

# 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 nowadays 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.

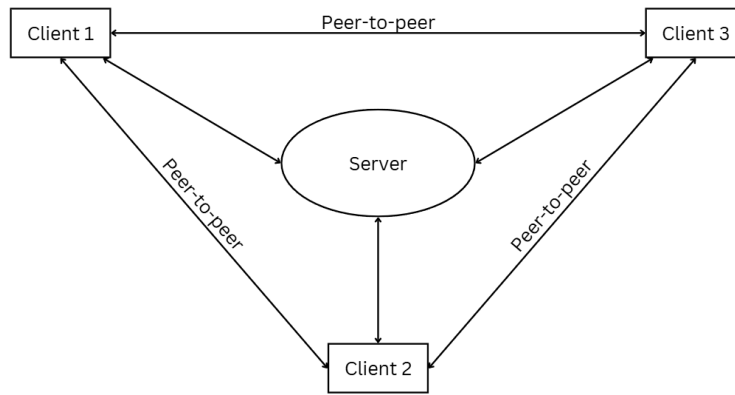


Figure 1: System Architecture Diagram

## 3 Implementation

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

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

```

3         self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4         self.server_socket.bind((host, port))
5         self.server_socket.listen(5) # Maximum 5
        connections
6         self.clients = {} # Dictionary to store client
        connections
7
8     def handle_client(self, client_socket,
        client_address):
9         try:
10             username = client_socket.recv(1024).decode()
11             self.clients[username] = client_socket
12             print(f"{username}_connected_from_{client_address}")
13
14             while True:
15                 message = client_socket.recv(1024).
                    decode()
16                 if message == 'PEER':
17                     client_socket.send(str(list(self.
                        clients.keys()))).encode())
18                 else:
19                     self.broadcast(message, username)
20             except:
21                 print(f"{username}disconnected.")
22                 del self.clients[username]
23             finally:
24                 client_socket.close()
25
26     def broadcast(self, message, username):
27         for user, conn in self.clients.items():
28             if user != username:
29                 try:
30                     conn.send(f"{username}:{message}".
                        encode())
31                 except:
32                     continue
33
34     def start(self):

```

```

35         print("Central server started...")
36         while True:
37             client_socket, client_address = self.
                server_socket.accept()
38             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.

```

1  class PeerClient:
2      def __init__(self, username, central_host='127.0.0.1',
        central_port=12345):
3          self.username = username
4          self.central_socket = socket.socket(socket.
            AF_INET, socket.SOCK_STREAM)
5          self.central_socket.connect((central_host,
            central_port))
6          self.central_socket.send(username.encode())
7          self.p2p_port = None # Placeholder for P2P port
8
9      def receive_messages(self):
10         while True:
11             try:
12                 message = self.central_socket.recv(1024)
13                     .decode()
14                 print(message)
15             except:
16                 print("Disconnected from server.")
17                 break
18
19         def send_message(self, message):
20             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

```

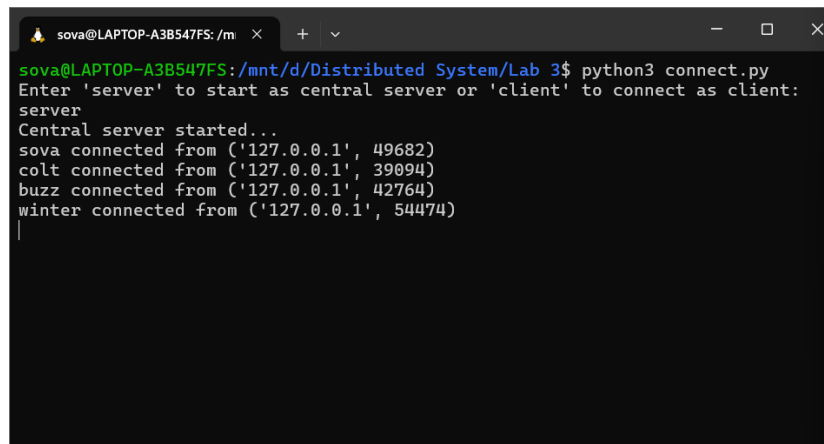
Listing 2: Peer-to-Peer Client

After enter username, client 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:

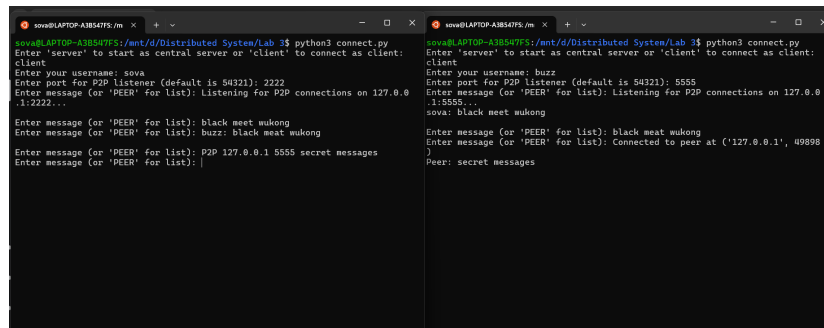
- Multiple clients successfully connected to the central server and sent messages.



