MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4981-4985

A Decentralized and Privacy-Focused Chat Application using Networking and Socketing

Akkala Abhilasha ,
Department of computer science and engineering, Geetanjali college of engineering and technology aabhilasha.cse@gcet.edu.
Avinash seekoli,
Department of computer science and engineering, St.Martins engineeringcollege,avinash.seekoli@gmail.com.

1. ABSTRACT
Existing messaging and chat applications often rely on centralized servers, raising concerns about privacy, data security, and potential censorship. While some applications claim to offer end-to-end encryption, the centralized nature of their architecture still exposes users' metadata, leaving them vulnerable to surveillance and potential data breaches. The disadvantages of centralized chat systems include the inherent trust required in the service provider, the risk of data misuse or disclosure, and the potential for censorship or service disruption by external entities. Moreover, these systems are susceptible to single points of failure, hindering their reliability and resilience. This paper proposes the development of "Secure Chat," a decentralized, peer-to-peer (P2P) chat application built using Python. Secure Chat aims to address the limitations of centralized messaging systems by leveraging the principles of decentralization and end-to-end encryption. By eliminating the need for a central server, Secure Chat ensures that no third party can access or monitor users' conversations, providing a truly private and secure communication channel. The proposed system will utilize advanced cryptographic techniques, such as elliptic curve cryptography and perfect forward secrecy, to ensure that messages are encrypted in transit and at rest, protecting them from unauthorized access. Additionally, SecureChat will implement a distributed hash table (DHT) or other P2P networking protocol to enable direct communication between peers, eliminating the need for a central server and ensuring resilience against censorship or service disruption.
***Keywords**:* Decentralized, peer-to-peer (P2P), chat application, Python, privacy, data security, end-to-end encryption.

2. INTRODUCTION
In the 1980s, Guido van Rossum developed Python language after the success of the ABC Programming Language. The code written in this language is very simple and readable, making it easy to understand. It doesn't require punctuation at the end of each line as C programming language instead uses indentation (i.e. equal spacing) depending on the block of code written. Python's syntax is very brief and readable helping the programmers to reduce the number of lines of code to execute on any particular notion. In 2000, Python 2.0 was released by incorporating list comprehensions and garbage collections. In the year 2008, Python 3.0 was released marking a significant milestone in its version by making it compatible and user-friendly, resolving all its inconsistencies gained a lot of popularity among users. Today, Python is one of the most widely used programming languages for various applications.Threading is a module, predefined in the Python library for execution of code with a separate flow. In programming, the entire code written by programmers is executed by using single a thread which is also known as the 'Main thread'. To execute any block of code separately we need to use threads. We can create multiple threads in a program and can execute them simultaneously. In theory, the thread module is used to execute different blocks of code simultaneously reducing the time of the program but when the code runs a single thread function at a particular period remaining threads are in sleep until the executing thread stops. The main thread is used to execute all the remaining threads in the program. We can import the thread module into the code by the line 'import thread' at the beginning of the code.The Python built-in socket module lets users talk to each other directly. Sockets are necessary for the basic functioning of the app, they let the app communicate to other instances of the app on the network, to send and receive messages while chatting. The app gets rid of the need for a central server, so users can just chat with each other directly.
To use the socket module work in our code, we can use the import statement (`import socket`) to use its features in our program. This lets the app make connections with other devices, send and receive messages through these connections, making P2P communication easy.We included all the above-mentioned modules in our program. The socket module to make connections with other devices, send and receive messages through these connections. The threading module facilitates concurrent execution, particularly for initiating server and client modules at the same time without freezing the main program.

