

Abstract

This project presents the design and implementation of a Peer-to-Peer (P2P) chat application using socket programming in Python. The application enables two users to establish a direct TCP connection and exchange messages in real-time without requiring a centralized server. The primary objectives were to gain hands-on experience with socket APIs, concurrency, and custom communication protocols.

The system was implemented using Python's built-in socket and threading libraries, ensuring cross-platform compatibility and minimal external dependencies. The chat application supports usernames, graceful termination using the /quit command, and file transfer functionality through a simple custom protocol. The results demonstrate successful message exchange and attachment transfer between peers on the same network. This project highlights the importance of understanding fundamental networking concepts, concurrent programming, and protocol design in real-world applications.

Introduction

Problem Statement

Traditional messaging systems often rely on centralized servers, which add overhead and reduce privacy. The challenge addressed in this project is to develop a lightweight, direct communication mechanism between peers using sockets.

Objectives

- Implement a peer-to-peer chat system using TCP sockets.
- Enable real-time, bi-directional communication between two peers.
- Integrate concurrency for simultaneous sending and receiving of messages.
- Add support for custom commands like /quit and /send.

Significance

The project demonstrates practical application of networking concepts taught in Computer Networks, bridging theory with real implementation. It also lays the foundation for developing more advanced distributed systems, such as secure messengers and collaborative tools.

Methodology

Approach

- Designed a peer-to-peer communication model where one peer runs in "listen" mode and another in "connect" mode.
- Implemented concurrency using Python threads for full-duplex communication.

- Developed a simple custom protocol for distinguishing text messages from file transfers.

Design Steps

1. Implemented an echo server and client to learn socket basics.
2. Added threading to allow simultaneous send and receive.
3. Unified server and client into a single peer program (chat.py).
4. Extended the program with usernames, /quit command, and /send file transfer support.

Algorithms / Protocols Used

- **Messaging Protocol:** newline-delimited text messages.
- **File Transfer Protocol:**
 - Header: /file filename size\n
 - Payload: raw bytes of the file.

Tools & Technologies

- **Language:** Python 3.11
- **Libraries:** socket, threading, argparse, os
- **Editor:** Visual Studio Code
- **Platform:** macOS (tested on LAN)

Results and Discussion

Figure 1: Real-time messaging between peers

Demonstrates successful exchange of text messages between 2 peers with usernames.

```

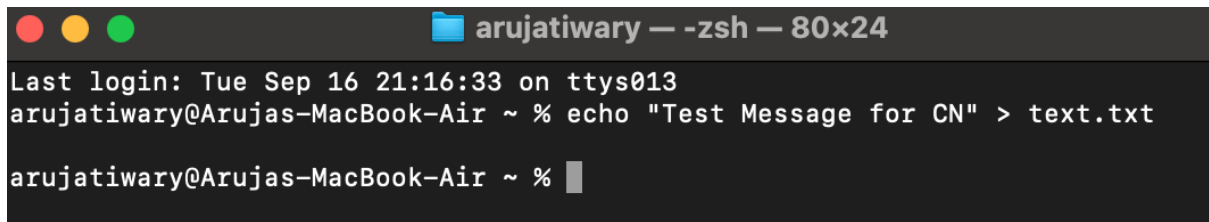
source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
● arujatiwary@Arujas-MacBook-Air CN Project % source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
● (venv) arujatiwary@Arujas-MacBook-Air CN Project % python3 chat.py listen --port 5050 --name I049
[System] Listening on 0.0.0.0:5050 ...
[System] Connected by ('127.0.0.1', 50483)
> Hi! My name is Lubhana.
[I056] Hi! I'm Daksh.
> /quit
[I056] has left the chat.
> [System] Exiting.
○ (venv) arujatiwary@Arujas-MacBook-Air CN Project %

source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
● arujatiwary@Arujas-MacBook-Air CN Project % source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
● (venv) arujatiwary@Arujas-MacBook-Air CN Project % python3 chat.py connect 127.0.0.1 5050 --name I056
[System] Connected to 127.0.0.1:5050
[I049] Hi! My name is Lubhana.
> Hi! I'm Daksh.
[I049] has left the chat.
> /quit
[System] Peer disconnected.
[System] Exiting.
○ (venv) arujatiwary@Arujas-MacBook-Air CN Project %

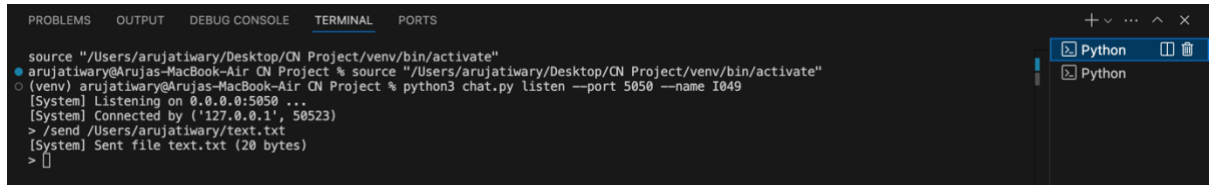
```

Figure 2: Text file transfer in progress

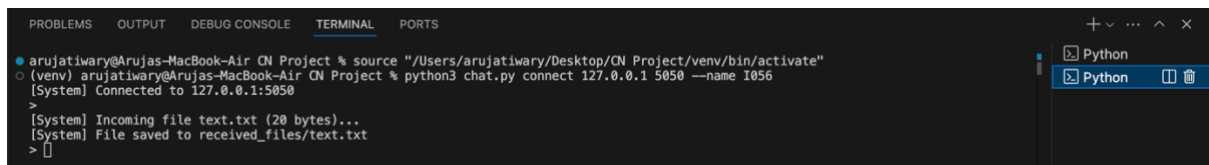
Shows Peer A receiving a file test.txt from Peer B, automatically saved in the received_files/ directory.



```
arujatiwary — -zsh — 80x24
Last login: Tue Sep 16 21:16:33 on ttys013
arujatiwary@Arujas-MacBook-Air ~ % echo "Test Message for CN" > text.txt
arujatiwary@Arujas-MacBook-Air ~ %
```



