

# Project Report

For

## Chat Application Implementation using Socket Programming

By

Navonil Ghosh (2105041)  
Debalina Bandyopadhyay (2105027)

06.11.2023

## **Abstract**

This paper explores the design and implementation of a chat application using connection-less Socket Programming for communication between two clients via a central server. connection-less sockets, exemplified by the User Datagram Protocol (UDP), provide an efficient means of transmitting data without the overhead of connection establishment, making them ideal for real-time applications like chat.

The study elucidates the architecture of the chat system, where clients communicate with one another through the intermediary server, showcasing the benefits of connection-less communication for instant messaging. It also addresses the challenges of implementing reliability and ordering of messages, which are inherent limitations of connection-less communication.

The research underscores the advantages and potential use cases of connection-less Socket Programming in chat applications, particularly in scenarios where low latency and reduced overhead are paramount. It highlights the need for error-handling mechanisms and message acknowledgement to ensure data integrity in a connection-less environment.

By offering practical insights into the implementation of a chat application using connection-less sockets, this study contributes to the understanding of network communication technologies and their applicability in real-time messaging systems. Developers and network engineers can leverage this knowledge to create efficient, low-latency chat applications tailored to specific use cases where traditional connection-oriented protocols may not be the optimal choice.

## **Introduction**

In an era marked by the ubiquity of digital communication, chat applications have emerged as a central pillar of our connected society. They empower individuals and organizations to engage in real-time conversations, fostering collaboration, sharing ideas, and bridging geographical boundaries. The architecture and underlying technologies of these chat systems play a pivotal role in delivering seamless communication experiences. Socket Programming, a fundamental networking technique, lies at the heart of these applications, allowing devices to exchange data over the Internet.

This paper embarks on a comprehensive exploration of chat application development, with a distinctive focus on leveraging connection-less Socket Programming. More specifically, it delves into the utilization of the User Datagram Protocol (UDP) to facilitate communication between two clients through an intermediary server.

connection-less Socket Programming offers an innovative approach to real-time messaging, placing a premium on minimal latency and reduced connection overhead. It departs from traditional connection-oriented protocols, making it an attractive choice for scenarios where immediate message delivery is of paramount importance.

This research immerses itself in the architectural intricacies, the inherent advantages, and the unique challenges that accompany the development of a chat system using connection-less sockets. It addresses the complexities associated with ensuring message reliability, maintaining proper message order, and handling errors within an environment that prioritizes connection-less communication.

By scrutinizing these facets, this study aims to present a practical perspective on the application of connection-less Socket Programming in chat application development. Its insights can guide developers and network engineers in the creation of efficient, low-latency messaging solutions tailored to specific use cases, ultimately shaping the landscape of real-time communication technology.

## **Methodology Used**

- i. **Client Initialization:** Both clients start by creating UDP sockets and binding them to their respective ports.
- ii. **Connection Establishment:** Each client sends a connection request message to the server, specifying the target client's IP address and port number.
- iii. **Server Routing:** The server receives the connection request messages from both clients and records the mapping between each client's socket and its target client's information.
- iv. **Message Exchange:** Clients send messages to the server, including the target client's identifier and the message content.

- v. **Server Forwarding:** The server receives messages from the clients, identifies the target client based on the message content, and forwards the message to the appropriate client's socket.
- vi. **Message Display:** The target client receives the forwarded message, extracts the message content, and displays it to the user.

## **Algorithms**

### **Server side:**

- i. Initialize server Socket, client1, client2, and an array to store connected clients.
- ii. Set up the server socket and bind it to a specified port (e.g., PORT).
- iii. Listen for incoming connections.
- iv. Accept the connection from the first client (client1).
- v. Accept the connection from the second client (client2).
- vi. Add client1 and client2 to the array of connected clients.
- vii. Initialize an empty message buffer.
- viii. Enter the main chat loop:
  - ix. a. Check for incoming messages from all connected clients.
  - x. b. If client1 sends a message, relay it to client2.
  - xi. c. If client2 sends a message, relay it to client1.
  - xii. d. If a client disconnects, remove it from the list of connected clients.
  - xiii. e. Continue listening for messages until both clients are disconnected.
- xiv. Close client1, client2, and the server socket when the chat is finished.
- xv. Terminate the server application.