3. LITERATURE SURVEY
If we think of the internet as a single web that connects all our devices together. In the end it is just a computer network, smoothly shared by the nodes [1]. These were the primitive roots of load balancing code, and it was back in 1962 when servers started connecting to each other. Nowadays, the networks are more complex and have to cope with security (preventing people from hacking into them). People like Manpreet Kaur study these networks, how they work, the rules that govern them (protocols), and all of the awesome things they enables us to do like watching movies / gaming / online.A network is like a big group of computers who are connected and can share information with each other. These networks can be made up of just two computers, or even millions, all connected together [2]. Computer networks are also sometimes called data networks, because they're used to move data (like files, pictures, and videos) from one computer to another. To do this, the computers in the network use special connections called links. These links can be made of wires (like Ethernet cables) or even be wireless (like Wi-Fi). The little pieces of information that travel through these links are called packets. The most well-known example of a computer network is the Internet. All the different computers (or nodes) in a network can do different jobs. For example, some nodes might be used to store information (like servers), while others help direct traffic between the nodes (like routers). Even your own laptop or phone can be part of a network! In her research, Sonia Rana dives deep into how computer networks work, exploring different types of connections between nodes and explaining the roles of different types of nodes. By understanding how data moves through these networks and how all the different parts fit together, we can gain a better understanding of the basics of computer networking.
Computer networks are like the backbone of modern communication, allowing us to share digital information between different devices and systems [3]. They're made up of various nodes, like computers, smartphones, servers, and network hardware like routers and switches. These networks help us access and share resources like the Internet, shared servers for apps and storage, printers, and even email and IM services. Basically, computer networks make it super easy for us to work together and get stuff done. There are two main types of computer networks: open and closed. Open networks are designed to be easily connected to other networks, making it easy for us to share info and collaborate. On the flip side, closed networks are more secure because they're harder to access and aren't as connected to other networks. This extra layer of security can be helpful when we want to keep our stuff private. But like anything else, computer networks have their challenges. One of the biggest is dealing with resource attacks, where people try to take advantage of our networks and use up all our resources. This can slow down our networks and make it hard for us to get stuff done. Luckily, there are lots of people out there like Dhyan Patel who are studying these networks and working on ways to make them more secure and efficient. In his research paper, Patel gives us a deep dive into the basics of computer networks. He explains how they're made up of nodes and how these nodes can be connected together through different types of connections. He also talks about distributed processing, which is when we split up tasks between multiple computers to make them go faster and use fewer resources. It's like when you have a big project, and you divide it up into smaller parts so you can all work on it together.

Sensor nodes, those little computers that cooperated wirelessly, We have built mechanisms that let us send new programs over to these nodes, but there isn't a very secure way to ensure that the program hasn't been tampered with in transit [4]. This study addresses this problem. It developed a mechanism where instead of sending the program at once, progressives escaped their first 2a destination. Every piece comes with a unique “code” that guarantees it is non-fungible, even better: they get linked to the previous one (just think of them as links in a chain). That way, they can still make sure they received the program that was sent to them even on these tiny computers.While a number of user-level protocols have been created to narrow the gap between physical network performance and what's actually available to applications, older applications built on kernel-level protocols like TCP haven't really been given much attention [5]. There's a need to make these older TCP apps work with newer user-level protocols like EMP or VIA, which offer both low latency and high bandwidth. We came up with a plan to support these apps, written using the sockets API, to run over EMP without requiring any changes to the apps themselves. Using this scheme, we were able to achieve a latency of 28.5 /micros/ for Datagram sockets and 37 /micros/ for Data Streaming sockets, compared to a latency of 120 /micros/ with TCP for 4-byte messages. This method gets us a peak bandwidth of around 840 Mbps, which is pretty close to what EMP can handle. When we tested the ftp app, it showed twice as much improvement on our sockets interface compared to TCP, while the Web server app saw up to six times more performance boost. This is the first design and implementation of this kind for Gigabit Ethernet, to the best of our knowledge.

This paper takes a look at how using parallel TCP flows can help boost end-to-end network performance for distributed data-intensive applications [6]. To test this theory, a series of transmission experiments were carried out on a wide-area network to see how much of an improvement parallel flow bring to throughput, and also to find out how many parallel flows are needed to improve throughput without causing congestion. The results of these experiments are then used to create an empirical throughput expression for parallel flows. Based on these findings, the paper also provides some guidelines for the effective use of parallel flows in various situations.