```
sova@LAPTOP-A3B547FS: /m x + v
sova@LAPTOP-A3B547FS:/mnt/d/Distributed System/Lab 3$ python3 connect.py
Enter 'server' to start as central server or 'client' to connect as client:
server
Central server started...
sova connected from ('127.0.0.1', 49682)
colt connected from ('127.0.0.1', 39094)
buzz connected from ('127.0.0.1', 42764)
winter connected from ('127.0.0.1', 54474)
```

Figure 2: Clients are connected.

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



```
sova@LAPTOP-A3B547FS: /m x + v
sova@LAPTOP-A3B547FS:/mnt/d/Distributed System/Lab 3$ python3 connect.py
Enter 'server' to start as central server or 'client' to connect as client:
client
Enter your username: sova
Enter port for P2P listener (default is 50321): 2222
Enter message (or 'PEER' for list): Listening for P2P connections on 127.0.0.1:2222...
Enter message (or 'PEER' for list): black meet wukong
Enter message (or 'PEER' for list): buzz: black meat wukong
Enter message (or 'PEER' for list): P2P 127.0.0.1 5555 secret messages
Enter message (or 'PEER' for list): |

sova@LAPTOP-A3B547FS: /m x + v
sova@LAPTOP-A3B547FS:/mnt/d/Distributed System/Lab 3$ python3 connect.py
Enter 'server' to start as central server or 'client' to connect as client:
client
Enter your username: buzz
Enter port for P2P listener (default is 50321): 5555
Enter message (or 'PEER' for list): Listening for P2P connections on 127.0.0.1:5555...
sova: black meet wukong
Enter message (or 'PEER' for list): black meat wukong
Enter message (or 'PEER' for list): Connected to peer at ('127.0.0.1', 49898)
Peer: secret messages
```

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

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

The image displays four terminal windows arranged in a 2x2 grid, illustrating the scalability testing of a chat system. Each window shows the execution of a Python script named 'connect.py' within a directory structure that includes a 'Distributed System/Lab' folder. The top-left window shows the server starting and accepting connections from three clients: 'sova', 'colt', and 'buzz'. The top-right window shows network statistics for a loopback interface, indicating successful communication. The bottom-left window shows a client ('sova') sending a message ('black meat wukong') to the server, which then distributes it to all connected clients. The bottom-right window shows a client ('buzz') sending a message ('secret messages') to the server, which then distributes it to all connected clients. The output of the script shows the server listening for P2P connections on port 127.0.0.1 and the clients sending and receiving messages.

```
sova@LAPTOP-A3B547F5:/mnt/d/Distributed System/Lab $ python3 connect.py
Enter 'server' to start as central server or 'client' to connect as client:
server
Central server started...
sova connected from ('127.0.0.1', 49682)
colt connected from ('127.0.0.1', 29090)
buzz connected from ('127.0.0.1', 42764)
winter connected from ('127.0.0.1', 54474)
```

```
sova@LAPTOP-A3B547F5:/mnt/d/Distributed System/Lab $ python3 connect.py
Enter 'server' to start as central server or 'client' to connect as client: c
client
Enter your username: colt
Enter port for P2P listener (default is 54321): 1111
Enter message (or 'PEER' for list): Listening for P2P connections on 127.0.0.1:1111...
sova: black meat wukong
buzz: black meat wukong
```

```
sova@LAPTOP-A3B547F5:/mnt/d/Distributed System/Lab $ python3 connect.py
Enter 'server' to start as central server or 'client' to connect as client:
client
Enter your username: sova
Enter port for P2P listener (default is 54321): 2222
Enter message (or 'PEER' for list): Listening for P2P connections on 127.0.0.1:2222...
Enter message (or 'PEER' for list): black meat wukong
Enter message (or 'PEER' for list): buzz: black meat wukong
Enter message (or 'PEER' for list): P2P 127.0.0.1 5555 secret messages
Enter message (or 'PEER' for list):
```

```
sova@LAPTOP-A3B547F5:/mnt/d/Distributed System/Lab $ python3 connect.py
Enter 'server' to start as central server or 'client' to connect as client:
client
Enter your username: buzz
Enter port for P2P listener (default is 54321): 5555
Enter message (or 'PEER' for list): Listening for P2P connections on 127.0.0.1:5555...
sova: black meat wukong
Enter message (or 'PEER' for list): black meat wukong
Enter message (or 'PEER' for list): Connected to peer at ('127.0.0.1', 49898)
Peer: secret messages
```

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.