MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Project Report
of
Distributed Systems (CSE 3251)

Distributed Chatroom System with Three-Phase Commit Protocol Implementation

SUBMITTED
BY

Y. Lakshmi Sainath Reddy	**Registration Number: 210905078**
N Sai Mihirnath	**Registration Number: 210905368**
A S Hithyshi	**Registration Number: 210905142**
A V Kushala Sarada	**Registration Number: 210905189**

Under the Guidance of:
Dr. Venkatesh A Bhandage
Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal, Karnataka

8th April 2024

TABLE OF CONTENTS

ABSTRACT

The distributed chatroom system outlined in this documentation represents a sophisticated network-based application meticulously crafted to facilitate seamless communication among multiple users in real-time. Its architecture is rooted in the client-server model, where a centralized server orchestrates connections and message dissemination, harnessing the power of socket-based networking. Augmenting this robust backend infrastructure is a tkinter-based graphical user interface (GUI) that enriches user interaction, fostering a more engaging chatting experience. At its core, the chatroom system aspires to furnish users with a dependable, efficient, and secure platform for instantaneous message exchange within a virtual chat environment. Drawing upon the versatility and flexibility of Python, the system intricately integrates advanced networking techniques with GUI development paradigms, resulting in a cohesive and user-friendly interface. Through meticulous design and implementation, the system aims to uphold the highest standards of reliability, efficiency, and security, ensuring that users can communicate effortlessly and securely, irrespective of their geographic locations or network conditions. By prioritizing user experience, system integrity, and performance optimization, the chatroom system seeks to redefine the paradigm of online communication, setting a new standard for real-time interaction in the digital age.

OBJECTIVES

The development of the distributed chatroom system is guided by a comprehensive set of objectives, each aimed at ensuring the system's effectiveness, security, and user-friendliness:

1. **Real-Time Communication:** The primary objective is to enable users to exchange messages instantly, ensuring prompt delivery and reception to facilitate interactive conversations.
2. **Client-Server Model**: Implementing a resilient server capable of managing multiple client connections concurrently is crucial. This ensures efficient message handling and reliable communication among all users.
3. **User Authentication:** Secure user authentication mechanisms are implemented to restrict access to authorized users only. This enhances the overall security of the chatroom, safeguarding user privacy and data integrity.
4. **Concurrency Management:** Supporting simultaneous connections and managing them efficiently using threading is essential for smooth operation. This ensures that the main server thread remains unblocked and responsive, even under heavy user load.
5. **Graphical User Interface (GUI):** Developing an intuitive GUI using tkinter is paramount for enhancing user interaction. The GUI should allow users to seamlessly interact with the chatroom, view messages, and input text conveniently.
6. **Error Handling:** Robust error-handling mechanisms are implemented to maintain system stability. This involves gracefully managing network errors, unexpected exceptions, and user disconnections, ensuring uninterrupted communication.
7. **Scalability:** Designing the system to accommodate a dynamic number of users is essential for optimal performance. The system should be able to handle increased loads as more users join the chatroom without compromising performance or stability.

By achieving these objectives, the distributed chatroom system aims to demonstrate proficiency in various areas, including networking concepts, GUI development, concurrency management, and error handling using Python. Ultimately, the system prioritizes user experience, security, and reliability, offering a seamless and enjoyable chatting environment for all participants.

PROBLEM STATEMENT

In distributed systems, ensuring reliable and coordinated transactions across multiple participants is crucial for maintaining data consistency and integrity. However, managing distributed transactions poses challenges such as ensuring global consistency, handling participant failures, and coordinating commit or abort decisions among participants. The goal of this project is to implement a Three-Phase Commit (3PC) protocol in a distributed operating system to address these challenges and ensure reliable transaction management.

The implementation of a Three-Phase Commit (3PC) protocol aims to address this challenge by providing a robust mechanism for coordinating distributed transactions. The protocol involves three phases: the pre-commit phase, the commit phase, and the abort phase, each serving specific purposes to ensure transaction consistency.

