Project Report:

Peer-to-Peer Communication Application

Introduction

The topic of this project is a Peer-to-Peer (P2P) Communication System, a distributed network architecture where nodes, referred to as "peers," communicate directly with each other without requiring a centralized server.
This project explores a Python-based P2P communication system that supports both single and directory-level file transfers over TCP, which is capable of peer discovery, message broadcasting, and direct communication. This type of system finds relevance in modern networking scenarios like blockchain-based applications, distributed databases, and real-time collaborative tools, emphasizing its importance in cybersecurity, distributed computing, and network engineering fields.

Overview

This Peer-to-Peer (P2P) communication application facilitates efficient local network communication, offering multiple features such as:

1. Broadcasting for peer discovery.
2. Multiple peers at a time.
3. Messaging (broadcast and private).
4. File sharing (individual and group).
5. Directory sharing with specific peers.
6. Manual peer addition by specifying IP and port.
7. Dynamic GUI for easy interaction.

The application utilizes **Python** for its implementation and leverages the **socket** module for networking and **Tkinter** for GUI creation.

Features and Functionalities

1. Peer Server

The server enables incoming peer connections and maintains a list of active peers:

- Uses a TCP socket server to accept connections.
- Stores peers as tuples of (IP, Port).
- Spawns a thread for each peer connection to handle communication independently.

2. Peer Discovery

Broadcasting and listening mechanisms allow dynamic peer discovery:

- **Broadcast Sender**: Sends the host's address (IP:Port) periodically over the network using UDP on a predefined port (9090).
- **Broadcast Listener**: Listens for incoming broadcast messages and adds new peers dynamically.

3. Messaging

- **Broadcast Messaging**: Sends messages to all connected peers using their respective IP and Port.
- **Private Messaging**: Allows targeting a specific peer for personalized communication.

4. File Sharing

- **Broadcast File Sharing**: Sends a file to all connected peers sequentially.
- Uses TCP sockets for reliable file transmission.

5. Directory Sharing

- Transfers an entire directory to a specific peer.
- Utilizes relative paths to preserve directory structure.

6. Manual Peer Addition

Users can manually add peers by specifying their IP and port using the GUI.

Tools and Techniques:

- **Programming Language:**

Python: Chosen for its simplicity, extensive libraries, and strong support for socket programming.

- **Libraries and Modules:**

Socket: Utilized for implementing TCP and UDP communication between peers.

Threading: Used to handle multiple connections concurrently and ensure seamless operations.

- **Networking Techniques:**

Broadcast Messaging: Essential for dynamic peer discovery in a decentralized setup.

TCP Communication: Ensured reliable and ordered message delivery between peers.

- **Development Framework:**

A modular approach to coding was adopted to simplify integration and debugging.

- **Tqdm Library:**

Enhances user experience by displaying real-time progress bars for file transmission.

- **OS Library:**

Facilitates file and directory handling, ensuring compatibility across platforms.

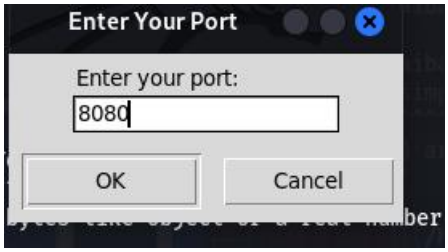
Technical Implementation

1. First step was to save our python scipt in a file I.e: “g2.py”

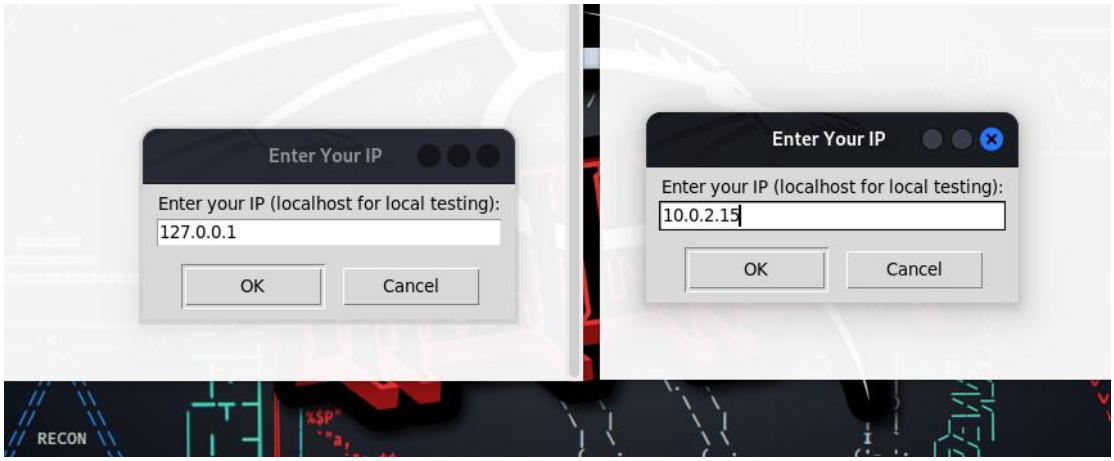
2. Then we excecuted that file on 2 different terminals, and this window popped up then shows a message saying we need to enter our local host ip for communication.

