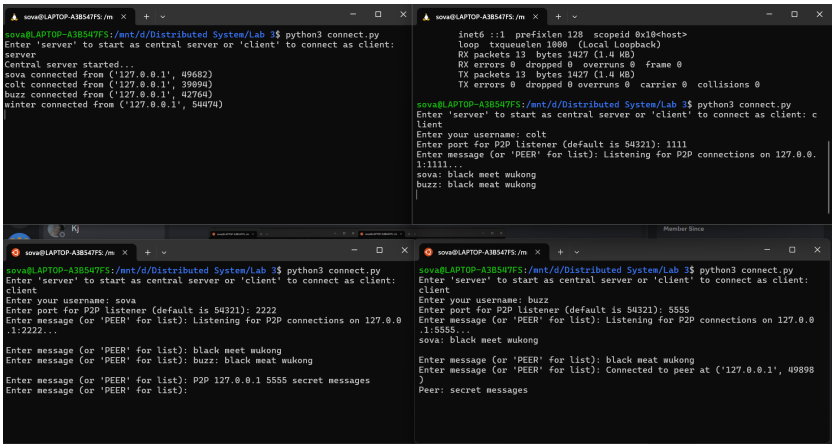

Figure 4: Scalability testing with multiple client connections.

5 Conclusion

This project successfully implemented a hybrid centralized and peer-to-peer chat system using socket.The combine of both models offers flexibility, robustness, and scalability. Future work could adding encryption for secure communication and enhancements for large-scale deployments.