University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 2 : Introduction to Client-Server Communication using Socket Programming (Simulating an ATM Machine Operation)

Submitted By:

Name:	Joty Saha
Roll No:	51
Name:	Ahona Rahman
Roll No:	59

Submitted On: February 1, 2024

Submitted To: Dr. Md. Abdur Razzaque
Md Mahmudur Rahman
Md. Ashraful Islam
Md. Fahim Arefin

1 Introduction

In the realm of computer networking, the establishment of robust communication channels between different entities is paramount. One of the fundamental paradigms in this domain is client-server communication, where clients request services or resources from servers, and servers respond accordingly. Socket programming serves as a cornerstone in facilitating this communication over networks by allowing processes to communicate with each other across a network.

1.1 Objectives

The primary objective of this experiment is to gain a comprehensive understanding of client-server communication using socket programming. Specifically, the experiment aims to achieve the following objectives:

- **Introduction to Socket Programming:** Familiarize with the concept of socket programming and its role in enabling communication between processes over a network.
- **Establishing TCP Connections:** Learn to create TCP connections between a server process running on one host and a client process running on another host.
- **Performing Operations via Client-Server Interaction:** Execute various operations initiated by the client and processed by the server, demonstrating bidirectional communication.
- **Implementing Basic Operations:** Implement basic operations such as converting text from capital letters to small letters and determining whether a given number is prime or a palindrome.
- **Understanding Idempotent Operations:** Explore the concept of idempotent operations in computing and distinguish them from non-idempotent operations.
- **Applying Exactly-One Semantics:** Implement non-idempotent operations with exactly-once semantics, ensuring reliable message delivery and error handling.
- **Designing an Application-Level Protocol:** Design a protocol for communication between an automatic teller machine (ATM) and a bank's centralized server, addressing authentication, balance inquiry, and fund withdrawal functionalities.
- **Enhancing Protocol Robustness:** Enhance the designed protocol to handle errors related to both request and response messages, ensuring fault tolerance and robustness in communication.

By achieving these objectives, participants will gain practical insights into client-server communication, socket programming, and the design considerations for developing robust network protocols. Additionally, they will grasp the significance of exactly-once semantics in ensuring reliable message delivery and error handling in distributed systems.

2 Theory

In the context of our experiment on client-server communication using socket programming, it's essential to understand several theoretical concepts that underpin the tasks and objectives outlined. These concepts include socket programming, TCP connections, idempotent operations, exactly-once semantics, and application-level protocol design.

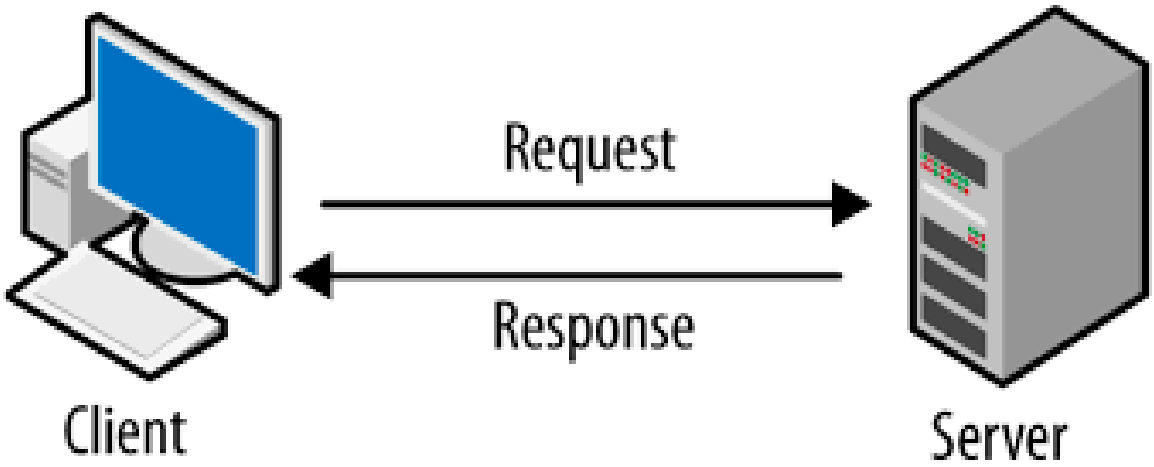

Figure 1: HyperText Transfer Protocol

2.1 Socket Programming:

Overview: Socket programming is a fundamental concept in network communication that enables processes to communicate over a network.

