University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 5: Implementation of flow control and reliable data transfer through management of timeout, fast retransmit, cumulative acknowledgment, loss of data and acknowledgment packets.

Submitted By :	**Submitted To :**
Name: Md. Emon Khan	Dr. Md. Abdur Razzaque
Roll No : 30	Dr. Md. Mamun Or Rashid
Name: Mahmudul Hasan	Dr. Muhammad Ibrahim
Roll No : 60	Md. Redwan Ahmed Rizvee

Submitted On : February 22, 2024

Contents

1 Introduction **2**

1.1 Objectives 2

2 Theory **2**

2.1 Flow Control 3

2.2 Congestion Control 3

3 Methodology **3**

3.1 Task1: Implementaion of TCP Flow Control 3

3.1.1 Client: 3

3.1.2 Server: 4

4 Experimental result **5**

4.1 Task 1: Implementaion of TCP Flow Control 5

5 Experience **9**

1 Introduction

The primary goal of Lab-5 is to investigate flow control and learn about the reliable data transfer mechanism. The experiment focuses on two key tasks: TCP flow control and reliable data transfer. In Task 1, we configure the server to implement TCP flow control by setting the receive window size, which dictates the amount of data the receiver can accept before sending an acknowledgment. Clients are configured to utilize Cumulative Acknowledgment, where the receiver sends acknowledgments for the highest sequence number received, ensuring the orderly reception of data packets. In Task 2, we calculate SampleRTT values and utilize the Exponentially Weighted Moving Average (EWMA) equation to estimate Round-Trip Time (RTT) and calculate retransmission timeouts (RTO). Finally, we analyze the captured network traffic to evaluate the performance of TCP flow control.

1.1 Objectives

Some of the preliminary objectives of this lab experiment are:

- To gather knowledge about how TCP controls the flow of data between a sender and a receiver
- To learn how TCP implements reliable data transfer and cumulative acknowledgment

2 Theory

TCP is a transport layer protocol that provides reliable, ordered, and error-checked delivery of a stream of bytes between two applications running on the same network connection. TCP involves one server and one client. Before a connection is established, the server must be listening for connection requests. TCP employs a three-way handshake (active open), retransmission, and error detection mechanisms to enhance reliability. Additionally, TCP maintains various operations to establish perfect communication between the server and client, including connection management, error detection, error recovery, congestion control, connection termination, flow control, etc.

There are mainly two types of TCP flow control: feedback-based control and rate-based flow control.

In feedback-based flow control, there are mainly two protocols: sliding window and stop-and-wait protocol.

2.1 Flow Control

Flow control is a process that regulates the data transmission rate between two nodes (Sender and Receiver). If the sender's data transmission rate is faster than the receiving rate, the receiver may be unable to get and process data. This situation arises due to high traffic load and low processing power. Flow control ensures that a sender does not overwhelm a receiver with too much data too quickly. The goal of flow control is to prevent buffer overflow, which can lead to dropped packets and poor network performance.

2.2 Congestion Control

Congestion occurs when a sender sends too much data over a network, which means there is an availability of many packets in the network. As a result, the network becomes overloaded, leading to dropped packets and poor network performance, as well as delays in packet delivery. To prevent this situation, congestion control is used. Congestion on the network may be significantly decreased by minimizing the load on the network by the transport layer. Congestion control is the responsibility of the transport and the network layer.

3 Methodology

3.1 Task1: Implementaion of TCP Flow Control

3.1.1 Client:

- In the client side we create a data from a PDF file for testing
- We receive the window size specified by the server and then send the data dispersed into a lot of chunks consisting of different packet sequences
- Upon receiving each chunk of data, the server sends an acknowledgement
- If the acknowledgment matches the sequence we don't need to resend
- If it doesn't, then we need to resend the data
- We send a dummy data of length 0 to tell the server that the data transmission has ended