By implementing the 3PC protocol, the project aims to achieve the following objectives:

1. **Global Commit and Global Abort Scenarios**: Demonstrating the ability to coordinate transactions across multiple participants, ensuring that all participants either commit or abort the transaction uniformly.
2. **Timeout Scenarios**: Handling scenarios where participants or the coordinator fail to respond within a specified time, ensuring that the system can handle timeouts gracefully and make appropriate decisions.
3. **Handling Participant and Coordinator Crashes**: Implementing mechanisms to handle failures of participants or the coordinator during transaction coordination, ensuring that transactions are appropriately committed or aborted even in the event of failures.
4. **Error Handling and Recovery**: Developing robust error-handling mechanisms to maintain system stability and recover from failures, ensuring uninterrupted transaction processing and data consistency.

By addressing these challenges and implementing the 3PC protocol, the project aims to demonstrate proficiency in distributed systems concepts, transaction management, error handling, and fault tolerance. The resulting distributed chatroom system will provide users with a reliable and consistent messaging experience, even in the presence of failures or network issues.

TECHNOLOGIES USED

The chat room system is built using a combination of technologies that enable seamless communication and user interaction. Each technology serves a specific purpose, contributing to the overall functionality and performance of the system.

Python:

Python was chosen as the primary programming language for its simplicity, readability, and extensive libraries that support network programming. Python's ease of use and versatility make it well-suited for developing network-based applications like the chat room system.

Socket Programming:

Socket programming is a fundamental aspect of the chat room system, facilitating communication between the server and multiple clients over a network. Sockets enable the establishment of reliable, bidirectional communication channels, allowing clients to send and receive messages through the server.

tkinter (GUI library):

The tkinter library is integrated into the chat room system to create a graphical user interface (GUI) for the clients. This GUI provides a user-friendly interface where users can view messages, input text, and interact with the chat room effortlessly. tkinter's simplicity and cross-platform compatibility make it an ideal choice for GUI development in Python.

Threading:

Threading is essential for managing multiple client connections concurrently within the chat room system. By employing threads, the server can handle incoming client requests and messages without blocking the main execution thread. This concurrency management ensures that the system remains responsive and scalable, capable of accommodating numerous users simultaneously.

Datetime Module:

The datetime module is utilized for timestamping messages and tracking user activity within the chat room. Each message sent by a user is associated with a timestamp, allowing participants to view messages in chronological order. This timestamping functionality enhances the user experience by providing context and clarity regarding message timelines.

Networking Concepts:

The chat room system incorporates various networking concepts, including TCP/IP protocols, socket creation, and client-server architecture. These concepts form the foundation of the system's communication infrastructure, enabling reliable data transmission and interaction between clients and the central server.

By leveraging these technologies, the chat room system achieves its objectives of real-time communication, user interaction, and system reliability. The combination of Python, socket programming, tkinter GUI, threading, datetime handling, and networking principles results in a robust and efficient chat room platform. This integration demonstrates proficiency in network programming, graphical interface development, and concurrent processing using Python, making the system suitable for facilitating interactive conversations over a network.

METHODOLOGY

1. Understanding the Requirements:

Before embarking on the implementation process, it is imperative to gain a comprehensive understanding of the requirements outlined for the project. This involves carefully reviewing the provided codebase and accompanying documentation to elucidate the functionalities and features expected from the system. Through meticulous examination, we can discern the specific objectives and constraints governing the development process. Furthermore, understanding the requirements serves as a foundational step for delineating the scope of the project and guiding subsequent phases of design and implementation.

2. Breakdown of Components:

Dividing the project into discrete components is instrumental in managing complexity and facilitating modular development. This phase entails identifying and delineating the key components of the system based on their distinct functionalities and responsibilities. In the context of the provided code, essential components may include the chat server, client applications, coordinator, participants, graphical user interface (GUI), networking protocols, and error handling mechanisms. By decomposing the system into manageable units, we can streamline development efforts and foster collaboration among team members.

