Software and Data Engineering
Assignment 1
M22MA003

Problem Statement
Design and implement a Peer-to-Peer (P2P) system. Create a P2P system that allows users to share files directly between their devices without the need for a centralized server. This objective of the assignment is to understand the fundamental concepts of distributed systems, networking, and data management.

Approach 1:USING SINGLE PYTHON CODE FILE.

1. System Design :

1.a The architecture of the implemented P2P system + 1.b How peers will connect, communicate, and share files:-

Peer Class:
This class represents a peer in the P2P network.
It has attributes to store the host, port, socket, list of connections, a flag for the initial message sent, and a directory for storing received files.
The class provides methods for connecting to other peers, listening for incoming connections, sending data, sending files, receiving files, and starting the peer.

Initialization:
Two instances of the Peer class, **peer1 and peer2**, are created to represent two peers.
Each peer is configured with a host (IP address) and port number.

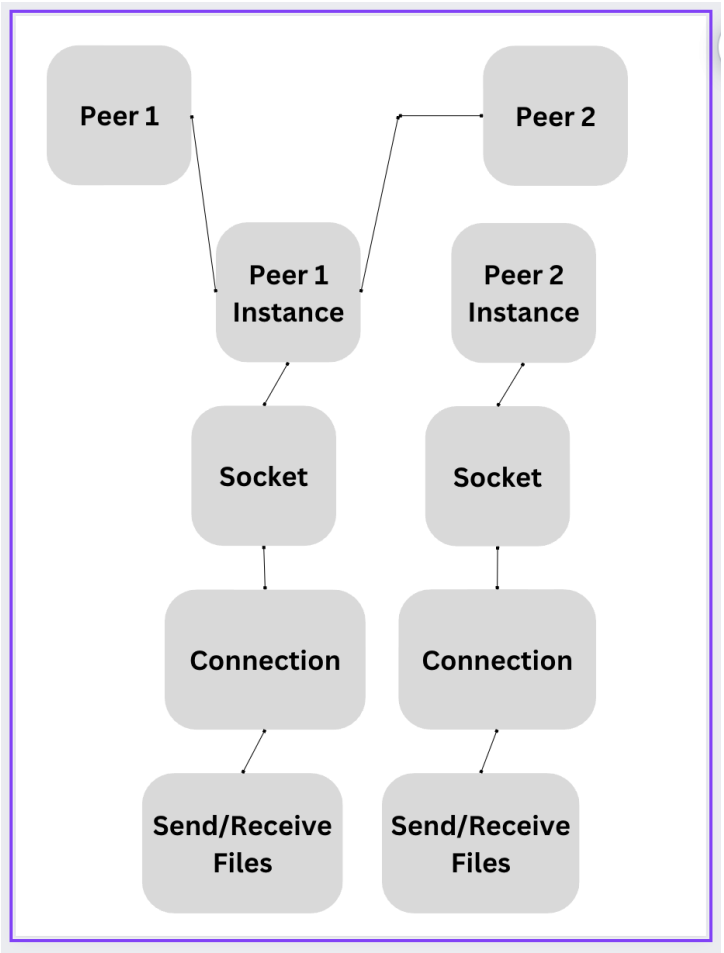
Start Method:
The start method of the Peer class is called for both peer1 and peer2. This method starts a separate thread for listening to incoming connections.

Connect:
The connect method allows a peer to connect to another peer. It creates a socket connection to the specified peer's host and port.
After establishing a connection, the peer sends an initial message ("Hello from peer!") and, in this case, sends a sample file named "sample.txt" to the connected peer.

Listen:
The listen method runs in a loop on the specified host and port, waiting for incoming connections.

When a **connection is accepted**, it is added to the list of connections, and the peer can receive files from the connected peer.

Send Data and Send File:
The send_data method sends data (text messages) to all connected peers.
The send_file method sends a file to all connected peers. It reads the file in chunks and sends them over the connections. An "EOF" marker indicates the end of the file.



Architecture Diagram

Receive File:
The receive_file method receives a file from a connected peer. It reads the file data in chunks and stores it in the "received_files" directory. The end of the file is marked by receiving "EOF."

Main Section:
In the main section, two Peer instances (peer1 and peer2) are created.
Separate threads are started for listening to incoming connections for each peer.
Both peer1 and peer2 are connected to each other, establishing a P2P connection.

Overall Architecture:
The architecture involves two peers, peer1 and peer2, communicating over a P2P network. Each peer can listen for incoming connections and send data or files to connected peers. The code demonstrates basic P2P communication with file sharing capabilities.

