**Lab Report: Peer-to-Peer (P2P) Chat Application with Group Chat**

INRODUCTION  
Peer-to-peer (P2P) networking offers a decentralized approach to communication, eliminating the single point of failure and potential bottlenecks of centralized server-based systems. The objective of this lab is to design and implement a decentralized peer-to-peer (P2P) chat application using socket programming. The application allows users to communicate directly with each other without relying on a central server, emphasizing the challenges of decentralized communication, such as peer discovery, message routing, and user presence management. Additionally, the application supports **group chat functionality**, enabling multiple users to participate in a shared conversation.   
  
**Goals:** The primary goals of this lab were to:

* + Develop a functional P2P chat application using Python and tkinter for the GUI.
  + Implement both direct connection (IP/port) and presence-based connection methods.
  + Enable users to exchange text messages and files in both individual and group chat settings.
  + Display user presence information (online/offline status) in the chat interface.

ARCHITECTURE

High Level System (Component Diagram)

A diagram of a diagram

Description automatically generated

**Explanation**:

* Each **Peer** encapsulates:
  + A **GUI** (Chat UI) to handle user interaction.
  + The **P2PChat** component for managing peer connections and message routing.
  + A **File\_Transfer** module for chunk-based file sending/receiving.
  + A **Presence Client** that communicates with the **Presence.**
* Lines (<-->) indicate TCP connections.
* Peers can connect directly to each other **and** optionally to the presence server.

**System Components**

* **ChatUI (tkinter-based):** The main GUI managing user interactions, messages, and system notifications. It connects with P2PChat, FileTransfer, and PresenceClient.
* **P2PChat:** Manages peer-to-peer messaging using TCP sockets, handles group chats, and maintains connections.
* **FileTransfer:** Splits files into chunks, sends them via sockets, and reassembles them on the recipient’s end. Supports individual and group file transfers.
* **PresenceClient:** Connects to a presence server to register users, discover peers, and update online statuses.

Presence-Based Discovery

A diagram of a system

Description automatically generated

**Explanation**:

* The **user interface** (U) calls methods on the PresenceClient (PC) to register or query users.
* The **presence server** (PS) responds with success or error messages, and eventually provides a list of online users.
* Once the **UI** receives that list, the user can **initiate a direct P2P connection** to any desired peer’s IP/port.

File Transfer Flow

A screenshot of a computer screen

AI-generated content may be incorrect.

**Explanation**:

* The **Sender’s UI** triggers the send\_file method in the FileTransfer object (SFT).
* FileTransfer reads the file, chunk by chunk, and instructs P2PChat to send each chunk as a JSON message to the **Receiver’s P2PChat**.
* The receiver’s P2PChat calls back into its own FileTransfer module to handle the chunk (file\_chunk\_callback).
* Chunks are buffered until the last chunk is detected, then the receiver writes the file to disk.
* The UI can display progress or completion messages as needed.

IMPLEMENTATION

**3.1 Communication Protocol**  
Uses Python’s socket library with JSON messages prefixed by a 4-byte length header. Send\_message and receive\_message in utils.py handle framing.

Code:  
A computer screen shot of a program code

Description automatically generated

**3.2 Peer Discovery**

* **Direct Connection:** Users connect via IP and port (join\_network in P2PChat).
* **Presence Server:** Tracks online users via PresenceClient.

**Code:**  
A computer screen shot of a program code

Description automatically generated

**3.3 Message Routing**

* **Direct Messaging:** Sent using recipient’s IP and port.
* **Group Chat:** Broadcasts messages via broadcast\_group\_message.

**3.4 User Presence**  
Peers send heartbeats; those inactive for 15 seconds are marked offline (\_send\_heartbeat).

A screen shot of a computer program

Description automatically generated

**3.5 Group Chat**  
Supports group creation/joining with unique group\_id. Messages are broadcast to all members.  
Code :  
A screen shot of a computer program

Description automatically generated

**3.6 Concurrency**  
Uses threading for concurrent connections, heartbeat management, and thread-safe resource access.

**3.7 User Interface**

The application features a graphical user interface (GUI) built using the tkinter library. The GUI allows users to: Enter their username. Create or join a chat network. Send and receive messages. Create or join group chats. View the list of connected peers and their status. Send files to other peers.