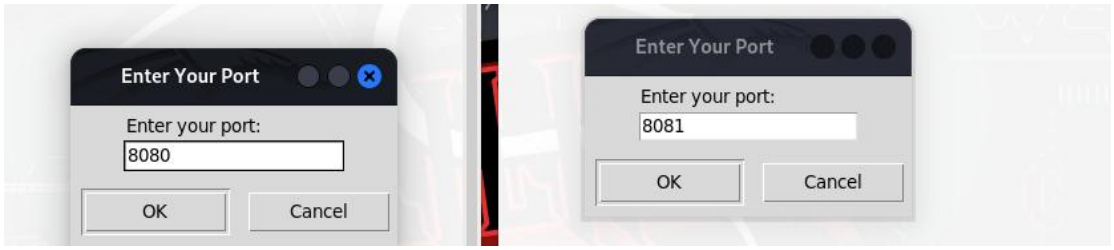
3. Our local ip was entered and pressed ok.

4. Then it asked for port number which was also entered.



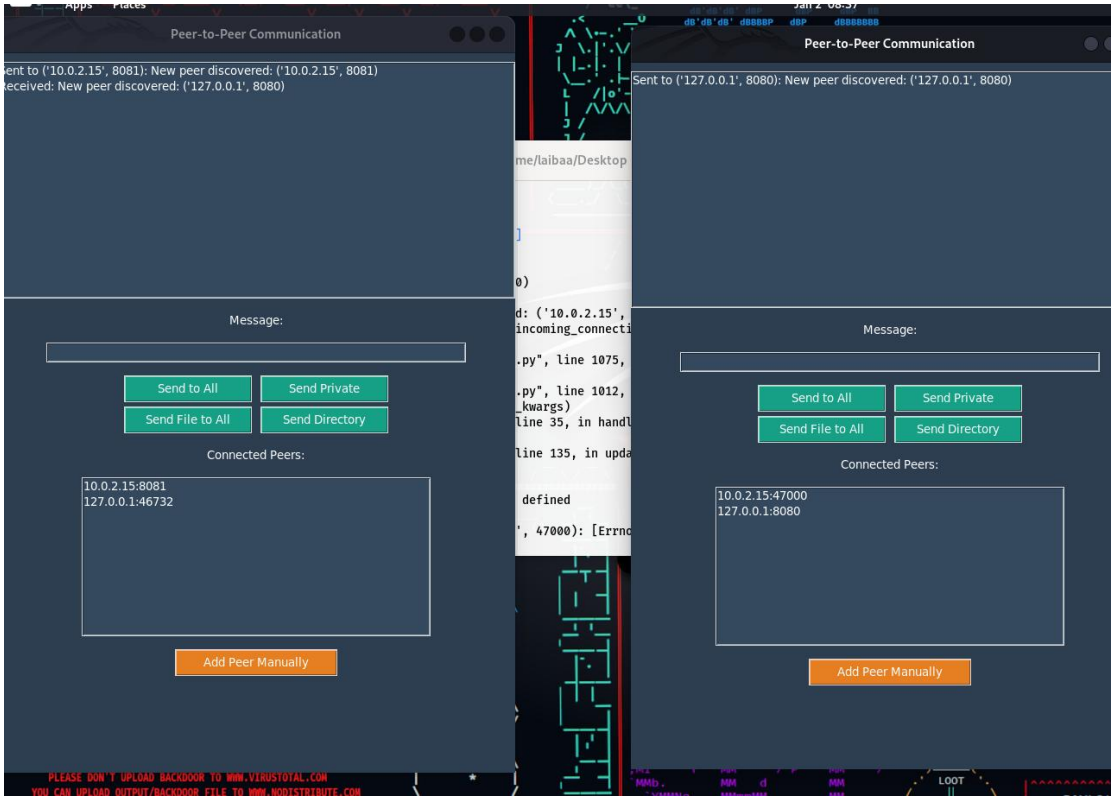
For other peer:



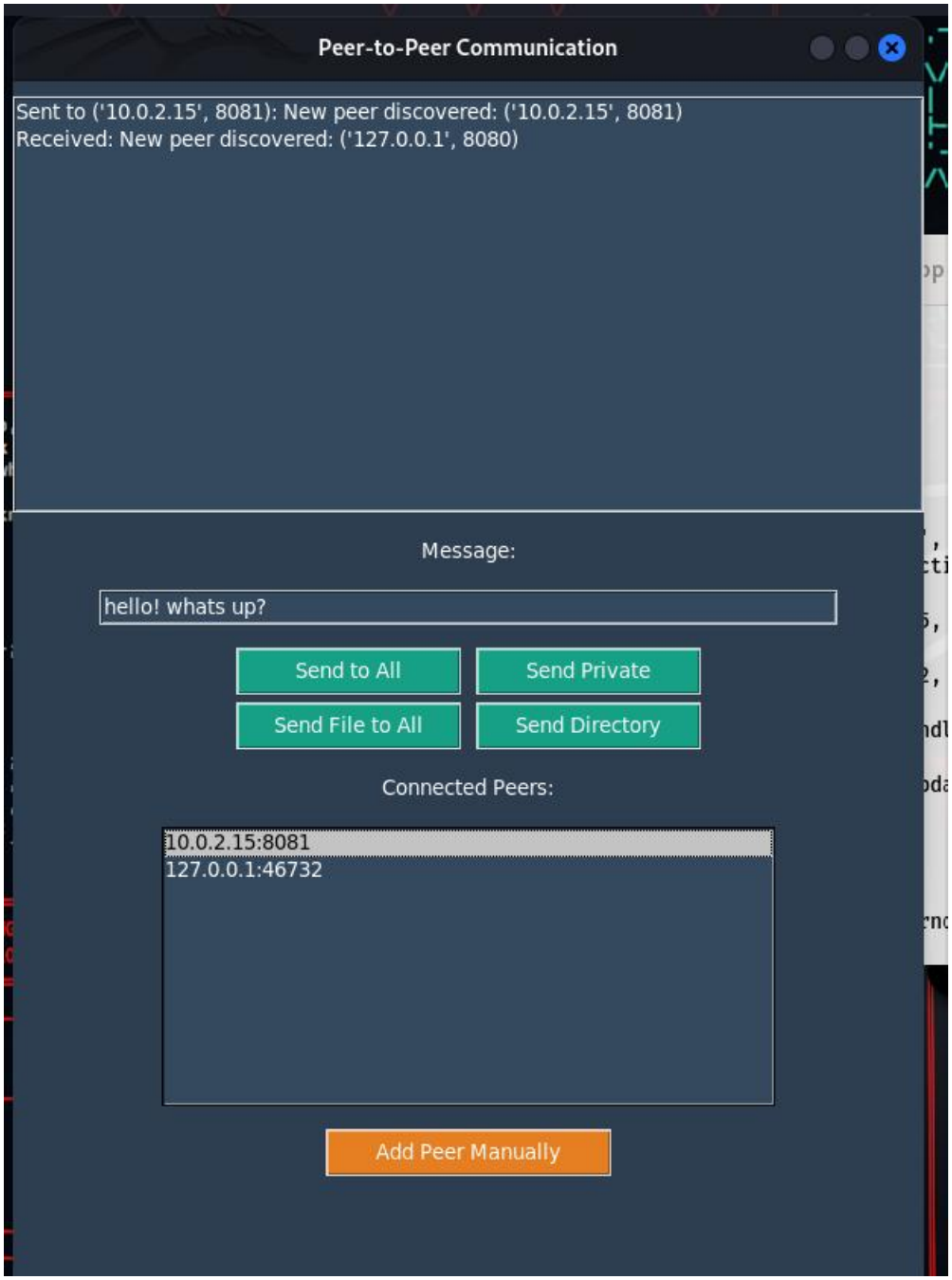


5. This is the GUI for project that will be visible after entering ip and port number

- We can see that automatically a connection formed between the windows and different ports, and automatically all the peers are connected.