1.c How this architecture handles these challenges and potential improvements:

Peer Discovery:
Current Approach: The code doesn't implement any form of automatic peer discovery. Instead, it requires peers to know each other's IP addresses and ports in advance and connect directly.
Improvements: For a more robust system, one might consider implementing a peer discovery mechanism, such as a distributed hash table (DHT) or a centralized tracker. This would enable peers to discover and connect to each other dynamically without prior knowledge of their addresses.

Security:
Current Approach: The code lacks authentication and authorization mechanisms. Any peer with knowledge of the IP and port can connect and exchange data.
Improvements:
To enhance security:-

1. Implement user authentication: Peers should authenticate themselves to ensure that only authorized users can join the network.
2. Implement data encryption: Encrypt data transmitted between peers to protect it from eavesdropping.
3. Use digital signatures: Sign messages to verify their authenticity and integrity.
4. Implement access control: Define who can access shared files and resources.

File Sharing Security:
Current Approach: The code allows peers to send and receive files without verifying their content. This poses a security risk, as malicious files could be sent.
Improvements:
To address file-sharing security:

1. Implement file validation: Check file integrity and authenticity using checksums or digital signatures.
2. Implement file scanning: Scan files for malware and viruses before accepting them.
3. Define file access controls: Specify who can access and download shared files.

Firewall:
Current Approach: The code doesn't handle Network Address Translation (NAT) traversal or firewall issues, which can prevent direct peer-to-peer connections.
Improvements: Consider techniques like TURN to enable peers to establish connections even when behind NAT or firewalls.

Reliability and Error Handling:
Current Approach: The code has minimal error handling and doesn't provide mechanisms for reliability or message acknowledgments.
Improvements: Implement message acknowledgments, retransmission mechanisms, and error recovery strategies to ensure reliable data transmission.

Scalability:
Current Approach: The code doesn't address how the system will scale as the number of peers increases.
Improvements: Consider load balancing, peer grouping, and distributed architecture to handle a larger number of peers efficiently.

User Experience:
Current Approach: The code lacks a user-friendly interface or feedback mechanisms for users.
Improvements: Implement a user-friendly GUI or CLI interface that provides feedback, progress indicators, and user instructions.

Resource Management:
Current Approach: The code doesn't manage resources efficiently, which could lead to resource exhaustion.
Improvements: Implement resource management to limit the number of concurrent connections, control bandwidth usage, and handle resource-intensive operations gracefully.

2. Implementation:

To implement a working P2P system with the given code as a starting point, one would need to:

Implement a user interface (UI) for interacting with peers and sharing files. This UI can be developed using a GUI library like Tkinter for desktop applications or a web-based interface using a framework like Flask or Django for a more comprehensive system.

Add functionality for peers to join the network. This may require a mechanism for peer discovery, such as using a central server for initial peer bootstrap or implementing a Distributed Hash Table (DHT) for decentralized peer discovery.

Enhance file sharing capabilities to allow users to select and share files other than just "sample.txt." one can achieve this by extending the UI to include file selection and implementing the necessary logic to send and receive various file types.

Ensure robust error handling for different scenarios, including network failures, file not found errors, and more. Enhance the code to handle exceptions gracefully and provide meaningful error messages to users.

3. Efficiency and Scalability :
Efficiency and scalability are critical aspects of a P2P system. To address these concerns:

Implement a more efficient file discovery mechanism. The current code shares files without any index or search functionality. one could implement a system-wide index of shared files or use a DHT to enable efficient file lookup.

Consider implementing techniques such as Distributed Hash Tables (DHTs) or Distributed Ledger Technologies (DLTs) for more efficient and scalable peer discovery and file sharing.

Discuss strategies for load balancing and data replication to ensure that the system can handle a growing number of peers without performance degradation.

4. User Interface:
Creating a user-friendly interface is essential for user interaction. one can achieve this by:

Developing a GUI application that allows users to search for files shared by other peers, initiate file transfers, and manage their own shared files.

Implementing features like progress bars during file transfers, clear error messages, and a user-friendly laonet.

Providing feedback on the status of connections and file transfers, so users have a clear understanding of what is happening.

5. Follow-up Question:
Advantages of a decentralized P2P architecture compared to a hybrid architecture with some centralization:

Advantages of Decentralized P2P Architecture:

Resilience: Decentralized P2P systems are more robust against failures. There is no single point of failure, making them highly resilient.

Scalability: Decentralized architectures can scale more easily as new peers can join without affecting existing infrastructure significantly.

Privacy: Users have more control over their data and interactions in a decentralized system, enhancing privacy.

Challenges of Decentralized P2P Architecture:

Complexity: Building and maintaining a fully decentralized system can be more complex, requiring advanced networking and consensus algorithms.