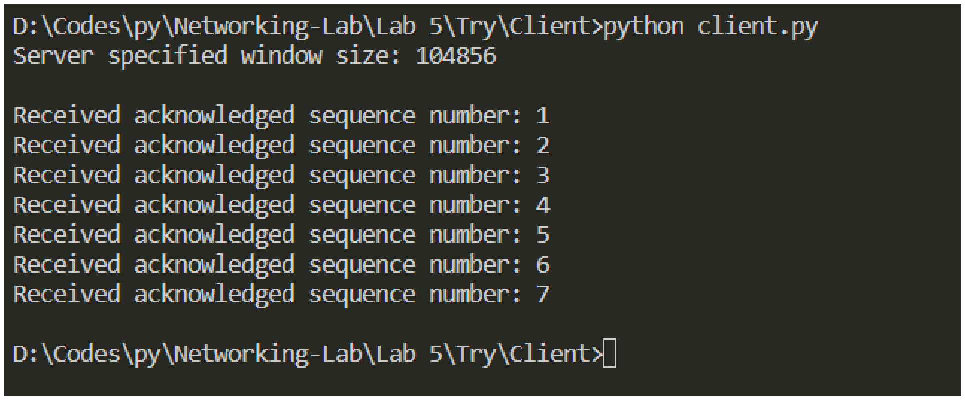
3.1.2 Server:

- In the server side, we first specify our window size
- Then we expect a sequence of data and wait for the client to send the expected sequence
- If the sent data is the expected sequence, then we send an acknowledgement
- Otherwise, we send the the acknowledgement of the last received data packet sequence
- Upon receiving a dummy data of 0 length, the transmission ends

4 Experimental result

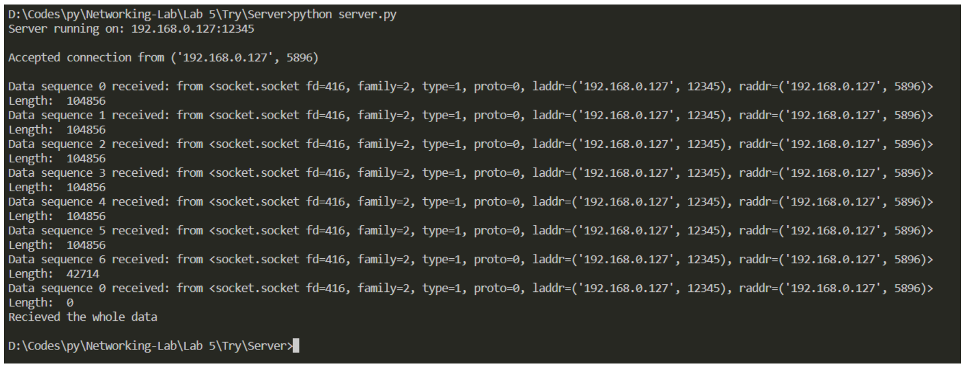
Some Snapshots of the terminal output for each of these tools.

4.1 Task 1: Implementaion of TCP Flow Control



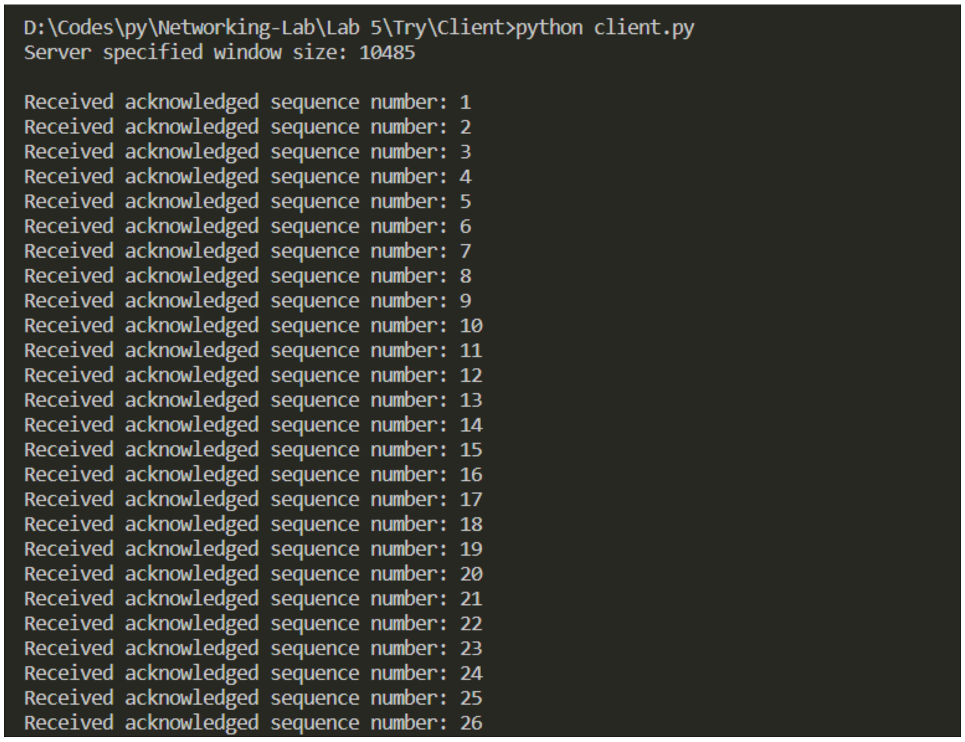
```
D:\Codes\py\Networking-Lab\Lab 5\Try\Client>python client.py
Server specified window size: 104856

Received acknowledged sequence number: 1
Received acknowledged sequence number: 2
Received acknowledged sequence number: 3
Received acknowledged sequence number: 4
Received acknowledged sequence number: 5
Received acknowledged sequence number: 6
Received acknowledged sequence number: 7

D:\Codes\py\Networking-Lab\Lab 5\Try\Client>
```