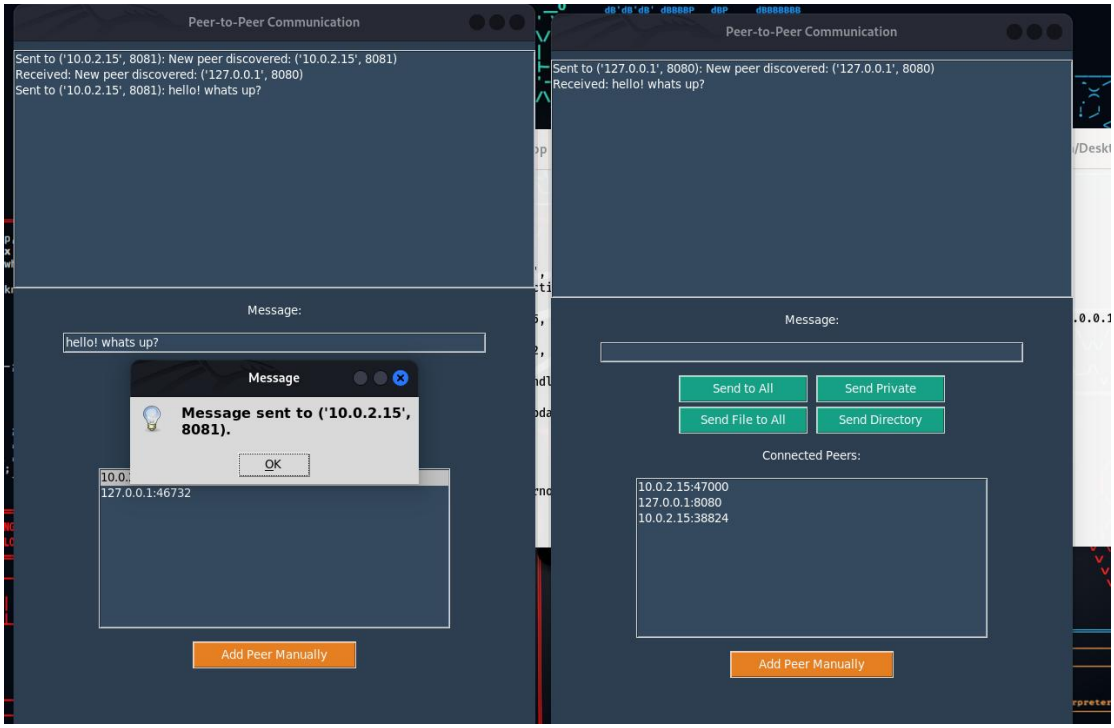


6. Now we will send messages to the other peer (10.0.2.15)



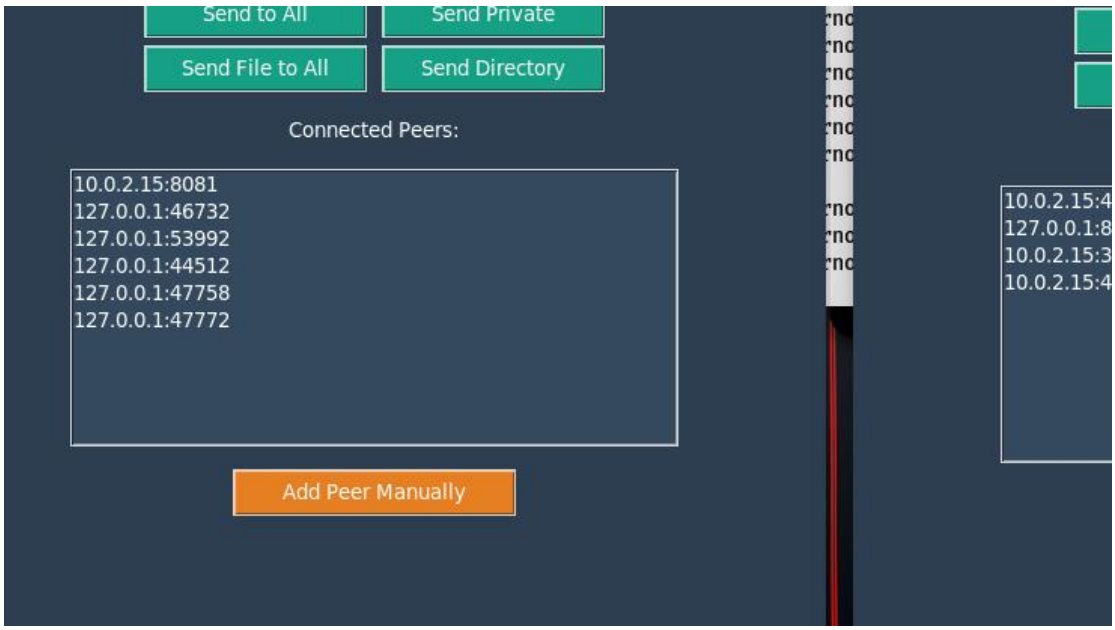
7. We will type our message in the message box down below and select the peer who we want to send the message to and press ok.

- When the message is sent successfully this window will pop up.