Transmission Control Protocol (TCP) is a protocol used by apps to make sure data is transferred reliably, like when you're sending a really important file to a friend or something. At first, TCP was designed for unreliable networks, but as high-speed wide area networks became more popular, people started looking for ways to make TCP work better and faster [7]. One way to do this is by having system administrators (like, the people in charge of making sure your computer network runs smoothly) tweak the network settings, but this can take a long time and be really complicated. With the creation PSockets (which stands for Parallel Sockets), which is a library that lets you improve TCP's performance without having to mess with network settings. The basic idea behind PSockets is to use "network striping," which basically means splitting up data and sending it across multiple open sockets (like, little pathways for information to travel through on the internet). We tested this out on the Abilene network and found that network striping using PSockets really works well for high-performance data-intensive computing applications where the data is spread out across different locations. The research shows that with PSockets, you can get TCP to work better and faster without having to rely on system administrators to do all the hard work for you. This could be really useful for people who need their computers to work fast and reliably, like scientists or engineers or anyone else who relies on big data sets spread out across the globe.

Network protocols are really hard to implement correctly[8]. Like, it's not easy to make sure all the implementations are the same, even when there are standards and stuff like RFCs. One reason for this is that the specs are usually kind of informal, so there's always room for interpretation. Conformance testing against those specs can be a pain. In this paper they we're like, hey, we got an idea for a better way to do it. We made a new technique for rigorous protocol specs that lets you test them really well. They tried it out on TCP, UDP, and the Sockets API, and made like a really detailed spec for them and tested it against three different implementations: FreeBSD 4.6, Linux 2.4.20-8, and Windows XP SP1. They found some differences and inconsistencies between them. But it worked for TCP, which is super complex!

In this paper we have examined the client/server (C/S) model for software design with special emphasis to socket based programming principles discussed in [9]. It also explains the communication between the client and server, along with some examples for the connection-oriented service programs that interact over a network using sockets. Transmission Layer offers two kinds of services: Connection-oriented, which uses TCP protocol and Non-connection oriented service uses UDP. The two types of sockets are different and knowing those differences is critical to creating effective, efficient client service programs. This enables us to know what to take care of in which protocol and what not, making our client service programs more resilient, robust and less resource sapping.

Think of two such systems, as if it were a special equipment: "reliable socket" and "reliable packets"[10]. The idea is to help keep internet connections nice and stable even if a device moves around in the world — or issues some random disconnects. Cryptography and connectivity-related services are also being offered, which work as part of an application framework. They run in the background capturing data that is being sent and when there is a glitch, they do the necessary to make the communication work again. But not so that the applications you use (e.g., web browsers) are interrupted and your data flows. It adds a small amount of slowdown but the extra encryption for data more than makes up for it.

4. PROPOSED METHODOLOGY

This article outlines development approach for a Peer-to-Peer chat application. The application uses Python's built in libraries. Tkinter for creating a graphical user interface. Socket for enabling network communication between connected devices.

The user interface is designed to be straightforward. It is divided into two distinct sections for both server and client functionalities. Users acting in server role can specify their desired IP address and port number. A designated "Start Server" button initiates server mode. It essentially creates virtual chat room. Client functionality allows users to enter their friend's IP address and port number.

A separate "Connect" button establishes connection with specified server forming direct communication channel. Both sections share chat window to display exchanged messages and input field for composing new messages. Finally, "Send" button allows users to transmit their messages.

For network communication application relies on sockets. These sockets are established using the socket.socket(socket.AF_INET, socket.SOCK_STREAM) function. In server mode application binds server socket to chosen IP address and port number using the bind method. It then enters listening state for incoming connections. A dedicated thread named accept_connections is created to manage incoming connections. Upon successful connection establishment. The server creates another thread, handle_client. This is to specifically manage communication with that particular client. Server receives messages from connected clients and displays them within chat window.

On client side. The application utilizes connect function to establish connection with server using provided IP address and port number. A separate thread named receive_messages, is created to continuously receive messages from server. These received messages are then displayed within the client's chat window.

The application utilizes threading to manage concurrent tasks efficiently. On server side accept_connections thread handles accepting new connections. While the handle_client thread manages communication. It deals with individual clients. Similarly, on client side receive_messages thread is responsible for receiving messages from server. This threading approach makes sure that user interface remains responsive.