Figure 1: Terminal Output for Client When Window Size is Big



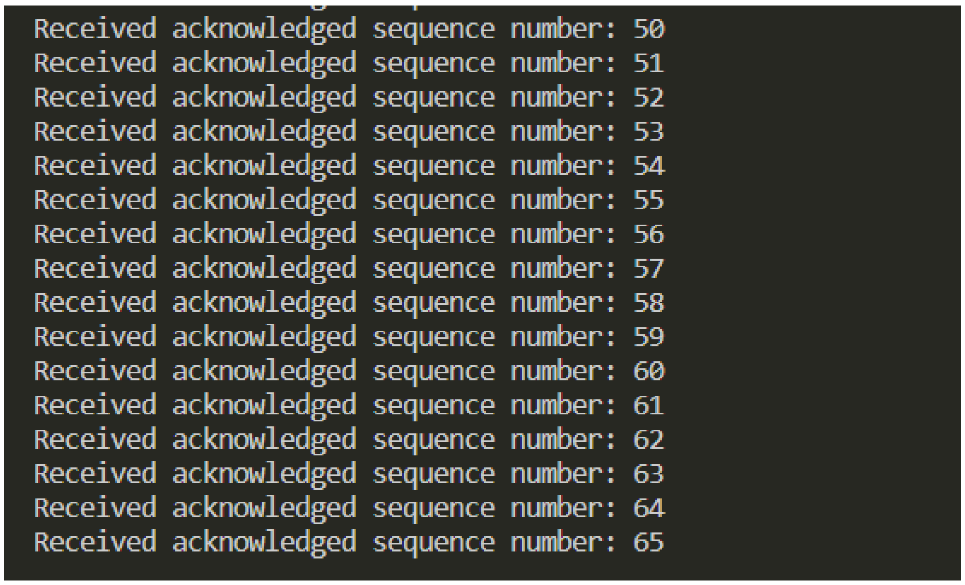
```
D:\Codes\py\Networking-Lab\Lab 5\Try\Server>python server.py
Server running on: 192.168.0.127:12345

Accepted connection from ('192.168.0.127', 5896)

Data sequence 0 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  104856
Data sequence 1 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  104856
Data sequence 2 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  104856
Data sequence 3 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  104856
Data sequence 4 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  104856
Data sequence 5 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  104856
Data sequence 6 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  42714
Data sequence 0 received: from <socket.socket fd=416, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5896)>
Length:  0
Recieved the whole data

D:\Codes\py\Networking-Lab\Lab 5\Try\Server>
```