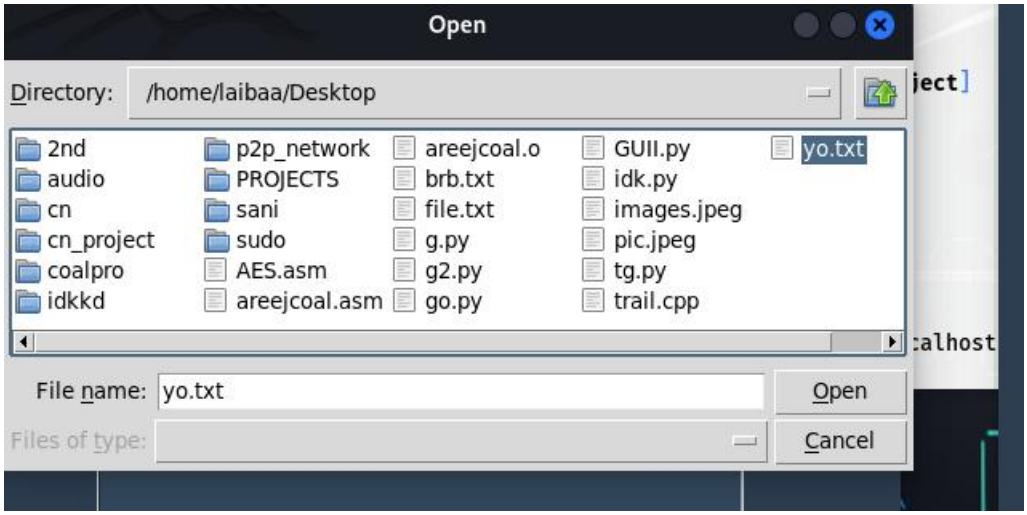


8. On the other peer's window we can see that we received the message.

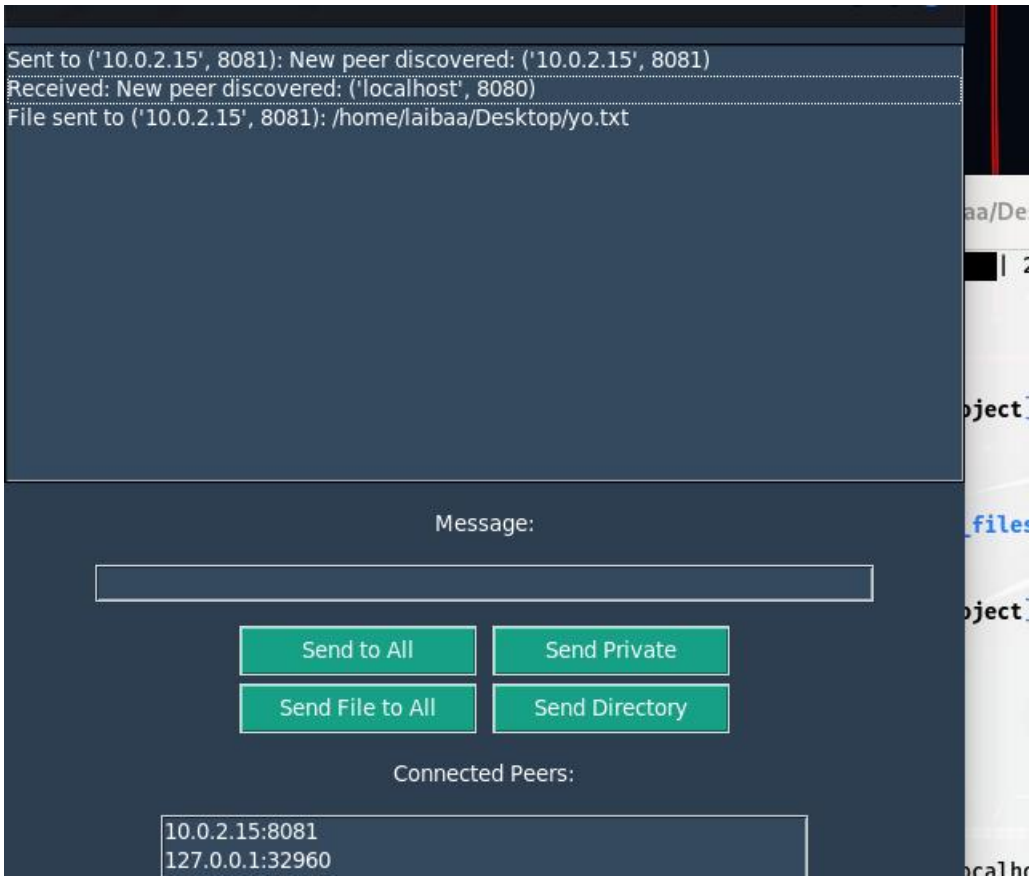
- We can also see the list of connceted peers in the window at the bottom.



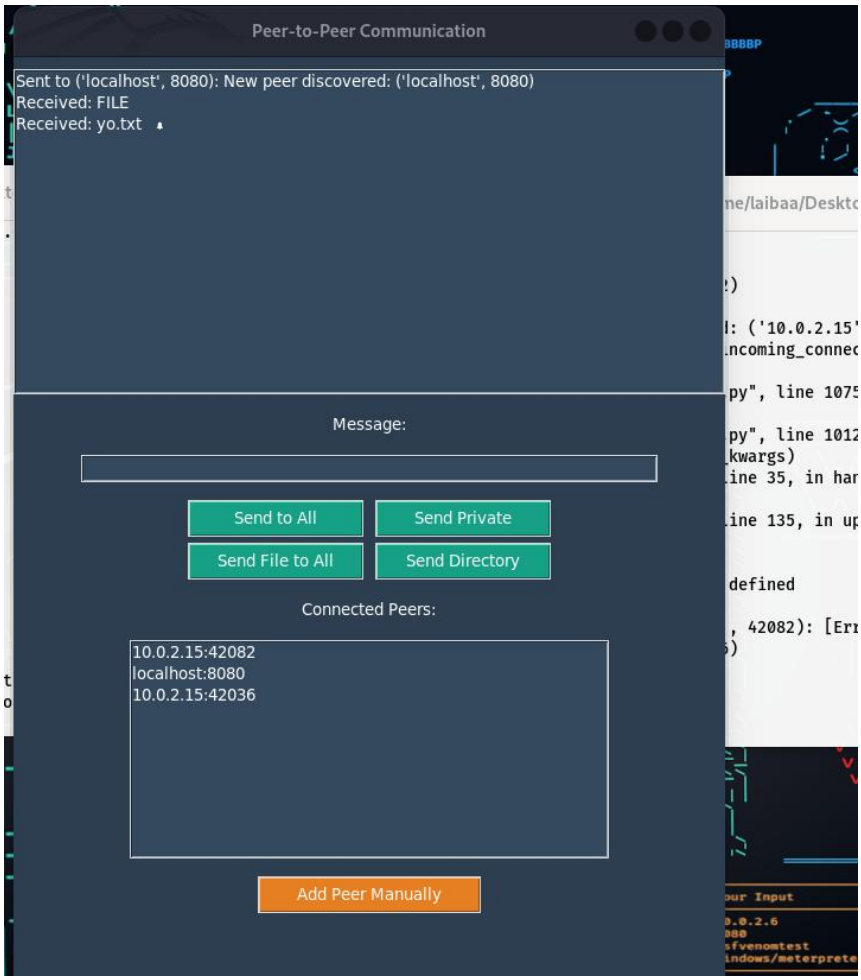
9. Now we will try sending a file to the selected peer connected. Click select ‘send file’ option and this window opens up, select the file we want to share and click ok.