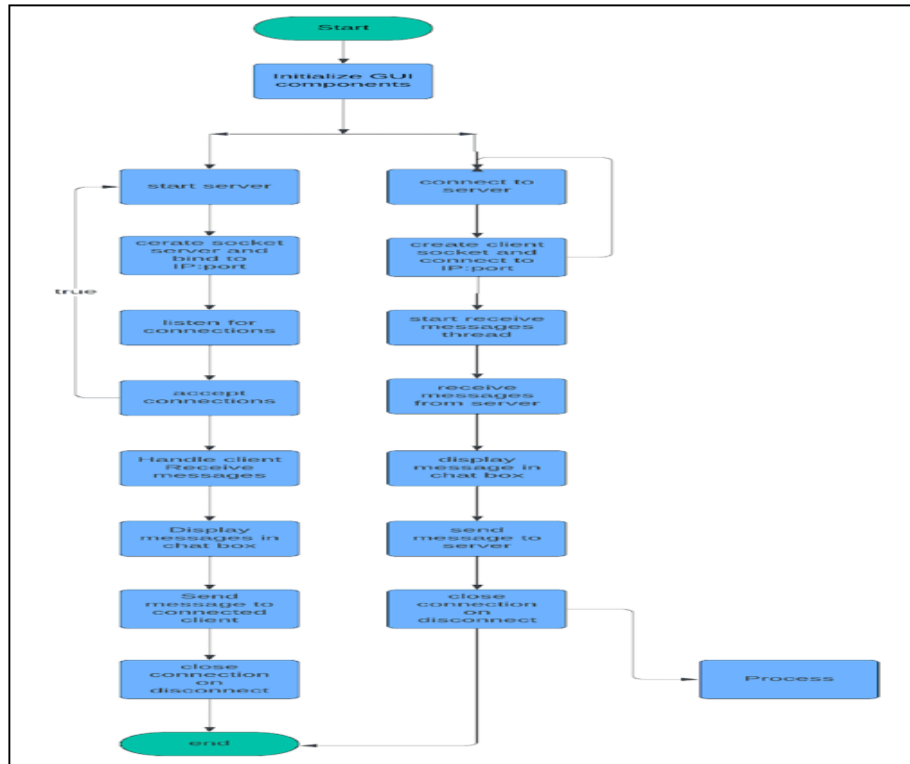
A function named add_message is used to incorporate both received and sent messages into chat window. Ensuring proper formatting for clarity. Clicking the "Send" button triggers the send_message function. For clients this function transmits message to server. It uses the established socket. However, for servers running without a connected client an error message is displayed. Indicating sending messages directly from server window is not permitted.

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4981-4985

Basic error handling mechanisms are incorporated using try-except blocks. These blocks catch potential exceptions that might arise during socket creation. They also handle issues during connection establishment or data transfer. Error messages are displayed using messagebox.showerror. This approach is taken to notify user of any issues.

5. SYSTEM ARCHITECTURE

5.1 FLOWCHART



Let us have a brief overview on this flowchart i.e**. Fig 5.1**.

Starting the application will display a new window. To start communication, please click on the "Start server" button, which will activate the server function that allows other users to connect to your program. If you wish to connect to another user, you must input thier IP address and socket number that you will use to access their server. Thus, both parties need to execute the server to establish a two-way connection. Once connected, the program will display any messages you type in the text box and click "Send", and the reciever will receive your messages in thier chat box, and also when the user types messages, they will appear on your screen, and thus allowing real time communication. If you want to close the connection, simply close the application window.

6.ALGORITHM

Server-Side:

Step 1: Start (The server application launches.)

Step 2: The server sets up its GUI elements, variables, and imports modules (Display message).

Step 3: User clicks "start server" (Get input from the user).

Step 4: If input is "start server", continue. Otherwise, repeat step 3 (If input is not 'start server', go to step 3).

Step 5: Server retrieves IP address and port from designated fields (Get input from the user).

Step 6: A server socket is created and bound to the specified IP and port (If input is 'start server', then display menu).