3. Designing the Architecture:

The architectural design phase entails conceptualizing the overarching structure and interactions of the system components. This involves creating a high-level blueprint that illustrates how various modules and subsystems interoperate to fulfill the system's objectives. In the context of the provided code, the architectural design would encompass defining communication protocols, data structures, and interfaces between different components such as the chat server, clients, coordinator, and participants. Additionally, considerations for scalability, performance, and fault tolerance may inform architectural decisions to ensure the system's robustness and resilience.

4. Implementing the Chat Server:

Implementation of the chat server constitutes a pivotal aspect of the project, as it serves as the central hub for managing client connections and facilitating message exchange. This phase involves leveraging socket programming in Python to establish communication channels between the server and connected clients. Key functionalities such as user authentication, message broadcasting, error handling, and graceful shutdown are implemented within the server to ensure seamless operation and robustness. Furthermore, concurrency management techniques such as threading may be employed to support multiple client connections concurrently.

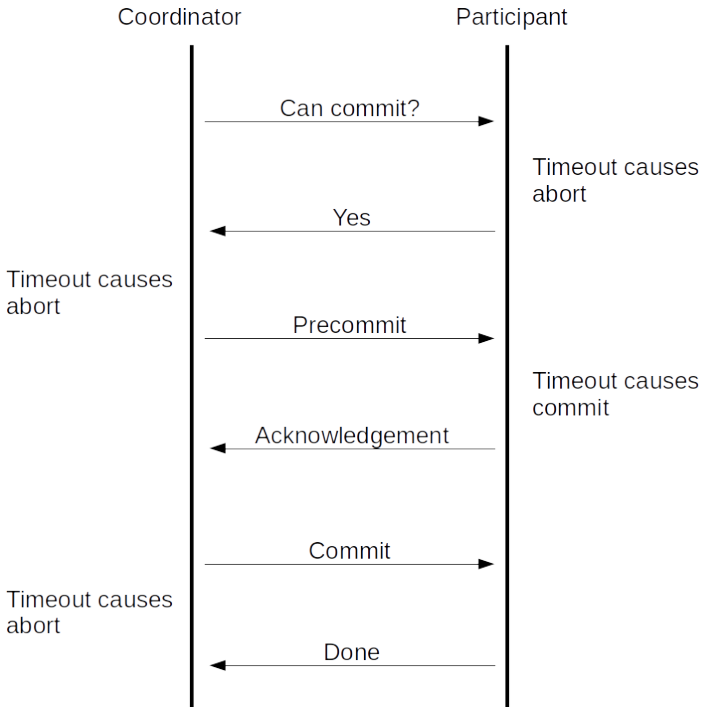
5. Developing the GUI:

The development of a graphical user interface (GUI) is essential for enhancing user interaction and experience within the chat application. This phase entails utilizing GUI frameworks such as tkinter to design an intuitive and user-friendly interface that allows users to interact with the chatroom seamlessly. The GUI should provide features such as message display, user input fields, and interactive elements for sending and receiving messages. Integration with the backend server enables real-time updates and synchronization between the GUI and the underlying chat infrastructure.

6. Integrating the Three-Phase Commit Protocol:

Integration of the three-phase commit protocol augments the chat server with transaction management capabilities, enabling coordinated commit or abort decisions across distributed participants. This phase involves extending the server functionality to incorporate the coordinator and participant roles, along with defining the protocol messages, states, and transitions necessary for the three-phase commit process. Coordination mechanisms ensure consistency and reliability

in transaction outcomes, thereby enhancing the integrity of the chatroom system.