Key Aspects: It involves the creation of communication endpoints, known as sockets, through which data can be transmitted and received.

Use in Experiment: In our experiment, socket programming facilitates the establishment of communication channels between the client and server processes, enabling data exchange.

2.2 TCP Connections:

Overview: Transmission Control Protocol (TCP) is a widely used communication protocol that provides reliable, ordered, and error-checked delivery of data between applications.

Three-Way Handshake: TCP connections are established through a three-way handshake process, ensuring synchronization and reliability.

Use in Experiment: TCP connections are utilized in our experiment to create a reliable communication channel between the client and server, enabling seamless data exchange.

2.3 Idempotent Operations:

Definition: An idempotent operation is one that has the same effect regardless of how many times it is executed with the same input parameters.

Importance: Idempotent operations ensure consistency and reliability in distributed systems by eliminating the need for additional processing or checks when operations are repeated.

Use in Experiment: Understanding idempotent operations is crucial for designing reliable communication protocols and handling message delivery in our experiment.

2.4 Exactly-Once Semantics:

Definition: Exactly-once semantics guarantee that each message is delivered precisely once, without loss or duplication.

Reliability: This message delivery guarantee is essential for ensuring data integrity and consistency in distributed systems.

Use in Experiment: Exactly-once semantics are employed in our experiment to ensure reliable communication between the client and server, especially in scenarios involving non-idempotent operations.

2.5 Application-Level Protocol Design:

Protocol Definition: Application-level protocols define the rules and conventions for communication between applications or entities in a network.

Components: They specify message formats, sequencing of message exchanges, error handling mechanisms, and other communication aspects.

Use in Experiment: Protocol design is central to our experiment, particularly in the design of a protocol for ATM-bank server communication, which involves authentication, balance inquiry, fund withdrawal, and error handling functionalities.

By understanding these theoretical concepts, participants gain insights into the fundamental principles of network communication and protocol design, enabling them to effectively implement and analyze client-server systems using socket programming.

3 Methodology

3.1 Server

A server is a computer or system that provides resources, data, or services to other computers or clients over a network. Upon activation, the server listens for incoming client requests. When a request is received, it establishes a connection and processes the client's query. In this section, the server's tasks include converting lowercase letters to uppercase and determining whether a number is prime. Additionally, we implemented a Bank server capable of handling operations such as checking balance, cash deposit, and withdrawal, responding to valid requests and rejecting invalid ones.

3.2 Client

A client is a computer or system that requests and uses resources, data, or services provided by a server. In our setup, the client can be any web browser. To initiate communication, the client provides the server's IP address and port number, sending a request to the server. In the lab task, we modeled the client as an ATM booth machine, receiving and processing responses from the server.

3.3 Problem A

Establish a TCP connection in between a server process, running on host A and a client process, running on host B and then perform some operation by the server process requested by the client and send responses from the server.

1. Capital letter to Small letter conversion for a line of text
2. Send an integer and operation name (either 'prime' or 'palindrome') to the server and check whether it's a prime (or palindrome) or not

Listing 1: Server Code

```
import socket

def convert_to_lowercase(data):
    return data.lower()

def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

def is_palindrome(num):
    return str(num) == str(num)[::-1]

def handle_client_connection(client_socket):
    request = client_socket.recv(1024).decode()
    request_data = request.split()

```

```
    if request_data[0] == 'convert':
        response = convert_to_lowercase(request_data[1])
    elif request_data[0] == 'check':
        num = int(request_data[1])
        operation = request_data[2]
        if operation == 'prime':
            response = 'Prime' if is_prime(num) else 'Not
               Prime'
        elif operation == 'palindrome':
            response = 'Palindrome' if is_palindrome(num)
               else 'Not Palindrome'
    else:
        response = 'Invalid request'

    client_socket.send(response.encode())
    client_socket.close()

def main():
    server_socket = socket.socket(socket.AF_INET, socket.
       SOCK_STREAM)
    server_socket.bind(('localhost', 1234))
    server_socket.listen(5)

    print("Server listening on port 1234...")

    while True:
        client_socket, addr = server_socket.accept()
        print(f"Accepted connection from {addr[0]}:{addr[1]}"
           )
        handle_client_connection(client_socket)

if __name__ == "__main__":
    main()
```