Step 7: The server enters a listening state for incoming connections (Check the input and validate...).

Step 8: A thread named accept_connections is created to manage incoming connections (Create a new thread...).

Step 1: The client application launches.

Step 2: The client sets up its GUI elements, variables, and imports modules (Display message).

Step 3: User interacts with the application (Get input from the user).

Step 3 & 4: - If the user clicks "connect," continue (If input is "connect", go to step 5). - Otherwise, repeat step 3 (If input is not 'connect', go to step 3).

Step 5: Client retrieves its IP address and port from designated fields (Get input from the user).

Step 6: A client socket is created (Create a new socket...).

Step 7: The client attempts to connect to the server using the retrieved IP address, port number, and the connect method (Connect the socket to the server...).

Step8: - If the connection is successful, a message is displayed, and proceed to step 9 (If connected successfully...). - Otherwise, an error message is displayed, and the user might need to retry (If there is an error...).

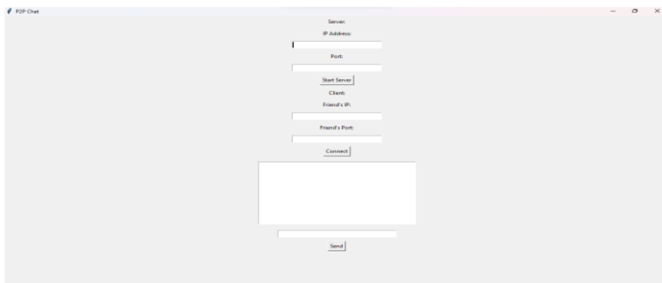
Step 9: A dedicated thread named receive_messages is created to handle receiving messages from the server (Create a new thread...).

Main Event Loop:

Throughout: The main event loop of the TKinter application remains active (This process occurs throughout the application's execution). This loop continuously handles user interactions with the GUI elements and updates the chat window accordingly (handling user interactions with GUI components and updating the chat box accordingly).