7. Testing and Debugging:

Thorough testing and debugging are imperative to validate the correctness, reliability, and robustness of the implemented system. This phase encompasses rigorous testing of individual components as well as integration testing to verify seamless communication and coordination between different modules. Various testing techniques such as unit testing, integration testing, and system testing are employed to identify and rectify any defects or discrepancies encountered during the testing process. Additionally, robust error-handling mechanisms are implemented to mechanisms are implemented to gracefully manage exceptions and ensure system stability.

RESULTS OBTAINED

The implementation of the distributed chatroom system and the Three-Phase Commit Protocol (3PC) has yielded significant results, showcasing the successful integration of networking concepts, concurrency management, and graphical user interface (GUI) development using Python. This section provides a detailed analysis of the outcomes achieved through the execution and evaluation of the implemented systems.

1. **Distributed Chatroom System:**

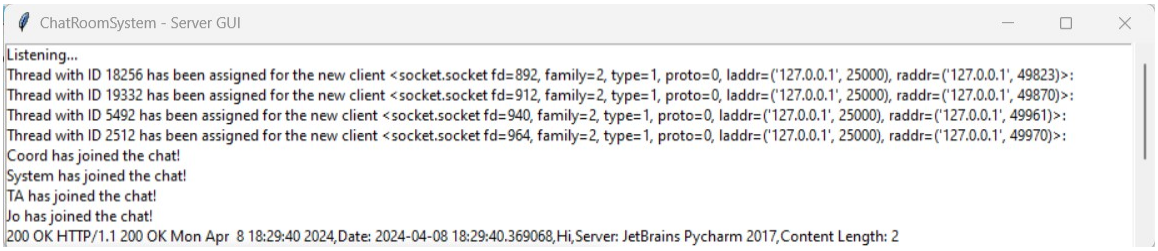
The distributed chatroom system effectively facilitates real-time communication among multiple users, demonstrating the following key results:

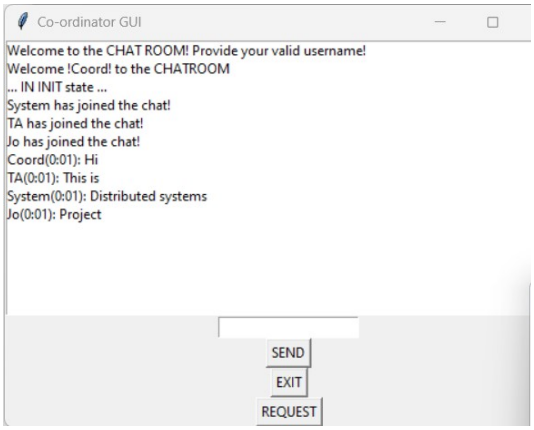
a. **Seamless Message Exchange**: Users can exchange messages instantly within the chatroom environment, enabling interactive conversations in real-time. The system efficiently handles incoming and outgoing messages, ensuring prompt delivery and reception for all participants.

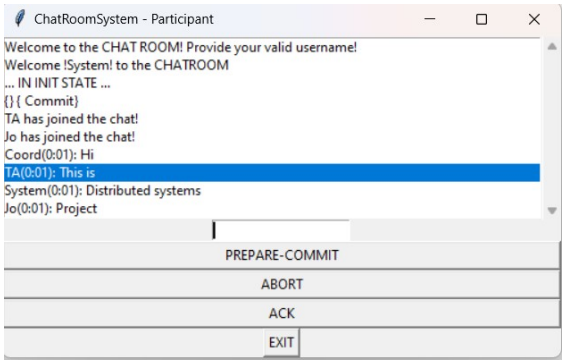
b. **Graphical User Interface (GUI)**: The tkinter-based GUI enhances user interaction and experience, providing an intuitive platform for users to view messages, input text, and navigate through the chatroom interface seamlessly. The GUI design promotes ease of use and accessibility, contributing to a positive user experience.