10. In the window we can see the pop up message that says file sent to peer.



11. In the other peer window we can see the file received.

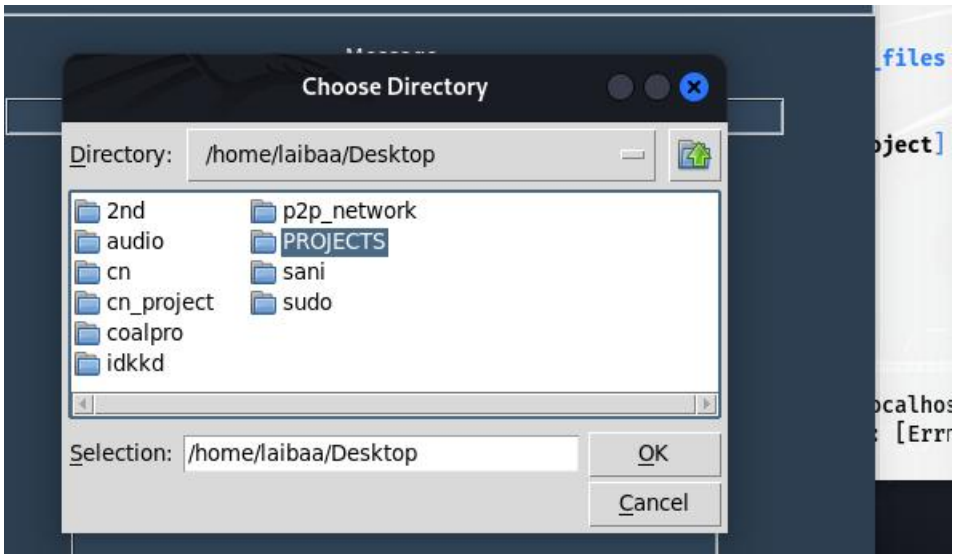


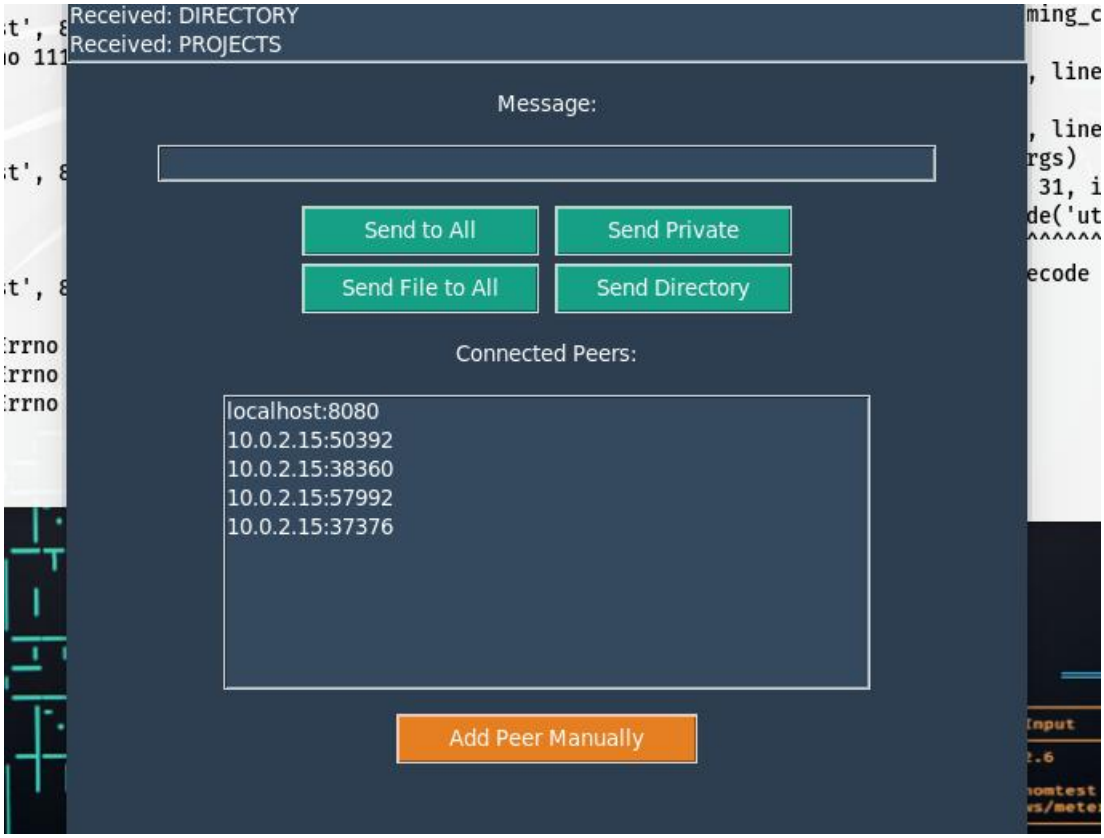
- Now if we check our folder we can that the sent files are downloaded in a folder called “received_files”.

