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:

Name : Abdullah Al Mahmud

Roll No : 15

Name : Zisan Mahmud

Roll No : 23

Submitted On:

February 21, 2024

Submitted To:

Dr. Md. Abdur Razzaque

Dr. Md Mamunur Rashid

Dr. Muhammad Ibrahim

Mr. Md. Redwan Ahmed Rizvee

1 Introduction

In this lab our main goal is to simulate how TCP ensures flow control and reliable data transfer in transport layer and find out what are the functions and features of transport layer protocol. TCP is a well established transport layer protocol. By implementing the flow control and reliable data transfer through management of timeout, fast retransmit, cumulative acknowledgment, loss of data and acknowledgment packets, we will understand the functionality and complexity of TCP protocol.

2 Objectives

The preliminary objective of this lab is to emulate the workflow of TCP in transport layer. In this lab, we will:

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

3 Theory

TCP is one of the protocols of the transport layer for network communication. TCP provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating via an IP network. TCP is connection-oriented, and a connection between client and server is established before data can be sent. The server must be listening (passive open) for connection requests from clients before a connection is established. Three-way handshake (active open), retransmission, and error-detection adds to reliability. Thus TCP can maintain various operations to establish perfect communications between a pair of hosts, e.g connection management, error detection, error recovery, congestion control, connection termination, flow control, etc. In this lab, we will have a look at the flow control mechanism and reliable data transfer mechanisms of the TCP protocol.
TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the receive window field the amount of additionally received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait

for an acknowledgement and window update from the receiving host.This acknowledgement is known as cumulative acknowledgement that means the receiver sends an acknowledgment for the highest sequence number it has received in order, and it assumes that all packets up to that sequence number have been received.

Advantages of flow control:

1. Flow control prevents buffer overflow by regulating the sending and receiving data rate.
2. Helps in handling different data rates.
3. By avoiding packet loss, flow control helps in efficient use of network resources.

Disadvantages of flow control:

1. Flow control may cause delay in sending and receiving data.
2. May not be effective in congested networks.
3. May require additional hardware or software to implement the flow control.

4 Methodology

Task 1: Implement TCP Flow Control

1. At first, we configure the server to use TCP flow control.
2. We set the receive window size which determines how much data the receiver is willing to accept before sending an acknowledgment.
3. Then we configure the clients to use cumulative acknowledgment.
4. Then we test the TCP flow control by sending data from sender to receiver.

Task 2: Implement reliable data transfer

1. We start a timer after sending data packets and calculate SampleRTT for each of the packets.

2. In client code, we implement the EWMA equation to calculate the TimeOUt value. The TimeOut value is used to retransmit a data packet that is sent but not received by the receiver. The EWMA equation is used to estimate the round-trip time (RTT) and calculate the retransmission timeout (RTO) value.

3. Another way is **Fast Retransmit** to handle the unsuccessful data transfer. We implement this fast retransmit that is if three duplicate acknowledgments for the same packet is received, then the packet is retransmitted before the TimeOut

4. We test this by sending data from sender to receiver.

Task 3: Analyze Results

1. We analyze the captured network traffic to evaluate the performance of the TCP flow control and reliability control algorithms.

2. Finally,. we compare the results under different network conditions and analyze how the algorithms affect the network

5 Experimental result

5.1 Task 1: Implement TCP Flow Control

5.1.1 Sender:

When a connection is established, the sender receives the buffer window size of the receiver side. Then the sender starts to send data and wait for an acknowledgement from the receiver. The acknowledgement will contain the sequence number of the next byte the receiver is willing to accept.

```
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task1$ python3 sender.py
Server is listening for incoming connections
Accepted connection from ('127.0.0.1', 49244)
size:  1470
Sent packet 1470
Received acknowledgment for packet 1470
size:  1470
Sent packet 2940
Received acknowledgment for packet 2940
size:  1470
Sent packet 4410
Received acknowledgment for packet 4410
size:  1470
Sent packet 5880
Received acknowledgment for packet 5880
size:  1470
Sent packet 7350
Received acknowledgment for packet 7350
```

Figure 1: Sender content for task 1

```
size:  1470
Sent packet 196980
Received acknowledgment for packet 196980
size:  1470
Sent packet 198450
Received acknowledgment for packet 198450
size:  1470
Sent packet 199920
Received acknowledgment for packet 199920
size:  1079
Sent packet 200999
file sent
Throughput: 8890.410006601496 B/s
Done
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task1$
```

Figure 2: Sender content for task 1

5.1.2 Receiver:

When the connection is established the receiver sends acknowledgement that contains the buffer window size to the sender. The window size defines how much data receiver can accommodate without overflow. Also the acknowledgement contains the sequence number of the next byte.

```
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task1$ python3 receiver.py
Connected to server
Received packet with seq 1470
Received 1470 and wrote to file
Received packet with seq 2940
Received 2940 and wrote to file
Received packet with seq 4410
Received 4410 and wrote to file
Received packet with seq 5880
Received 5880 and wrote to file
Received packet with seq 7350
Received 7350 and wrote to file
Received packet with seq 8820
Received 8820 and wrote to file
Received packet with seq 10290
Received 10290 and wrote to file
```

Figure 3: Receiver content for task 1

```
Received 192570 and wrote to file
Received packet with seq 194040
Received 194040 and wrote to file
Received packet with seq 195510
Received 195510 and wrote to file
Received packet with seq 196980
Received 196980 and wrote to file
Received packet with seq 198450
Received 198450 and wrote to file
Received packet with seq 199920
Received 199920 and wrote to file
Received packet with seq 200999
file receive complete
file received in 0.02 seconds
Throughput: 9093.321285456957 B/s
Connection closed
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task1$
```