Listing 2: Client Code

```
import socket


def send_request(request):
    client_socket = socket.socket(socket.AF_INET, socket.
       SOCK_STREAM)
    client_socket.connect(('localhost', 1234))
    client_socket.send(request.encode())
    response = client_socket.recv(1024).decode()
    print("Response:", response)
    client_socket.close()


```

```
def main():
    while True:
        print("\nMenu:")
        print("1. Convert to lowercase")
        print("2. Check prime or palindrome")
        print("3. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            data = input("Enter text to convert to lowercase:
                ")
            send_request(f"convert {data}")
        elif choice == '2':
            num = input("Enter number: ")
            operation = input("Enter operation (prime/
               palindrome): ")
            send_request(f"check {num} {operation}")
        elif choice == '3':
            break
        else:
            print("Invalid choice. Please try again.")


if __name__ == "__main__":
    main()
```

3.4 Problem B

Using the above connection, design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.

1. Design and describe an application-level protocol to be used between an automatic teller machine and a bank's centralized server. Your protocol should allow a user's card and password to be verified, the account balance (which is maintained at the centralized computer) to be queried, and an account withdrawal to be made (that is, money disbursed to the user). Your protocol entities should be able to handle the all-too-common cases in which there is not enough money in the account to cover the withdrawal. Specify your protocol by listing the messages exchanged and the action taken by the automatic teller machine or the bank's centralized computer on transmission and receipt of messages. Sketch the operation of your protocol for the case of a simple withdrawal with no errors.

2. Enhance the above protocol so that it can handle errors related to both request and response messages to and from the server.

Listing 3: Bank Server Code

```
import random
```

```
import socket
import time

# Dummy database for card and balance information
database = {
    "123456789": {"name": "Joty", "password": "1234", "balance": 10000},
    "987654321": {"name": "Ahona", "password": "5678", "balance": 5000}
}

def verify_card_and_password(card_number, password):
    if card_number in database and database[card_number]["password"] == password:
        return True
    else:
        return False

def get_balance(card_number):
    return database[card_number]["balance"]

def withdraw_money(card_number, amount):
    if amount <= 0:
        return False, "WITHDRAW_ACK FAILURE Invalid amount"

    if database[card_number]["balance"] >= amount:
        database[card_number]["balance"] -= amount
        return True, f"WITHDRAW_ACK SUCCESS Withdrawn {amount} successfully"
    else:
        return False, "WITHDRAW_ACK FAILURE Insufficient funds"

def create_account(card_number, name, password, initial_balance):
    if card_number not in database:
        database[card_number] = {"name": name, "password": password, "balance": initial_balance}
        return True, "CREATE ACCOUNT SUCCESSFULLY"
    else:
        return False, "FAILS TO CREATE ACCOUNT"

def deposit_money(card_number, amount):
    if amount <= 0:
        return False, "DEPOSITE_ACK FAILURE Invalid amount"

    database[card_number]["balance"] += amount
    return True, "DEPOSIT_ACK SUCCESS Amount deposited successfully"