- Inside the received_files folder we can that the file we sent is saved here.

```
┌──(root㉿kali)-[/home/laibaa/Desktop/cn_project/received_files]
└─# ls
peer2.py  peer8.py   server22.py  three.txt
peer6.py  server2.py  server3.py   yo.txt
```

12. We wil repeat the same process for sharing a directory with a peer.

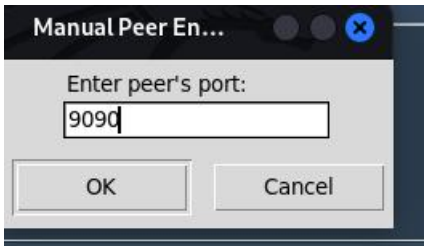


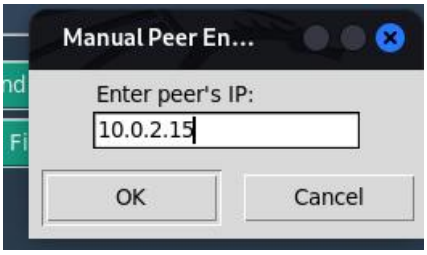


The directory Is received.

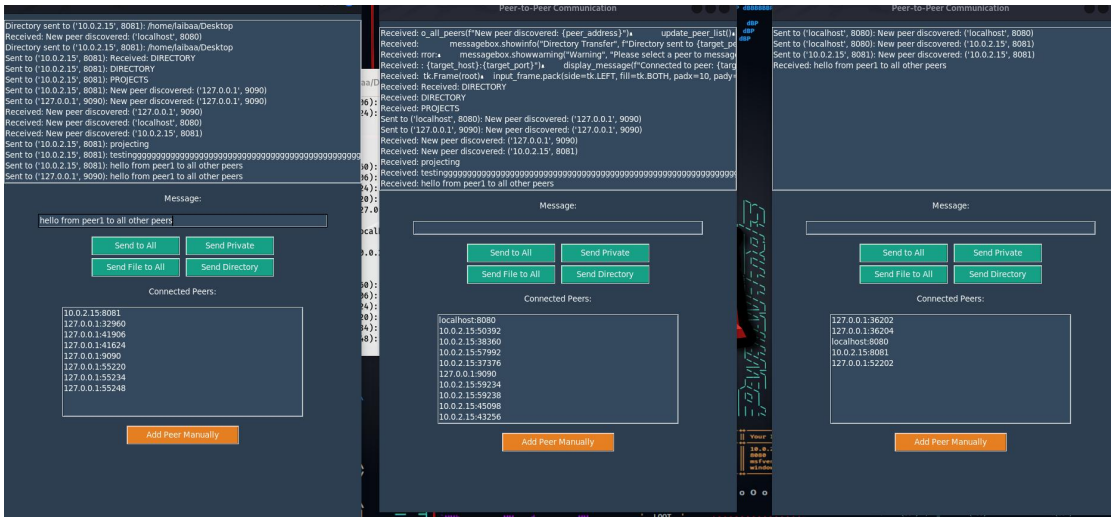
13. If the user wants to add another peer manually he can do so my clicking the option ‘add peer manualy’ .

- Then he’ll be prompted to add the peer’s ip and port number.





- The message is displayed saying the peer was added and messages were exchanged between peers once again.



1. Networking

- **Socket Programming**: The application uses both **TCP** and **UDP** sockets for communication.
- **Concurrency**: Threads handle multiple incoming connections, broadcasts, and GUI responsiveness simultaneously.
- **Broadcast Port**: UDP-based broadcasts use port 9090 for peer discovery.

2. Graphical User Interface (GUI)

- **Tkinter**: Provides an intuitive interface for interacting with the application.
- **Components**:
    - **Listboxes**: Display chat messages and the list of connected peers.
    - **Buttons**: Enable actions like sending messages, files, and directories.
    - **Input Fields**: Collect messages and peer details from users.

3. File and Directory Transfer

- **File Transfer**: Reads the file in chunks and sends it sequentially to maintain memory efficiency.
- **Directory Transfer**: Recursively traverses the directory and sends files while preserving relative paths.

Functionality Details

1. Peer Server Initialization

Function: start_peer_server(host, port)

Purpose: Initializes a server socket that listens for incoming connections from peers.

Workflow:

- The server binds to a specified host and port.
- It listens for incoming connections.

- Upon connection, it spawns a new thread to handle the communication.

Testing:

Test 1: Simulate multiple peers attempting to connect simultaneously. Ensure all connections are accepted and managed correctly.

Test 2: Introduce invalid connection requests (e.g., incorrect protocol or data format) and confirm the server gracefully handles errors.

2. Handling Incoming Connections

Function: handle_incoming_connections(peer_socket)

Purpose: Handles communication from a connected peer.

Workflow:

- Reads messages sent by the peer.
- Displays messages in the GUI.
- Handles disconnections and removes the peer from the list.