Initial Bootstrap: Decentralized systems may require initial centralization or bootstrapping mechanisms for peers to discover each other.

Performance: Ensuring high performance in a fully decentralized system can be challenging, especially for tasks like efficient file searching.

Advantages of a Hybrid Architecture:
Efficiency: Hybrid architectures can combine the resilience of P2P with the efficiency of centralized systems for tasks like indexing and search.

Security: Centralized components can provide additional security and authentication measures.

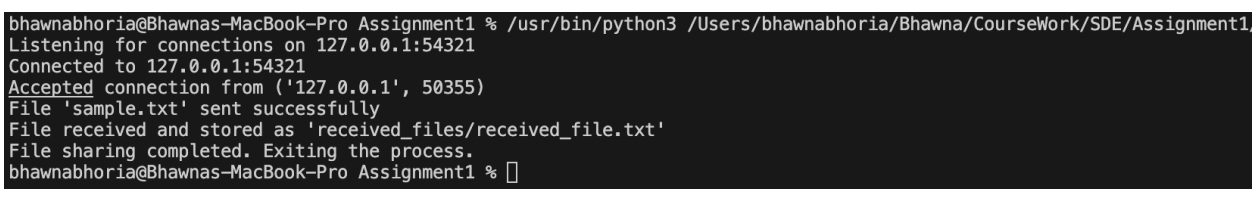
Challenges of a Hybrid Architecture:
Centralization Risks: The central components in a hybrid architecture can become single points of failure and security vulnerabilities.

Scalability Limits: The scalability of hybrid architectures may be limited by the capacity of central components.

Ultimately, the choice between a fully decentralized P2P architecture and a hybrid one depends on the specific use case and the trade-offs one are willing to make between resilience, complexity, and performance.

OUTPUT:-

Listening for connections on 127.0.0.1:54321
Connected to 127.0.0.1:54321
Accepted connection from ('127.0.0.1', 50355)
File 'sample.txt' sent successfully
File received and stored as 'received_files/received_file.txt'
File sharing completed. Exiting the process.



Approach 2:-

1.a Architecture and how peers communicate:

The P2P system is designed as a decentralized network of peers where each peer can act both as a client and a server. The architecture consists of the following components:

Peers: These are the individual participants in the P2P network. Each peer has the capability to connect to other peers, share files, and receive files.

Server Thread: Each peer runs a server thread to listen for incoming connections from other peers. This server thread handles the initial setup of connections and file sharing.

Client Thread: Peers can also run client threads to connect to other peers. These threads handle the initiation of connections to other peers.

Chat Application: The user interacts with the P2P system through a chat application. This application provides a graphical user interface (GUI) for sending and receiving messages, sharing files, and issuing commands.

1.b Connection and Communication:

Peer Discovery: Peers can connect to each other using their IP addresses and ports. The initial peer-to-peer connection is established when a client thread connects to a server thread on another peer. This connection allows peers to exchange initial information such as nicknames, IP addresses, and ports.

Messaging: Peers can exchange messages with each other. Messages can be simple text messages or commands that control the behavior of the P2P system.

File Sharing: Peers can share files by sending them to connected peers. Files are transferred in chunks, and the receiving peer reconstructs the file from these chunks.

1.c Handling Challenges:

Peer Discovery: Peer discovery is done manually by specifying the IP address and port of the peer one wants to connect to. This approach simplifies the system but lacks automatic discovery. Implementing a distributed hash table (DHT) or tracker-based system could enhance peer discovery.

Security: The current design lacks security measures such as encryption. To enhance security, one can implement end-to-end encryption for messages and files exchanged between peers.

2. Implementation:

The implementation includes the ability for peers to join the network, connect to other peers, share files, and communicate through the chat application.

Appropriate error handling is implemented to deal with issues such as failed connections or file transfers.

User-friendly interactions are provided through the chat application's GUI, allowing users to initiate file sharing, send messages, and issue commands.

3. Efficiency and Scalability:

The system's efficiency is good for a small number of peers due to the simplicity of direct peer-to-peer connections.

However, as the number of peers increases, peer discovery may become challenging, and maintaining direct connections to many peers could lead to scalability issues. To address this, implementing a DHT or a tracker-based system for peer discovery would be more efficient.

To ensure efficient file discovery and download, a distributed index or metadata system can be implemented. This system would allow peers to search for files and retrieve them efficiently without the need for direct connections to all peers in the network.

4.User Interface:

The chat application provides a user-friendly interface for interacting with the P2P system.

Users can easily send messages, share files, and issue commands through the GUI.