Figure 2: Terminal Output for Server When Window Size is Big



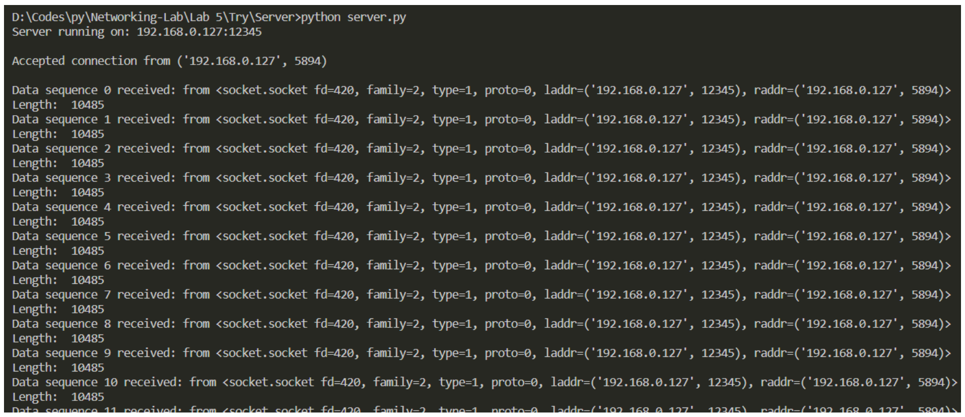
```
D:\Codes\py\Networking-Lab\Lab 5\Try\Client>python client.py
Server specified window size: 10485

Received acknowledged sequence number: 1
Received acknowledged sequence number: 2
Received acknowledged sequence number: 3
Received acknowledged sequence number: 4
Received acknowledged sequence number: 5
Received acknowledged sequence number: 6
Received acknowledged sequence number: 7
Received acknowledged sequence number: 8
Received acknowledged sequence number: 9
Received acknowledged sequence number: 10
Received acknowledged sequence number: 11
Received acknowledged sequence number: 12
Received acknowledged sequence number: 13
Received acknowledged sequence number: 14
Received acknowledged sequence number: 15
Received acknowledged sequence number: 16
Received acknowledged sequence number: 17
Received acknowledged sequence number: 18
Received acknowledged sequence number: 19
Received acknowledged sequence number: 20
Received acknowledged sequence number: 21
Received acknowledged sequence number: 22
Received acknowledged sequence number: 23
Received acknowledged sequence number: 24
Received acknowledged sequence number: 25
Received acknowledged sequence number: 26
```

Figure 3: Terminal Output for Client When Window Size is Small



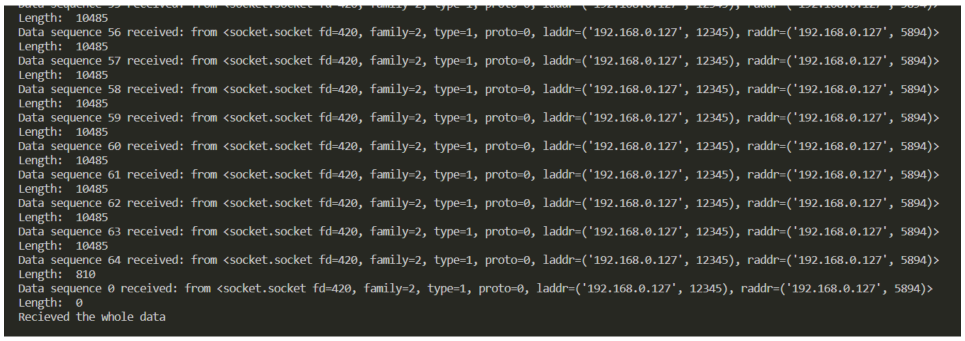
```
Received acknowledged sequence number: 50
Received acknowledged sequence number: 51
Received acknowledged sequence number: 52
Received acknowledged sequence number: 53
Received acknowledged sequence number: 54
Received acknowledged sequence number: 55
Received acknowledged sequence number: 56
Received acknowledged sequence number: 57
Received acknowledged sequence number: 58
Received acknowledged sequence number: 59
Received acknowledged sequence number: 60
Received acknowledged sequence number: 61
Received acknowledged sequence number: 62
Received acknowledged sequence number: 63
Received acknowledged sequence number: 64
Received acknowledged sequence number: 65
```

Figure 4: Terminal Output for Client When Window Size is Small



```
D:\Codes\py\Networking-Lab\Lab 5\Try\Server>python server.py
Server running on: 192.168.0.127:12345

Accepted connection from ('192.168.0.127', 5894)

Data sequence 0 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 1 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 2 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 3 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 4 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 5 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 6 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 7 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 8 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 9 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 10 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 11 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
```

Figure 5: Terminal Output for Server When Window Size is Small



```
Data sequence 55 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 56 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 57 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 58 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 59 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 60 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 61 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 62 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 63 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  10485
Data sequence 64 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  810
Data sequence 0 received: from <socket.socket fd=420, family=2, type=1, proto=0, laddr=('192.168.0.127', 12345), raddr=('192.168.0.127', 5894)>
Length:  0
Recieved the whole data
```

Figure 6: Terminal Output for Server When Window Size is Small

5 Experience

- Learned how TCP regulates the flow of data between sender and receiver, ensuring efficient utilization of network resources and preventing overwhelming the receiver with excessive data.
- Understand TCP's resilience in recovering from packet loss and ensuring data integrity.
- Deeper understanding of of TCP's operation and its implications for network performance.
- We had to learn the use of a new library "struct"

References

[1] `https://www.geeksforgeeks.org/`

[2] `https://www.javatpoint.com/`