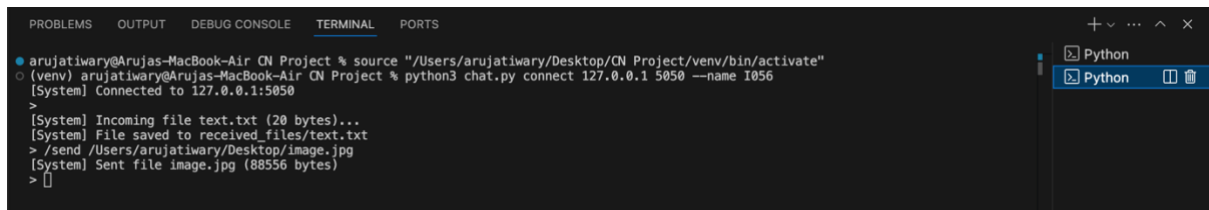
```
source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
arujatiwary@Arujas-MacBook-Air CN Project % source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
arujatiwary@Arujas-MacBook-Air CN Project % python3 chat.py listen --port 5050 --name I049
[System] Listening on 0.0.0.0:5050 ...
[System] Connected by ('127.0.0.1', 50523)
> /send /Users/arujatiwary/text.txt
[System] Sent file text.txt (20 bytes)
>
```



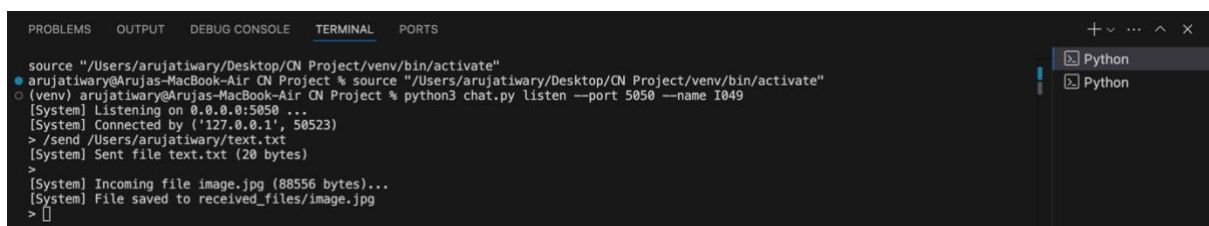
```
arujatiwary@Arujas-MacBook-Air CN Project % source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
arujatiwary@Arujas-MacBook-Air CN Project % python3 chat.py connect 127.0.0.1 5050 --name I056
[System] Connected to 127.0.0.1:5050
>
[System] Incoming file text.txt (20 bytes)...
[System] File saved to received_files/text.txt
>
```

Figure 3: Image file transfer in progress

Shows Peer A receiving a file image.jpg from Peer B, automatically saved in the received_files/ directory.



```
arujatiwary@Arujas-MacBook-Air CN Project % source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
arujatiwary@Arujas-MacBook-Air CN Project % python3 chat.py connect 127.0.0.1 5050 --name I056
[System] Connected to 127.0.0.1:5050
>
[System] Incoming file text.txt (20 bytes)...
[System] File saved to received_files/text.txt
> /send /Users/arujatiwary/Desktop/image.jpg
[System] Sent file image.jpg (88556 bytes)
>
```



```
source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
arujatiwary@Arujas-MacBook-Air CN Project % source "/Users/arujatiwary/Desktop/CN Project/venv/bin/activate"
arujatiwary@Arujas-MacBook-Air CN Project % python3 chat.py listen --port 5050 --name I049
[System] Listening on 0.0.0.0:5050 ...
[System] Connected by ('127.0.0.1', 50523)
> /send /Users/arujatiwary/text.txt
[System] Sent file text.txt (20 bytes)
>
[System] Incoming file image.jpg (88556 bytes)...
[System] File saved to received_files/image.jpg
>
```

Discussion

The application successfully enables direct peer-to-peer messaging and file transfer.

- **Efficiency:** Lightweight implementation using standard libraries.
- **Reliability:** TCP sockets ensure ordered and reliable message delivery.

- **Limitations:**
 - Works best on local/LAN; requires NAT traversal or port forwarding for Internet use.
 - Supports only two peers in the basic version.
 - Text-based interface may not be user-friendly for non-technical users.
-

Conclusion

The project achieved its objectives by building a functional peer-to-peer chat application using Python sockets. It reinforced concepts of networking, concurrency, and custom protocol design. While the application is currently limited to LAN-based communication between two peers, it can be extended into a full-fledged messaging platform with encryption, GUI, and group chat features. This project provided valuable hands-on experience with the underlying mechanics of real-world network applications.

References

1. Python Software Foundation. *socket* — *Low-level networking interface*. Available at: <https://docs.python.org/3/library/socket.html>
2. Python Software Foundation. *threading* — *Thread-based parallelism*. Available at: <https://docs.python.org/3/library/threading.html>
3. Forouzan, B. A. (2017). *Data Communications and Networking*. McGraw Hill Education.
4. Kurose, J. F., & Ross, K. W. (2017). *Computer Networking: A Top-Down Approach*. Pearson.
5. GeeksforGeeks. *Socket Programming in Python*. Available at: <https://www.geeksforgeeks.org/socket-programming-python/>
6. Real Python. *Socket Programming in Python (Guide)*. Available at: <https://realpython.com/python-sockets/>