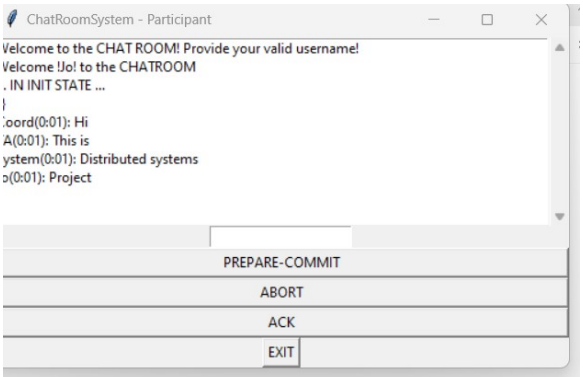
c**. Concurrent Client-Server Communication**: The system successfully manages multiple client connections concurrently, leveraging threading to handle concurrency effectively. Clients can connect to the server simultaneously and engage in communication without blocking the main server thread, ensuring smooth operation and responsiveness.

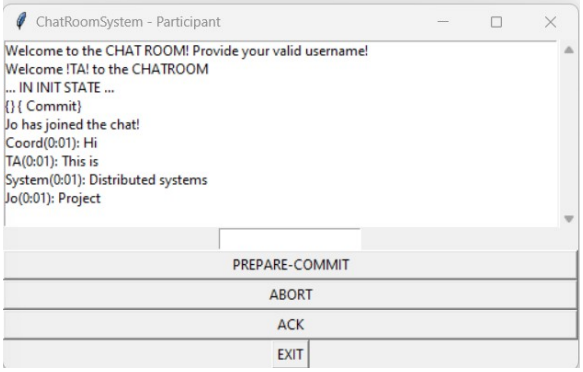
d. **Error Handling and Stability**: Robust error-handling mechanisms have been implemented to maintain system stability and reliability. The system gracefully manages network errors, disconnections, and unexpected exceptions, preventing disruptions in communication and ensuring uninterrupted service for users.


ChatRoomSystem - Server GUI
Listening...
Thread with ID 18256 has been assigned for the new client <socket.socket fd=892, family=2, type=1, proto=0, laddr=('127.0.0.1', 25000), raddr=('127.0.0.1', 49823)>:
Thread with ID 19332 has been assigned for the new client <socket.socket fd=912, family=2, type=1, proto=0, laddr=('127.0.0.1', 25000), raddr=('127.0.0.1', 49870)>:
Thread with ID 5492 has been assigned for the new client <socket.socket fd=940, family=2, type=1, proto=0, laddr=('127.0.0.1', 25000), raddr=('127.0.0.1', 49961)>:
Thread with ID 2512 has been assigned for the new client <socket.socket fd=964, family=2, type=1, proto=0, laddr=('127.0.0.1', 25000), raddr=('127.0.0.1', 49970)>:
Coord has joined the chat!
System has joined the chat!
TA has joined the chat!
Jo has joined the chat!
200 OK HTTP/1.1 200 OK Mon Apr 8 18:29:40 2024,Date: 2024-04-08 18:29:40.369068,Hi,Server: JetBrains Pycharm 2017,Content Length: 2


Co-ordinator GUI
Welcome to the CHAT ROOM! Provide your valid username!
Welcome !Coord! to the CHATROOM
... IN INIT state ...
System has joined the chat!
TA has joined the chat!
Jo has joined the chat!
Coord(0:01): Hi
TA(0:01): This is
System(0:01): Distributed systems
Jo(0:01): Project
SEND
EXIT
REQUEST


ChatRoomSystem - Participant
Welcome to the CHAT ROOM! Provide your valid username!
Welcome !System! to the CHATROOM
... IN INIT STATE ...
{} { Commit}
TA has joined the chat!
Jo has joined the chat!
Coord(0:01): Hi
TA(0:01): This is
System(0:01): Distributed systems
Jo(0:01): Project
PREPARE-COMMIT
ABORT
ACK
EXIT