The chat application's interface simplifies the process of initiating file transfers and monitoring chat messages.

5. Advantages and challenges of a decentralized P2P architecture compared to a hybrid architecture that involves some centralization.

Advantages of a Decentralized P2P Architecture:
Resilience: Decentralized P2P networks are more resilient to failures and attacks. There's no single point of failure, making it difficult for malicious actors to disrupt the network.

Scalability: Decentralized networks can scale organically as more peers join, without relying on centralized servers that may become bottlenecks.

Privacy: Decentralization can enhance user privacy since there's no central authority collecting user data.

Challenges of Decentralized P2P Architecture:
Peer Discovery: It can be challenging to discover other peers in the network, especially without a central tracker. Implementing efficient peer discovery mechanisms is crucial.

Security: Ensuring the security and privacy of communication in a decentralized network can be complex, as there's no central authority to enforce security measures. Peer-to-peer encryption and authentication are essential.

Efficiency: Maintaining direct connections to a large number of peers can be inefficient. Implementing distributed indexing and routing mechanisms is necessary for efficient file sharing and search.

Advantages of a Hybrid Architecture:
Centralized Management: A hybrid architecture can provide centralized management for tasks like peer discovery, indexing, and authentication, improving efficiency and security.

Reliability: Centralized components can offer higher reliability and availability, especially in cases of network congestion or high demand.

Scalability: The centralization of certain functions can simplify the management of large-scale networks.

Challenges of a Hybrid Architecture:
Single Point of Failure: Hybrid architectures introduce single points of failure in the centralized components. If the central server or authority goes down, the entire network can be affected.

Privacy Concerns: Centralized components may collect and store user data, raising privacy concerns and potential data breaches.

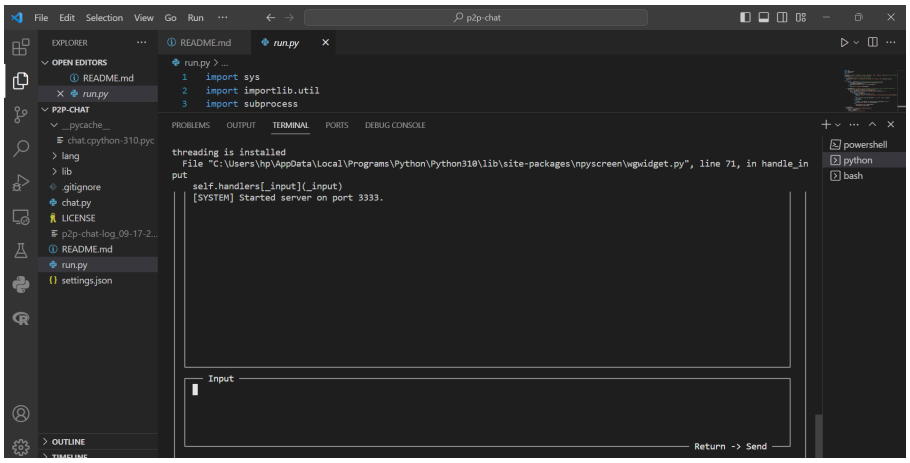
Dependency: Hybrid architectures depend on the availability and reliability of central components, which can introduce vulnerabilities and dependencies on third-party services.

Ultimately, the choice between a decentralized P2P architecture and a hybrid architecture depends on the specific use case and the trade-offs between decentralization, efficiency, and centralized management.

OUTPUT:-

```
threading is installed
  File "C:\Users\hp\AppData\Local\Programs\Python\Python310\lib\site-packages\npyscreen\wgwidget.py", line 71, in ha
put
    self.handlers[_input](_input)
    [SYSTEM] Here's a list of available commands:
    [SYSTEM] /connect [host] [port] | Connect to a peer
    [SYSTEM] /disconnect | Disconnect from the current chat
    [SYSTEM] /nickname [nickname] | Set your nickname
    [SYSTEM] /quit | Quit the app
    [SYSTEM] /port [port] | Restart server on specified port
    [SYSTEM] /connectback | Connect to the client that is connected to your server
    [SYSTEM] /clear | Clear the chat. Logs will not be deleted
    [SYSTEM] /eval [code] | Execute python code
    [SYSTEM] /status | Returns the clients status
    [SYSTEM] /log | Logs all messages of the current session to a file
    [SYSTEM] /help | Shows this help
    [SYSTEM] /lang [language] | Changes language to specified two digit country code


  Input
```



References:-

1. https://medium.com/@luishrsoares/implementing-peer-to-peer-data-exchange-in-python-8e69513489af
2. https://github.com/F1xw/p2p-chat/tree/master