### **Client Side:**

- i. Initialize the client Socket and server Addr.
- ii. Create a socket to connect to the server.
- iii. Specify the server's IP address and port number in server Addr.
- iv. Connect to the server using the client Socket.
- v. Initialize an empty message buffer.
- vi. Enter the main chat loop:
  - vii. a. Prompt the user to input a message.
  - viii. b. Read and store the user's input in the message buffer.

- ix. c. Send the message to the server using the client Socket.
- x. d. Receive messages from the server and display them.
- xi. e. Continue the loop until the user chooses to exit or disconnects.
- xii. Close the client Socket when the user exits the chat.
- xiii. Terminate the client application.

## Results

- i. The server waits for the two clients to establish a connection.

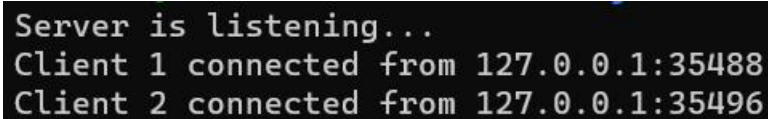


```
Server is listening...
```

A terminal window with a dark background and light-colored text showing the message "Server is listening...".

Fig 1. Server starts listening for incoming packets.

- ii. Connection Established between both the clients.

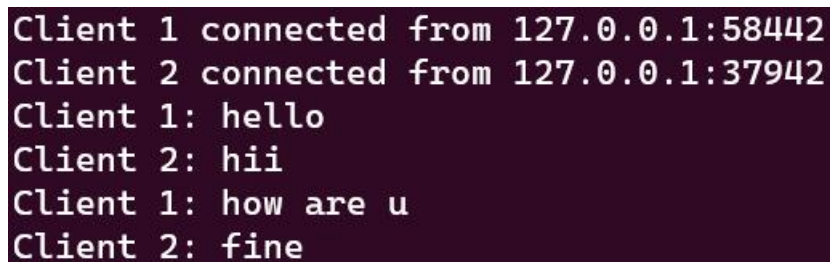


```
Server is listening...
Client 1 connected from 127.0.0.1:35488
Client 2 connected from 127.0.0.1:35496
```

A terminal window with a dark background and light-colored text showing the server listening and two clients connecting from 127.0.0.1 with ports 35488 and 35496.

Fig 2. Connection Established between the clients and the server.

- iii. After the Connection is established, normal text feature is enabled between the two clients allowing them to exchange texts between in one another.



```
Client 1 connected from 127.0.0.1:58442
Client 2 connected from 127.0.0.1:37942
Client 1: hello
Client 2: hii
Client 1: how are u
Client 2: fine
```

A terminal window with a dark background and light-colored text showing the server listening and two clients connecting. It then shows a conversation where Client 1 says "hello", Client 2 responds "hii", Client 1 says "how are u", and Client 2 responds "fine".

Fig 3. Normal Texting between the two clients is possible successful over the server.

## **Conclusion**

The exploration of connection-less Socket Programming, employing the User Datagram Protocol (UDP) for chat applications, emphasizes its efficient approach to real-time messaging with low latency and reduced connection overhead. This method excels when immediate message delivery is crucial, as in online gaming chat or financial trading. Yet, it presents reliability challenges that require careful handling.

Connection-less Socket Programming offers a valuable alternative for developers aiming to create efficient, low-latency messaging solutions tailored to specific use cases. Understanding its advantages and limitations can drive the advancement of real-time communication technology, allowing chat applications to prioritize speed and efficiency while maintaining data integrity.