ChatRoomSystem - Participant
PREPARE-COMMIT
ABORT
ACK
EXIT


ChatRoomSystem - Participant
Welcome to the CHAT ROOM! Provide your valid username!
Welcome !TA! to the CHATROOM
... IN INIT STATE ...
{} { Commit}
Jo has joined the chat!
Coord(0:01): Hi
TA(0:01): This is
System(0:01): Distributed systems
Jo(0:01): Project
PREPARE-COMMIT
ABORT
ACK
EXIT

2. **Three-Phase Commit Protocol:**

The implementation of the 3PC protocol demonstrates effective coordination and decision-making in a distributed environment, yielding the following results:

a. **Reliable Transaction Management**: The 3PC protocol ensures reliable transaction management across distributed participants, enabling coordinated commit or abort decisions based on consensus. Participants engage in a series of phases, including voting, pre-commit, and commit, to reach a unanimous decision regarding transaction outcome.

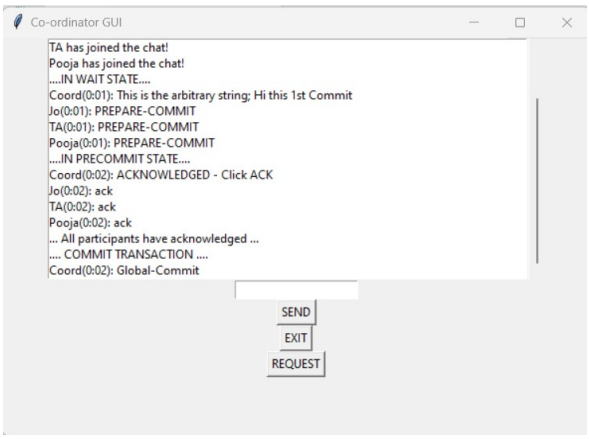
b. **Fault Tolerance and Resilience**: The protocol exhibits fault tolerance and resilience, allowing the system to recover from failures and ensure transaction consistency. Participants handle coordinator and participant failures gracefully, mitigating risks associated with crashes or network interruptions.

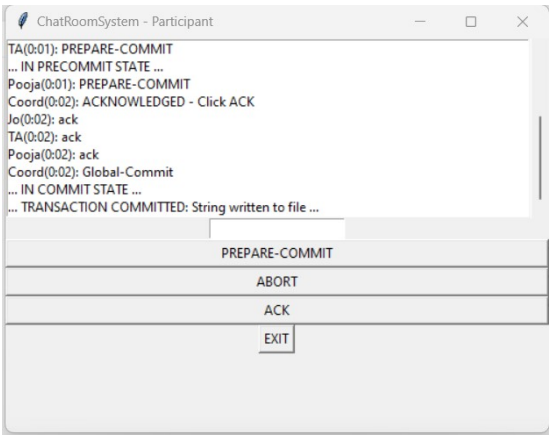
c. **Global Commit and Abort Scenarios**: The protocol effectively manages global commit and abort scenarios, ensuring that transactions are either successfully committed or aborted across all participants. Consistency is maintained, and transactional integrity is preserved, even in the presence of failures or timeouts.

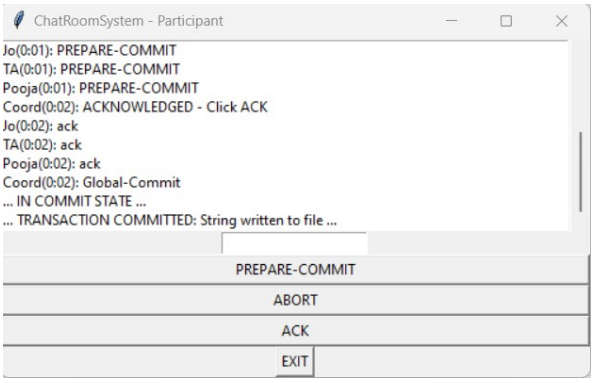
d. **Timeout Handling and Decision Making**: Timeout scenarios are handled effectively, enabling timely decision making and preventing indefinite delays in transaction processing. Participants adhere to predefined timeout intervals, ensuring that transactions progress smoothly and terminate within acceptable timeframes.