Figure 4: Receiver content for task 1

5.2 Task 2: Implement reliable data transfer

5.2.1 Sender:

When a connection is established, the sender starts a timer after sending data packets and calculate SampleRTT values for each of the packets. Sender uses the EWMA equation to calculate the TimeOut value. If sender receives same acknowledgement 3 times repeatedly, then it retransmit the data apcket before the TimeOut value.

```
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task2$ python3 sender.py
Server is listening for incoming connections
Accepted connection from ('127.0.0.1', 32862)
size:  1470
Sent packet 1470
Received acknowledgment for packet 1470
sampleRTT:  0.3154 ms
estimatedRTT:  0.0394 ms
devRTT:  0.069 ms
timeout:  0.3154 ms
size:  1470
Sent packet 2940
Received acknowledgment for packet 2940
sampleRTT:  0.1082 ms
estimatedRTT:  0.048 ms
devRTT:  0.0668 ms
timeout:  0.3152 ms
```

Figure 5: Sender content for task 2

```
Sent packet 35280
Received acknowledgment for packet 35280
sampleRTT:  1.5099 ms
estimatedRTT:  0.3713 ms
devRTT:  0.3254 ms
timeout:  1.6727 ms
size:  1470
Sent packet 36750
No acknowledgment received within 5 seconds
Throughput: 14148.23551606894 B/s
Done
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task2$
```

Figure 6: Sender content for task 2

5.2.2 Receiver:

When the connection is established the receiver sends acknowledgement that contains the buffer window size to the sender. The window size defines how much data receiver can accommodate without overflow. Also the acknowledgement contains the sequence number of the next byte.

```
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task2$ python3 receiver.py
Connected to server
Received packet with seq 1470
Received 1470 and wrote to file
Received packet with seq 2940
Received 2940 and wrote to file
Received packet with seq 4410
Received 4410 and wrote to file
Received packet with seq 5880
Received 5880 and wrote to file
Received packet with seq 7350
Received 7350 and wrote to file
Received packet with seq 8820
```

Figure 7: Receiver content for task 2

```
Received packet with seq 30870
Received 30870 and wrote to file
Received packet with seq 32340
Received 32340 and wrote to file
Received packet with seq 33810
Received 33810 and wrote to file
Received packet with seq 35280
Received 35280 and wrote to file
Received packet with seq 36750
Received 36750 and wrote to file
No packet received.
file received in 0.03 seconds
Throughput: 1230.0846068518622 B/s
Connection closed
zisan@zisan-VirtualBox:~/Downloads/Telegram Desktop/Lab5/Task2$
```

Figure 8: Receiver content for task 2

5.3 Task 3: Analyze Results

EstimatedRTT is a weighted average of the SampleRTT values. This weighted average puts more weight on recent samples than on old samples. Such an average is called an exponential weighted moving average (EWMA). The higher estimated RTT suggests that the network has higher delay or packet loss.

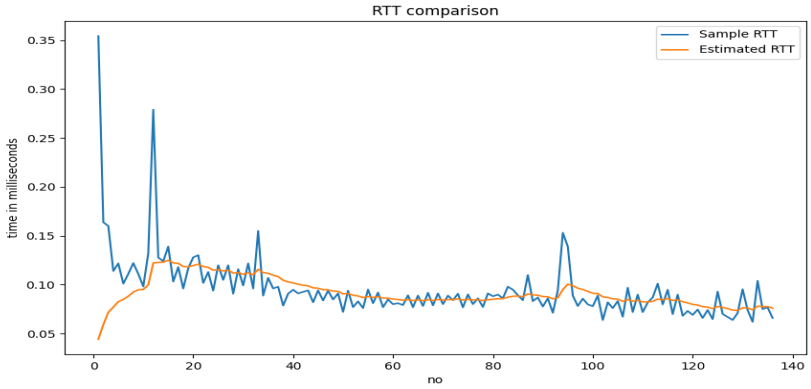

Figure 9: RTT vs estimated RTT

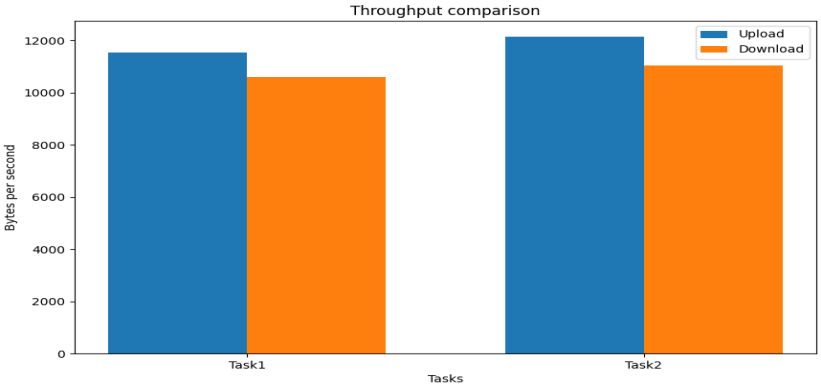

Figure 10: Throughput Comarison

References

[1] James F. Kurose and Keith W. Ross, *Computer Networking: A Top-Down Approach*, 8th Edition, Pearson, 2022.

[2] Geant, *TCP Acknowledgements*, Available at: https://wiki.geant.org/display/public/EK/TCP+Acknowledgements#:~:text=In%20TCP's%20sliding%2Dwindow%20scheme,consecutive%20data%20it%20has%20received.

[3] educative, *What is the TCP Flow Control mechanism?*, Available at: https://www.educative.io/answers/what-is-the-tcp-flow-control-mechanism

[4] Scaler Topics, *TCP Flow Control*, Available at: https://www.scaler.com/topics/computer-network/tcp-flow-control/