```

```
44 def random_error(error_probability):
45     return random.random() <= error_probability
46
47 def handle_client_connection(client_socket):
48     request = client_socket.recv(1024).decode()
49     request_data = request.split()
50     response = ""
51
52     if random_error(0.3):  # Adjust the probability as needed
53         response = "ERROR_ACK Random error occurred"
54     else:
55         if request_data[0] == "VERIFY_CARD_AND_PASSWORD":
56             card_number, password = request_data[1],
                request_data[2]
57             if verify_card_and_password(card_number, password
                ):
58                 response = "VERIFY_ACK SUCCESS"
59             else:
60                 response = "VERIFY_ACK FAILURE"
61
62         elif request_data[0] == "BALANCE_INQUIRY":
63             card_number = request_data[1]
64             balance = get_balance(card_number)
65             response = f"BALANCE_ACK {balance}"
66
67
68         elif request_data[0] == "WITHDRAW":
69             card_number, amount = request_data[1], int(
                request_data[2])
70             success, message = withdraw_money(card_number,
                amount)
71             response = message if success else message
72
73
74         elif request_data[0] == "CREATE_ACCOUNT":
75             card_number, name, password, initial_balance =
                request_data[1], request_data[2], request_data
                [3], int(request_data[4])
76             if create_account(card_number, name, password,
                initial_balance):
77                 response = "CREATE_ACCOUNT_ACK SUCCESS
                    Account created successfully"
78             else:
79                 response = "CREATE_ACCOUNT_ACK FAILURE
                    Account already exists"
80
81         elif request_data[0] == "DEPOSIT":
82             card_number, amount = request_data[1], int(
                request_data[2])
83             success, message = deposit_money(card_number,
```

```
                amount)
84              response = message if success else message
85
86      client_socket.send(response.encode())
87      client_socket.close()
88
89  def main():
90      server_socket = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)
91      server_socket.bind(('localhost', 1234))
92      server_socket.listen(5)
93
94      print("Bank server listening on port 1234...")
95
96      while True:
97          client_socket, addr = server_socket.accept()
98          print(f"Accepted connection from ATM booth: {addr
                [0]}:{addr[1]}")
99          handle_client_connection(client_socket)
100
101 if __name__ == "__main__":
102     main()
```

Listing 4: Atm Client Code

```
1   import socket
2   import random
3
4   def random_error(error_rate):
5       return random.randint(1, 100) <= error_rate
6
7   def send_request(request):
8       client_socket = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)
9       client_socket.connect(('localhost', 1234))
10
11      if random_error(30):  # Adjust the error rate as needed
            (10% error rate in this example)
12          print("Simulating random error.")
13          response = "ERROR"
14      else:
15          client_socket.send(request.encode())
16          response = client_socket.recv(1024).decode()
17
18      print("Response from bank server:", response)
19      client_socket.close()
20
21  def main():
22      while True:
```

```
        print("\nMenu:")
        print("1. Verify Card and Password")
        print("2. Balance Inquiry")
        print("3. Withdraw Money")
        print("4. Create Account")
        print("5. Deposit Money")
        print("6. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            card_number = input("Enter card number: ")
            password = input("Enter password: ")
            send_request(f"VERIFY_CARD_AND_PASSWORD {card_number} {password}")
        elif choice == '2':
            card_number = input("Enter card number: ")
            send_request(f"BALANCE_INQUIRY {card_number}")
        elif choice == '3':
            card_number = input("Enter card number: ")
            amount = input("Enter amount to withdraw: ")
            send_request(f"WITHDRAW {card_number} {amount}")
        elif choice == '4':
            card_number = input("Enter new card number: ")
            name = input("Enter your name: ")
            password = input("Enter password for new account: ")
            initial_balance = input("Enter initial balance: ")
            send_request(f"CREATE_ACCOUNT {card_number} {name} {password} {initial_balance}")
        elif choice == '5':
            card_number = input("Enter card number: ")
            amount = input("Enter amount to deposit: ")
            send_request(f"DEPOSIT {card_number} {amount}")
        elif choice == '6':
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

4 Experimental result

Some Snapshots of the Client Side queries can be seen in the following figures:

4.1 Problem A

4.1.1 After Running Server Code:

```
/home/hp/PycharmProjects/pythonProject/.venv/bin/python /home/hp/PycharmProjects/pythonProject/server.py
Server listening on port 1234...
```

Figure 2: Content of Server

4.1.2 After Running Client Code:

```
/home/hp/PycharmProjects/pythonProject/.venv/bin/python /home/hp/PycharmProjects/pythonProject/client.py

Menu:
1. Convert to lowercase
2. Check prime or palindrome
3. Exit
Enter choice:
```

Figure 3: Server Connect to Client

4.1.3 Testing Client-Server Connection1:

Convert to Lowercase

```
Menu:
1. Convert to lowercase
2. Check prime or palindrome
3. Exit
Enter choice: 1
Enter text to convert to lowercase: JOTY
Response: joty
```

Figure 4: Successfully Convert to Lowercase

4.1.4 Testing Client-Server Connection2:

Detect Prime Number

```
Menu:
1. Convert to lowercase
2. Check prime or palindrome
3. Exit
Enter choice: 2
Enter number: 5
Enter operation (prime/palindrome): prime
Response: Prime
```

Figure 5: Successfully Detect any prime number

4.1.5 Testing Client-Server Connection3:

Detect Palindrome Number