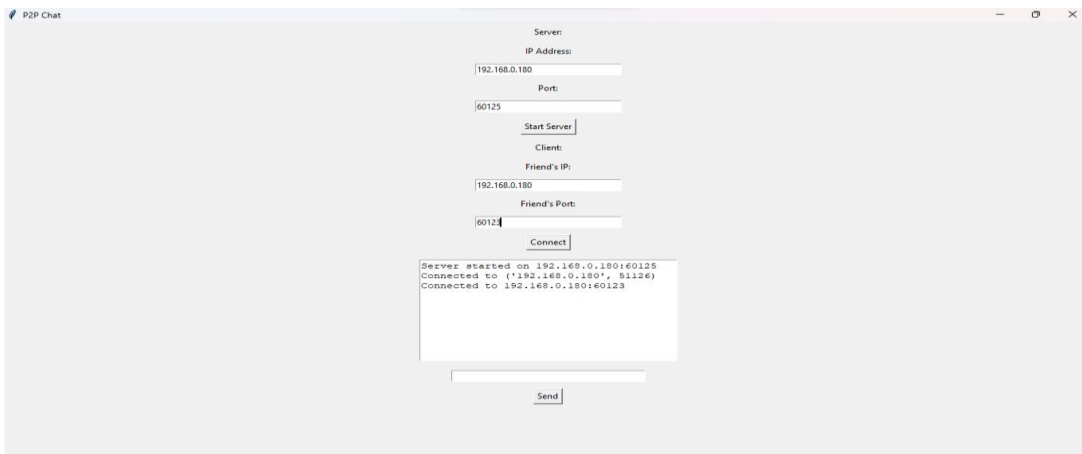
7. EXPERIMENTAL RESULTS

FIG 7.1

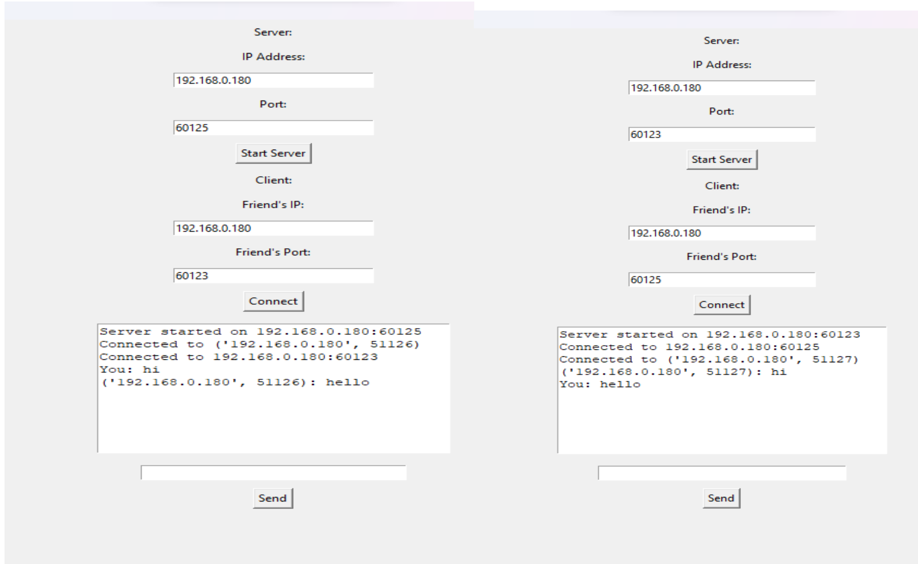


In fig 7.1, We initiate the application and a new chat windo opens.

In fig 7.2, after initiating the application , we start a server by specifying our IP and port.

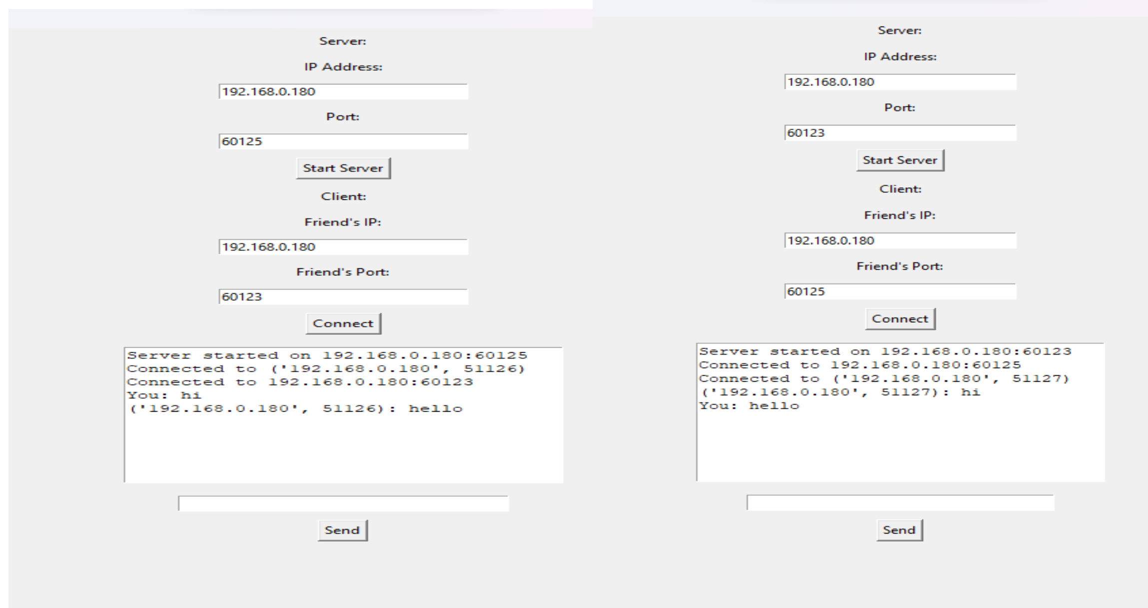


In fig 7.3, after starting a server instance, we connect to others by specifying their IP and port with which they have started their sever instance and vice versa.



MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4981-4985

In fig 7.3, after starting a server instance, we connect to others by specifying their IP and port with which they have started their sever instance and vice versa.