TESTING

**4.1 Test Cases**

* **Peer Discovery: Tested direct IP connection and presence server-based discovery.A screenshot of a chat

  Description automatically generatedA screenshot of a computer

  Description automatically generatedA screenshot of a computer

  Description automatically generated A screenshot of a computer

  Description automatically generatedScreens screenshot of a chat

  Description automatically generatedA screenshot of a phone

  Description automatically generated**
* **Message Routing: Verified message delivery, including handling offline peers.**

**Screens screenshot of a chat

Description automatically generated**

* **Group Chat: Ensured group messages reach all members.**
* **File Transfer: Verified correct file transmission and saving.**

**Screens screenshot of a chat

Description automatically generatedA screenshot of a computer

Description automatically generated**

* **User Presence: Confirmed online/offline status updates via heartbeats.**

**A screenshot of a computer

Description automatically generated**

* **Concurrency: Ensured stable performance with multiple simultaneous connections.**

**4.2 Test Results**  
All tests passed successfully, confirming proper functionality in peer discovery, messaging, group chat, user presence, file transfer, and concurrency handling.

DISCUSSION  
**Challenges Faced:  
The development of the P2P chat application presented several challenges, particularly in the areas of file transfer, UI implementation, group chat across different machines, peer discovery, and concurrency.**

**Challenges and Solutions**

|  |  |
| --- | --- |
| **Challenge** | **Solution** |
| **File Transfer** | Split files into chunks, encode in Base64, and reassemble on the receiver. |
| **UI Implementation** | Used Tkinter with threading to ensure responsiveness and real-time updates. |
| Group Chat Across Machines | Broadcast messages to all group members using a unique group\_id. |
| **Peer Discovery** | Supported direct connection and presence-based discovery using a server. |
| **Concurrency** | Used threading and locks to manage shared resources and prevent race conditions. |

CONCLUSION  
This project provided helpful practice in designing and creating a decentralized P2P chat application with group chat functionality. The application is successfully able to utilize peer discovery, message routing, user presence, group chat, and file transfer in a decentralized environment. Though the current implementation is okay for small networks, future efforts can be towards enhancing scalability and making advanced functionality such as DHT-based peer discovery and encryption available.

APPENDIX (Code)  
  
The well-commented source code for the project is included in the following files:

* chat.py: Main GUI and application logic.
* p2p\_chat.py: Core P2P chat functionality, including group chat.
* message\_handler.py: Handles incoming messages.
* file\_transfer.py: Handles file transfer between peers.
* utils.py: Utility functions for message framing and socket communication.

<https://github.com/keshavghimire/p2pchat>  
  
**AI Usage in Coding**

**Prompt Used for AI Assistance:**

1. python or java for basic p2p socket programming chat system
2. What will be p2p chat flow diagram look like?
3. How can I handle multiple concurrent connections in a Python P2P chat application using threading?
4. Is my code implementing this?   
   To ensure that when one peer sends a message, it gets displayed to all connected peers (i.e., a true group chat behavior), you need to make sure that the message is broadcast to every peer in the network, not just the sender or direct connections.
5. modify my code to align messages differently based on whether they're from me or others.
6. add border radius in this code join\_button = tk.Button(root, text="Join Chat", font=("Helvetica", 14), bg="#4CAF50", fg="white", relief="flat", command=on\_join\_chat) join\_button.pack(pady=10, ipady=10, ipadx=20) # Add padding and rounded edges effect
7. Can we use this as a group cat and run it on different devices?
8. How can I implement file transfer functionality in a P2P chat application using Python?
9. Write a Python function that reads a file, splits it into chunks of 4096 bytes each, and encodes each chunk using base64.