```
Menu:
1. Convert to lowercase
2. Check prime or palindrome
3. Exit
Enter choice: 2
Enter number: 55
Enter operation (prime/palindrome): palindrome
Response: Palindrome
```

Figure 6: Successfully Detect any palindrome number

4.2 Problem B

4.2.1 After Running Bank Server Code:

```
/home/hp/PycharmProjects/pythonProject/.venv/bin/python /home/hp/PycharmProjects/pythonProject/bank_server.p
Bank server listening on port 1234...
```

Figure 7: Content of Bank Server

4.2.2 After atm Client Code:

```
/home/hp/PycharmProjects/pythonProject/.venv/bin/python /home/hp/PycharmProjects/pythonProject/atm_client.p

Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice:
```

Figure 8: Atm Client Successfully Connect to Bank Server

4.2.3 Verify Card and Password:

This option allows users to verify their card details and password to authenticate themselves.

```
/home/hp/PycharmProjects/pythonProject/.venv/bin/python /home/hp/PycharmProjects/pythonProject/atm_client.p

Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 1
Enter card number: 123456789
Enter password: 1234
Response from bank server: VERIFY_ACK SUCCESS
```

Figure 9: Successfully Verify existing data

4.2.4 Balance Inquiry:

With this option, users can check the current balance of their bank account.

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 2
Enter card number: 123456789
Response from bank server: BALANCE_ACK 10000
```

Figure 10: Successfully Check Balance

4.2.5 Withdraw Money:

This option enables users to withdraw money from their bank account.

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 3
Enter card number: 123456789
Enter amount to withdraw: 1000
Response from bank server: WITHDRAW_ACK SUCCESS Withdrawn 1000 successfully
```

Figure 11: Successfully withdraw money from bank account

4.2.6 Check Balance after Withdraw Money:

Account Balance is reduced after money withdrawal.

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 2
Enter card number: 123456789
Response from bank server: BALANCE_ACK 9000
```

Figure 12: Withdrawl of money is reduced the account balance

4.2.7 Create Account:

Users can utilize this option to create a new bank account.

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 4
Enter new card number: 78932541
Enter your name: Oyshi
Enter password for new account: 1234
Enter initial balance: 1000
Response from bank server: CREATE_ACCOUNT_ACK SUCCESS Account created successfully
```

Figure 13: Create account Successfully

4.2.8 Deposit Money:

This option allows users to deposit money into their bank account.

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 5
Enter card number: 123456789
Enter amount to deposit: 500
Response from bank server: DEPOSIT_ACK SUCCESS Amount deposited successfully
```

Figure 14: Successfully deposit money to bank account

4.2.9 Insufficient Funds:

If a user want to withdraw more money than has in his account, then this message will be send form the server.

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 3
Enter card number: 123456789
Enter amount to withdraw: 10000
Response from bank server: WITHDRAW_ACK FAILURE Insufficient funds
```

Figure 15: Insufficient Funds

4.3 Errors

If a user puts an request for cash withdrawal, the request is saved using the request id ,user name, and amount. If for any circumstances the request process is failed because of re appearing of request, the server reject the request of the client. So , if a client resend requests by mistake or for any kind off network problem, if the money is withdrawn with the same request, the server will reject all the other requests.

4.3.1 Random Error in balance checking:

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 2
Enter card number: 123456789
Response from bank server: ERROR_ACK Random error occurred
```

Figure 16: Random error in balance checking

4.3.2 Random Error in creating account:

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 4
Enter new card number: 789456123
Enter your name: Arnob
Enter password for new account: 7894
Enter initial balance: 50000
Simulating random error.
Response from bank server: ERROR
```

Figure 17: Random Error in creating account

4.3.3 Negative Deposit Error:

If a user want to withdraw less than 00 from his account, then this message will be send form the server.