In fig 7.4 & fig 7.5, after establishing a connection, both users can commence real-time communication by typing messages into the text box and clicking "send" to dispatch the message.**GRAPHICAL REPRESENTATION**

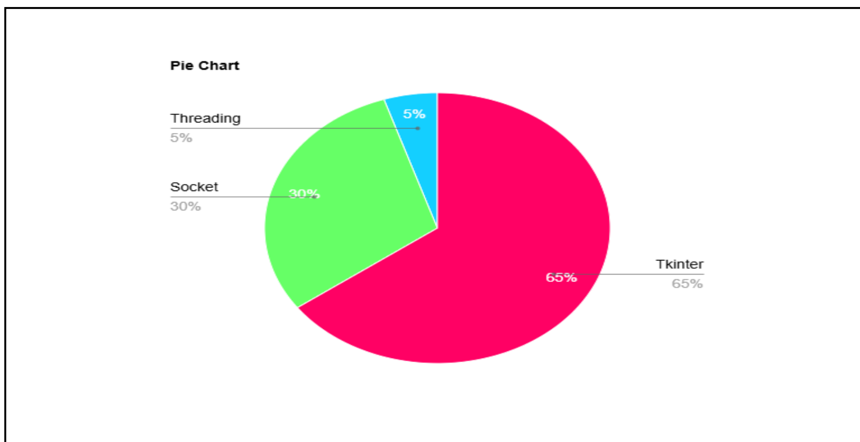


Fig 7.6 – Represents percentage of modules used in this program

8. CONCLUSION

This peer-to-peer (P2P) messaging tool changes the existing chat systems by establishing direct connections between users, removing the vulnerabilities present in centralized platforms. Unlike conventional applications dependent on centralized servers, it uses Python's tkinter library for a responsive graphical interface (GUI) and the socket library for seamless network communication. This decentralized architecture eliminates the necessity for a central authority to oversee data storage or communication management, effectively addressing issues regarding privacy and censorship. Users can operate as hosts or guests, with hosts designating their IP address and port number to create virtual chat rooms. Guests can then access these rooms by inputting the host's IP and port details. Both roles share a chat interface where messages traverse via sockets managed by Python's socket library, ensuring smooth conversation flow.

9. FUTURE ENHNACEMENT

This initial iteration of our P2P chat application leverages capabilities we currently possess. However, the development roadmap envisions exciting new functionalities. These will further enhance user experience. Our primary focus is on bolstering security. This will integrate robust cryptographic libraries. This ensures confidentiality of communication between users. Moreover, we aim to streamline contact and chat management. We will introduce dedicated database. This database will empower users. They can effortlessly add edit, or delete contacts. It will conveniently save and load chat history. Users can also conduct efficient searches through past conversations.Recognizing potential limitations of manual user discovery, we are actively exploring various mechanisms to automate process of finding other users within network. Two promising approaches are under consideration. A centralized server for more straightforward implementation. Or Distributed Hash Table (DHT) for increased scalability and enhanced security. Finally, to seamlessly integrate these new features we plan to refine user interface. Ensuring intuitive access to contact management functions. Chat history retrieval and potentially interaction with discovered peers if utilizing a DHT. Security will remain paramount throughout development process.

10. REFERENCES

1. A Research on Computer Networking: Protocol, Challenges & Applications (researchgate.net)
2. https://ijirt.org/master/publishedpaper/IJIRT142642_PAPER.pdf
3. https://www.semanticscholar.org/paper/A-Research-Paper-on-Basic-of-Computer-Network-Patel/81f90c2d2a5bd7eadbbd45235d43e5e1536dd11e
4. https://dl.acm.org/doi/abs/10.1145/1127777.1127826
5. https://ieeexplore.ieee.org/abstract/document/1137745
6. https://ieeexplore.ieee.org/abstract/document/1015527
7. https://ieeexplore.ieee.org/abstract/document/1592751 https://dl.acm.org/doi/abs/10.1145/1080091.1080123
8. https://ieeexplore.ieee.org/abstract/document/5232440
9. https://dl.acm.org/doi/abs/10.1145/570645.570657