CMSC 626: P2P File System

Arya Honraopatil Sai Krishna Vishnumolakala Venkat Pantham Abhigna Ramineni
Lakshman Teja Gudivada

I. Introduction

Our P2P file system represents a decentralized network where each participant (peer) serves both as a client and a server. This architecture has direct sharing and communication between peers, which ensures efficiency and scalability of the system. The system is built with Python and utilizes PyQt5 for the graphical user interface.

II. System Architecture

Central Server

The central server acts as a registry for peers to register and discover each other. It maintains a list of active peers but does not involve itself in file transfers.

Peer

Each peer is capable of performing server-like (e.g., hosting files) and client-like (e.g., requesting files) operations. Peers register with the central server and use direct peer-to-peer connections for file transfers and messaging.

Graphical User Interface (GUI)

It is developed using PyQt5, the GUI provides a user-friendly platform for users to engage with the P2P network, manage files, send messages, and view active peers and system logs.

III. Implementation

The central server acts as a registry for active peers. On startup, it listens on a specified port for incoming connections. When a peer connects, it registers the peer's details (IP, port, name) and shares the list of currently active peers with it. For handling connections, it uses Python's socket and threading libraries to handle multiple simultaneous peer registrations.

Each peer acts as both client and server. On launch, it connects to the central server to register itself and fetch the list of active peers. While sharing files, it can initiate or accept file transfer requests. It handles file data transfer through direct socket connections with other peers. It sends and receives text messages directly to/from other peers, bypassing the central server.

For concurrency control it manages multiple operations (like listening for incoming connections and sending data) concurrently using threads. The GUI displays active peers, allows file transfer initiation, and supports sending/receiving messages. It updates the peer list and messages dynamically, reflecting the current network state. Each peer logs its activity to a unique file, aiding in monitoring and debugging. Logs include timestamps, operations performed, messages sent/received, and any errors or important events.

Requirements:
You will need Python and PyQt5 for running this file system. For PyQt5 you can install via pip install pyqt5.

Procedure
1. Start the Central Server:
   a. Navigate to the directory containing the server script (central_server.py).
   b. Run the script: python central_server.py.
   c. The server will start and listen for incoming peer registrations.
2. Launch a Peer Instance:
   a. Navigate to the directory with the peer script (peer.py and peer_gui.py).
   b. Run python peer_gui.py.
   c. Enter a unique name for the peer when prompted.
   d. The GUI will launch, displaying available options like active peers, file transfer, and messaging.

Repeat the process for launching a peer instance for each additional peer you want in the network and ensure each peer has a unique name for identification.

IV. Key Functionalities and Advantages

1. End-to-End File Sharing

- Direct File Transfers: Peers can send and receive files directly with each other, bypassing the need for a centralized server. This approach reduces bottlenecks and potential points of failure.
- GUI-Based Interaction: Users can easily choose files to send and specify destinations for incoming files through an intuitive interface.
- Dynamic Participation: New peers can join the network and immediately start sharing files, enhancing the system's flexibility and responsiveness to changing network conditions.

2. Real-Time Messaging System

- Instant Communication: The system supports real-time messaging, allowing peers to communicate instantly. This feature is crucial for coordinating file transfers and collaborative activities.

- Seamless Integration: Messaging is seamlessly integrated into the peer interface, providing a holistic user experience that combines file sharing and communication.

3. Efficient Peer Discovery

- Central Server for Initial Connection: A central server facilitates the initial discovery of peers. Once connected, peers communicate directly.
- Scalable Network Topology: The system easily scales as the number of peers increases, maintaining efficiency in peer discovery and overall network performance.

4. Robust Logging Mechanism

- Detailed Activity Logs: Each peer session is logged in detail, including actions taken, files transferred, messages sent, and any errors encountered.
- Enhanced Troubleshooting: The logs provide valuable insights for troubleshooting and understanding peer behavior, aiding in system maintenance and improvement.

5. Concurrent Read/Write Operations

- Simultaneous Operations: The system is designed to handle concurrent read and write operations, ensuring that multiple peers can interact with the system without significant delays or conflicts.

V. Future Enhancements

We have effectively demonstrated the core principles of a P2P network, but future enhancements could include:

1. Advanced File Versioning: To ensure users always access the latest version of a file.
2. Encryption: For secure data storage and communication.
3. User Authentication and Permissions: To provide more granular control over file access and modifications.

VI. Conclusion

This P2P file system implements decentralized file sharing and communication. It has the advantages of P2P networks, such as scalability, direct peer interactions, and reduced reliance on centralized components.

VII. Pseudocode

Pseudocode outline for the key functions of your Peer-to-Peer (P2P) file system, covering the central server, peers, and the GUI components.

### Central Server

```
Function StartCentralServer():
    Initialize server_socket to listen on specified IP and port
    While True:
        Wait for a peer to connect
        When a peer connects, start a new thread:
            Function HandlePeerConnection(peer_socket):
                While True:
                    message = Receive message from peer_socket
                    If message type is "register":
                        Add peer to active_peers list
                    ElseIf message type is "request_list":
                        Send active_peers list to peer
                    ElseIf peer disconnects:
                        Remove peer from active_peers
                        Break loop
```

### **Peer**

```
Class Peer:
    Function __init__(name, server_host, server_port):
        Set peer name, server address
        Create peer_socket to listen for incoming connections
        Connect to central server and register
        Start thread for accepting incoming connections

    Function RegisterWithServer():
        Send registration message to central server

    Function FetchActivePeers():
        Request list of active peers from central server

    Function AcceptIncomingConnections():
        While True:
            Accept connection
            Start a new thread to handle the connection

    Function HandleIncomingConnection(client_socket):
        Determine message type (file transfer request, message, etc.)
        Process message accordingly

    Function SendFile(target_peer, file_path):
        Connect to target_peer
        Send file transfer request
        If accepted, send file data
```

```
 Function SendMessage(target_peer, message):
     Connect to target_peer
     Send message
```

### GUI (Graphical User Interface)

```
Class PeerGUI:
    Function __init__(peer):
        Create main window
        Add buttons for refreshing peers, sending files, and messaging
        Attach functions to buttons

    Function RefreshPeers():
        Display updated list of active peers

    Function SelectFileToSend():
        Open file dialog to select a file
        Call peer's SendFile function

    Function SendMessage():
        Get selected peer and message text
        Call peer's SendMessage function
```

VIII. References:

[1] https://www.geeksforgeeks.org/p2p-peer-to-peer-file-sharing/
[2] https://www.tutorialspoint.com/securing-communication-channels-with-diffie-hellman-algorithm-an-implementation-guide
[3] https://cseweb.ucsd.edu/classes/sp16/cse291-e/applications/ln/lecture13.html