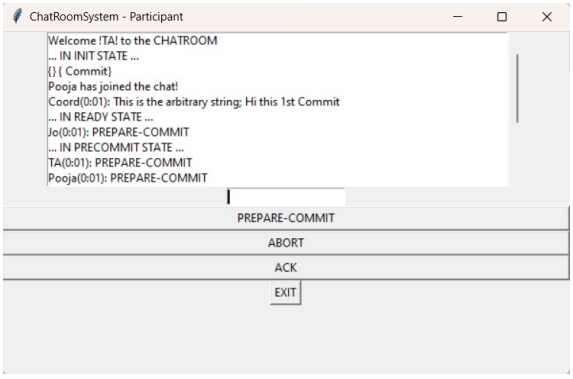
In summary, the implementation of both the distributed chatroom system and the Three-Phase Commit Protocol has resulted in robust, reliable, and efficient systems capable of facilitating communication and transaction management in distributed environments. The achieved results highlight the successful integration of networking concepts, GUI development, concurrency

management, and fault tolerance mechanisms, underscoring the effectiveness and versatility of the implemented solutions.









LIMITATIONS

While the distributed chatroom system presented here offers a robust platform for realtime communication among multiple users, it also has several limitations that may impact its performance, scalability, and usability. These limitations stem from various aspects of the system's design and implementation, including networking protocols, user interface, error handling, and concurrency management. Below are some of the key limitations of the distributed chatroom system:

1. **Scalability Challenges**:

One of the primary limitations of the chatroom system is its scalability. The system's design, particularly its reliance on a single server to manage all client connections and message distribution, may pose scalability challenges as the number of users increases.

With a large number of concurrent users, the server may become overwhelmed, leading to performance degradation and potential message delivery delays.

Additionally, the system's reliance on threading for managing multiple client connections may introduce scalability limitations, as the server's ability to handle a large number of threads simultaneously may be limited by hardware resources and operating system constraints.

2. **Lack of Persistence**:

Another significant limitation of the chatroom system is its lack of persistence. The system does not implement any form of message persistence or storage, meaning that messages sent by users are not saved once they leave the chatroom or if the server restarts.

As a result, users cannot access past messages or view message history, which may limit the usability of the chatroom for users who join late or want to refer back to previous conversations.

Lack of message persistence also means that users cannot resume conversations or catch up on missed messages after leaving and rejoining the chatroom.

3**. Limited Error Handling:**

The chatroom system's error handling capabilities are relatively limited, which may impact its reliability and robustness, particularly in the face of network failures, client disconnects, or unexpected exceptions.

The system does not implement sophisticated error recovery mechanisms or fault tolerance strategies, which may result in message loss, incomplete transactions, or server crashes in the event of network errors or server failures.

Additionally, the system's error messages and feedback to users may be insufficient or unclear, making it challenging for users to troubleshoot issues or understand the cause of errors.

4. **Security Concerns:**

The chatroom system does not incorporate robust security mechanisms, such as encryption, authentication, or access control, which may expose users to security risks, such as eavesdropping, data tampering, or unauthorized access.

Messages transmitted between clients and the server are sent in plaintext, making them vulnerable to interception and exploitation by malicious actors.

Lack of user authentication means that anyone with knowledge of the server's address and port number can connect to the chatroom and potentially impersonate other users or disrupt conversations.

Overall, while the distributed chatroom system provides a functional platform for realtime communication, its limitations in scalability, persistence, error handling, and security may hinder its usability and reliability in largescale or missioncritical applications. Addressing these limitations would require significant enhancements to the system's design, architecture, and implementation, including the implementation of scalable server architectures, message persistence mechanisms, robust error handling strategies, and comprehensive security measures.