```
Menu:
1. Verify Card and Password
2. Balance Inquiry
3. Withdraw Money
4. Create Account
5. Deposit Money
6. Exit
Enter choice: 5
Enter card number: 123456789
Enter amount to deposit: -300
Response from bank server: ERROR_ACK Random error occurred
```

Figure 18: Negative Deposit Error

4.4 Graph

4.4.1 Server Side Error Graph:

Figure 19: Server Side Error

4.4.2 Client Side Error Graph:

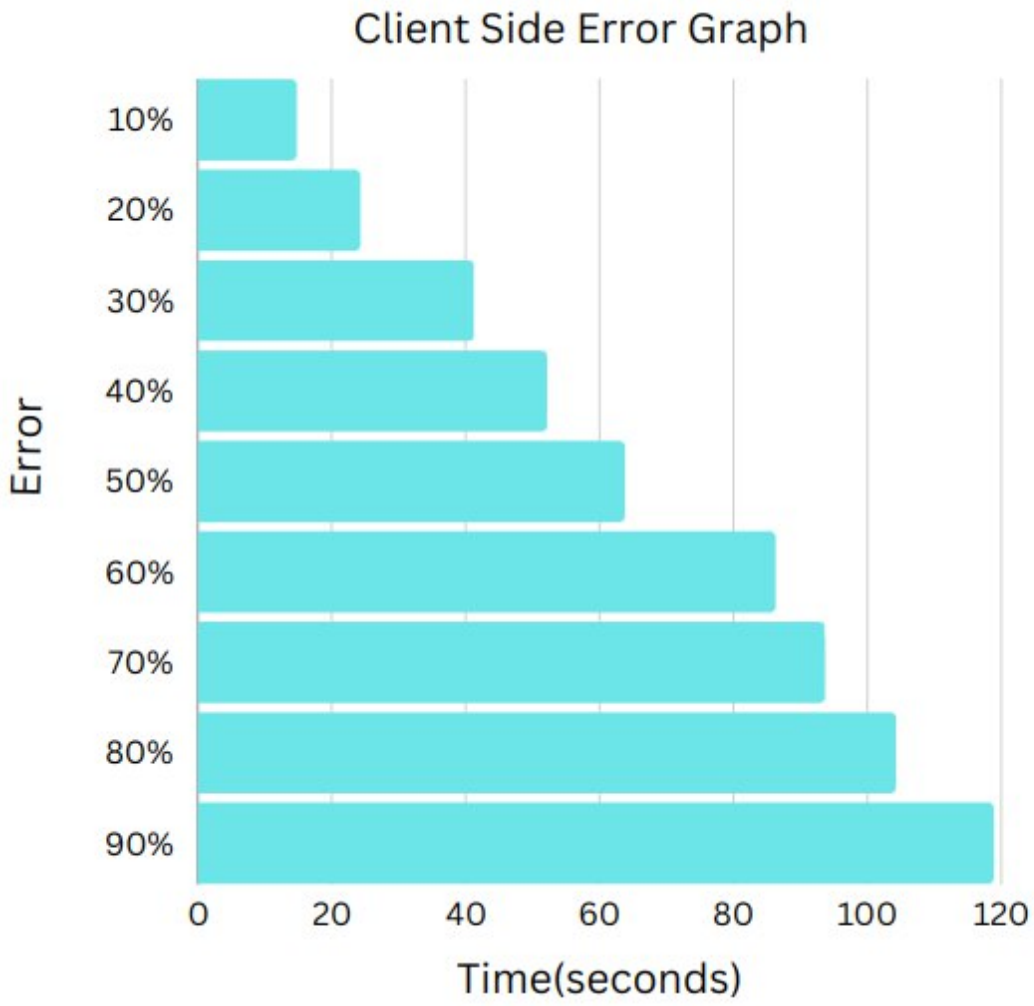

Figure 20: Client Side Error

5 Experience

1. We had a wonderful experience with how servers and clients work in real life.
2. Experienced how socket programming work and establish connection between client and server.
3. Explored socket programming fundamentals for network communication.
4. Implemented TCP connections, understanding reliability and handshaking.
5. Learned about idempotent operations and exactly-once semantics.
6. Faced challenges debugging connectivity and message errors.
7. Gained insights into network protocol design and implementation.

References

[1] Socket programming in Python. GeeksforGeeks, June 20, 2017. [Online; accessed 2023-01-25].

[2] Pankaj. Python socket programming - Server, client example. DigitalOcean, August 3, 2022. [Online; accessed 2023-01-25].

[3] Real Python. Socket programming in Python (guide). Real Python, February 21, 2022. [Online; accessed 2023-01-25].