Testing:

Test 1: Send a variety of valid and malformed messages to test parsing and error handling.

Test 2: Simulate abrupt disconnection of a peer and ensure proper cleanup of resources.

3. Broadcasting Presence

Function: start_peer_discovery(host, port)

Purpose: Sends periodic broadcast messages to announce the peer's presence.

Workflow:

- A UDP socket broadcasts the host and port at regular intervals.
- Other peers listening for broadcasts can add this peer to their list.

Testing:

Test 1: Verify that all peers in the same network receive the broadcast messages.

Test 2: Test with different network setups (e.g., subnet isolation, firewalls) to evaluate behavior.

4. Listening for Broadcasts

Function: listen_for_broadcasts(host, port)

Purpose: Listens for broadcast messages to discover new peers.

Workflow:

- A UDP socket listens on a predefined port.
- Upon receiving a broadcast message, the peer's details are added to the list.

5. Sending Messages

Functions:

send_message_to_all_peers(message) – Sends a message to all peers.

send_private_message(target_peer, message) – Sends a message to a specific peer.

Purpose: Facilitates communication between peers.

Workflow:

- A TCP connection is established with the target peer(s).
- Messages are transmitted over the connection.

Testing:

Test 1: Send messages of varying lengths and confirm delivery.

Test 2: Simulate a peer going offline during communication and ensure the error is handled gracefully.

Test 3: Test message integrity by verifying that the received message matches the original.

6. File Sharing

Functions:

send_file_to_all_peers(file_path) – Transfers a file to all connected peers.

send_directory_to_peer(target_peer, directory_path) – Transfers a directory to a specific peer.

Purpose: Enables sharing of files and directories.

Workflow:

- For files, the file's name and contents are sent.
- For directories, all files are iterated, and their relative paths and contents are sent.

Testing:

- **Test 1:** Transfer files of various sizes and confirm successful reception.
- **Test 2:** Interrupt file transfer midway and verify recovery or graceful failure.
- **Test 3:** Send directories with nested subdirectories and large file counts, ensuring all files are received correctly.

7. Manually Adding Peers

Function: add_peer_manually()

Purpose: Allows users to manually add a peer by specifying its IP and port.

Workflow:

- Prompts the user for IP and port via GUI.
- Attempts to establish a connection.
- Adds the peer to the list if successful.

Testing:

Test 1: Input valid IP/port combinations and verify the peer is added.

Test 2: Input invalid or unreachable IP/port combinations and ensure errors are displayed appropriately.

8. Peer List Management

Functions:

update_peer_list() – Updates the GUI to display the current list of peers.

Purpose: Provides users with a real-time view of connected peers.

Workflow:

- Clears the list.
- Populates it with current peer details.

Testing:

Test 1: Add and remove peers dynamically and verify the list updates correctly.

Test 2: Test for synchronization issues when multiple peers are added simultaneously.

User Interface

Main GUI Components:

1. **Chat Display**:
    - Listbox to show sent and received messages.
2. **Peer List**:
    - Displays connected peers dynamically.

3. **Buttons**:
    - Send to All: Broadcast a message.
    - Send Private: Message a specific peer.
    - Send File to All: Share files.
    - Send Directory: Share directories.
    - Add Peer Manually: Add peers via IP and port.
4. **Message Entry Field**:
    - Textbox to input messages.

Testing

Testing Scenarios

1. **Peer Discovery**:
    - Validate automatic detection of new peers on the network.
2. **Broadcast Messaging**:
    - Ensure messages are sent and received across all peers.
3. **Private Messaging**:
    - Verify targeted delivery of messages to specific peers.
4. **File and Directory Sharing**:
    - Test reliable transfer of files and directories between peers.
5. **Manual Peer Addition**:
    - Confirm successful addition of peers by specifying IP and Port.

Observations

- Broadcast discovery works seamlessly on local networks.
- File sharing is reliable and preserves file integrity.
- GUI remains responsive during heavy data transfers due to threading.

Challenges and Solutions

1. Peer Disconnection

- **Issue**: Detecting disconnected peers.
- **Solution**: Implement periodic health checks to validate active connections.

2. File Size Limitation

- **Issue**: Large files may encounter memory or timeout issues.
- **Solution**: Implement chunked file transfer and acknowledgment mechanisms.

3. Cross-Network Communication

- **Issue**: The application currently supports only local networks.
- **Solution**: Add NAT traversal using libraries like STUN or TURN.

Future Enhancements

1. **Encryption**:
    - Add end-to-end encryption for secure communication and file sharing.
2. **Cross-Network Support**:
    - Implement NAT traversal for internet-wide peer communication.
3. **Peer Management**:
    - Provide options to remove or block peers.
4. **Improved GUI**:
    - Add themes, user profiles, and status indicators.
5. **Resumable Transfers**:

- Enable pausing and resuming file or directory transfers.

Conclusion

This P2P communication application is a robust framework for local network communication and file sharing. By integrating advanced features like encryption, cross-network support, and enhanced peer management, it has the potential to evolve into a comprehensive communication tool for diverse use cases.
