**Computer Network Project Submission**

**A Peer-To-Peer Application**

**Submitted By:**

**Name: Harsh Trada, Roll No: U22CS116**

**Name: Hardik Hemnani, Roll No: U22CS081**

**Name: Aasutosh Baraiya, Roll No: U22CS109**

**Name: Anil Prajapati, Roll No: U22CS136**

**Submitted To: Dr. Keyur Parmar**

**Department of Computer Science and Engineering**

SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY SURAT

2024

**Application Details:**

The provided code implements a simple peer-to-peer (P2P) communication system using User Datagram Protocol (UDP). In a P2P network, each participant (peer) can act both as a client and a server, enabling direct communication between peers without the need for a central server. This decentralized architecture is particularly useful for applications like file sharing, messaging, and real-time collaboration.

**Client-Side Details:**

**Description:**

This Python script serves as the client-side component of a peer-to-peer (P2P) application. The application allows users to communicate with each other over a network using User Datagram Protocol (UDP). It enables users to sign up with a nickname and exchange messages with other users who are also connected to the same network.

**Pseudocode:**

1. **Import necessary libraries:**
   1. Import the socket library for network communication.
   2. Import the threading library for concurrent execution of tasks.
2. **Create a UDP socket:**
   1. Create a UDP socket using socket.socket() function with parameters AF\_INET for IPv4 address family and SOCK\_DGRAM for UDP.
3. **Bind the client to a random port:**
   1. Bind the client to a random port within a specified range using bind() function with the localhost address.
4. **Prompt user for a nickname:**
   1. Use input() function to prompt the user to enter a nickname.
5. **Define a function to receive messages:**
   1. Create a function receive() to continuously receive messages from other users.
   2. Use a try-except block to handle exceptions while receiving messages.
   3. Decode the received message and print it to the console.
6. **Start a thread to receive messages:**
   1. Create a thread t targeting the receive() function.
   2. Start the thread using the start() method.
7. **Send signup request:**
   1. Send a signup request to the server with the user's chosen nickname.
   2. Use sendto() function to send the encoded signup request to the server.
8. **Main message sending loop:**
   1. Start an infinite loop to continuously accept user input and send messages.
   2. If the input message is "!q", exit the loop and terminate the program.
   3. Otherwise, send the message along with the user's nickname to the server using sendto() function.

**Server-Side Details:**

**Description:**

This Python script serves as the server-side component of a P2P communication application. It facilitates communication between multiple clients connected to the network. The server listens for incoming messages from clients and broadcasts them to all connected clients. Additionally, it handles signup requests from new clients and notifies existing clients about new user joinings.

**Pseudocode:**

1. **Import necessary libraries:**
   1. Import the socket library for network communication.
   2. Import the threading library for concurrent execution of tasks.
   3. Import the queue module to handle message queuing.
2. **Initialize variables:**
   1. Create a queue named messages to store incoming messages from clients.
   2. Create an empty list client to store the addresses of connected clients.
3. **Create a UDP socket:**
   1. Create a UDP socket using socket.socket() function with parameters AF\_INET for IPv4 address family and SOCK\_DGRAM for UDP.
   2. Bind the server to the localhost address and port 9999 using bind() function.
4. **Define a function to receive messages:**
   1. Create a function receive() to continuously receive messages from clients.
   2. Use a try-except block to handle exceptions while receiving messages.
   3. Put the received message and client address into the messages queue.
5. **Define a function to broadcast messages:**
   1. Create a function broadcast() to continuously broadcast messages to connected clients.
   2. Iterate over the messages queue until it's empty.
   3. Extract the message and client address from the queue.
   4. Print the decoded message to the console.
   5. If the client address is not in the clients list, add it.
   6. Broadcast the message to all connected clients, including signup notifications.
6. **Create and start threads:**
   1. Create two threads t1 and t2 targeting the receive() and broadcast() functions, respectively.
   2. Start both threads using the start() method.

**Applications:**

1. **Chat Application:**
   1. Users can connect to the P2P network and exchange messages with each other in real-time.
   2. Each user can have a unique nickname to identify themselves in the chat.
   3. Messages are broadcasted to all connected users, allowing group communication.
2. **File Sharing:**
   1. Users can share files directly with each other without relying on a centralized server.
   2. Each user can publish files they want to share with the network.
   3. Other users can search for and request files directly from peers who have them available.
3. **Collaborative Editing:**
   1. Multiple users can collaborate on a document or project in real-time.
   2. Changes made by one user are immediately reflected to all other connected users.
   3. This enables distributed teamwork without the need for a central editing server.
4. **Gaming:**
   1. Users can play multiplayer games with each other over the P2P network.
   2. Game state updates are exchanged directly between players, reducing latency.
   3. This can facilitate peer-to-peer gaming experiences without relying on dedicated game servers.
5. **Voice/Video Calling:**
   1. Users can establish direct audio/video calls with each other.
   2. Communication is encrypted and transmitted directly between peers.
   3. This provides a decentralized alternative to centralized voice/video calling services.

**Conclusion:**

In summary, the peer-to-peer (P2P) application developed demonstrates the efficacy of decentralized networking for real-time communication and collaboration. By leveraging UDP sockets and threading, it enables seamless interaction among users without relying on a central server. Through features like chat, file sharing, and gaming, the application showcases the versatility and potential of P2P architecture in modern networking applications.