FUTURE WORK

The distributed chatroom system presented in this project lays a solid foundation for further enhancements and future development. While the current implementation offers a functional and efficient communication platform, there are several areas where additional features and improvements could be incorporated to enhance its functionality, scalability, and user experience. Below are some suggestions for future work on this project:

1. **Integration of Three-Phase Distributed Commit Protocol:**
   One significant area for future enhancement is the integration of the Three-Phase Distributed Commit Protocol into the chatroom system. This protocol ensures atomicity and consistency in distributed transactions, making it ideal for managing chatroom operations such as message broadcasting and user management. By implementing this protocol, the system can ensure that messages are reliably delivered to all participants and that operations are either fully committed or fully aborted, preventing data inconsistencies and ensuring system integrity.

2. **Enhanced User Authentication and Authorization:**
   Currently, the system employs a basic user authentication mechanism to restrict access to authorized users. However, for improved security and user management, future work could focus on implementing more robust authentication and authorization mechanisms. This could include features such as user registration, password encryption, role-based access control, and session management. By enhancing user authentication, the system can better protect user accounts and prevent unauthorized access to the chatroom.

3. **Advanced Message Filtering and Moderation:**
   To ensure a positive and respectful chatroom environment, future work could involve implementing advanced message filtering and moderation features. This could include automatic detection and filtering of offensive or inappropriate language, spam detection, and moderation tools for chatroom administrators. Additionally, features such as message

threading, message editing, and message deletion could be implemented to give users more control over their conversations and content.

4. **Scalability and Performance Optimization:**
   As the number of users and messages in the chatroom grows, scalability and performance optimization become crucial considerations. Future work could focus on optimizing the system's architecture and implementing scalable solutions such as load balancing, distributed caching, and message partitioning. By improving scalability and performance, the system can handle larger user populations and higher message volumes without experiencing degradation in performance or responsiveness.

5. **Integration with External Services and APIs:**
   Another area for future enhancement is the integration of external services and APIs to extend the functionality of the chatroom system. For example, integration with social media platforms could enable users to share chatroom content externally, while integration with messaging APIs could allow for cross-platform communication with users on other messaging platforms. Additionally, integration with translation APIs could facilitate real-time translation of messages to support multilingual communication among users.

6. **User Interface Refinement and Accessibility:**
   Improving the user interface (UI) design and accessibility features of the chatroom system can enhance the overall user experience. Future work could involve UI refinements such as customizable themes, chatroom customization options, and support for multimedia content sharing. Additionally, accessibility features such as keyboard navigation, screen reader compatibility, and text-to-speech support could be implemented to ensure that the chatroom is accessible to users with disabilities.

7. **Enhanced Logging and Analytics:**
   Implementing comprehensive logging and analytics capabilities can provide valuable insights into user behavior, system performance, and security incidents. Future work could focus on integrating logging frameworks and analytics tools to capture and analyze chatroom activity,

user interactions, and system metrics in real-time. This data can then be used to identify usage patterns, detect anomalies, and optimize system performance and security.

In conclusion, the distributed chatroom system presented in this project offers a solid foundation for further development and enhancement. By incorporating features such as the Three-Phase Distributed Commit Protocol, advanced authentication and authorization mechanisms, message filtering and moderation tools, integration with external services, UI refinements, and logging and analytics capabilities, the system can be transformed into a robust, scalable, and feature-rich communication platform suitable for a wide range of applications and use cases.

REFERENCES

1. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm
2. https://www.daniweb.com/programming/software-development/threads/505937/python-tkinter-delay-hangs-using-after-method
3. http://www.bogotobogo.com/python/python_network_programming_tcp_server_client_chat_server_chat_client_select.php
4. https://wiki.python.org/moin/TkInter
5. https://pythonspot.com/tk-message-box/