



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 5: Implementation of TCP flow control and congestion control algorithm
(TCP Tahoe)

Submitted By:

Name: Meherun Farzana

Roll No : 05

Name: Mohd. Jamal Uddin Mallick

Roll No : 07

Submitted On :

February 21, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Redwan Ahmed Rizvee

1 Introduction

The Transmission Control Protocol (TCP), a popular protocol in computer networking, relies heavily on algorithms for flow control and congestion control. Congestion control limits network overload by controlling data transmission rates, whereas flow control controls the transfer of data from the originator to the recipient. Early TCP versions, such as TCP Tahoe, combine congestion and flow control methods. With TCP Tahoe, flow control is achieved with a sliding window mechanism in which the sender keeps an unacknowledged data window that may be adjusted based on the capacity of the recipient. In response to the receiver's acknowledgments, the window size dynamically changes. A method called slow start is used in TCP Tahoe for congestion control. When it detects network congestion, the sender gradually increases the window size from the modest one it uses initially. To prevent congestion, the sender first detects it, reduces the window size, and then gradually increases the window size until congestion is detected once more. Slow start method integration and flow control configuration using the sliding window mechanism are part of implementing TCP Tahoe.

1.1 Objectives

1. Put into practice the congestion control and flow control methods of TCP Tahoe in a network simulation.
2. Examine how TCP Tahoe controls data flow on the network under various network scenarios.

2 Theory

TCP, as a transport layer protocol in network communication, ensures dependable, ordered, and error-checked transmission of byte streams between applications on hosts communicating via an IP network. It operates in a connection-oriented manner, necessitating the establishment of a connection between client and server before data transmission can occur. For this, the server needs to be in a listening state (passive open) to entertain connection requests from clients. The reliability of TCP is enhanced through mechanisms such as the three-way handshake (active open), retransmission, and error detection. As a result, TCP manages various tasks crucial for establishing seamless communication between host pairs, including connection management, error detection, error recovery, congestion control, connection termination, and flow control. This lab will focus on examining the flow control mechanism and congestion control mechanisms employed by TCP. TCP utilizes a sliding window flow control protocol, wherein the receiver specifies, in the receive window field of each TCP segment, the additional data bytes it can buffer for the connection. The sending host is restricted to sending data up to this specified amount before awaiting acknowledgment and a window update from the receiving host.

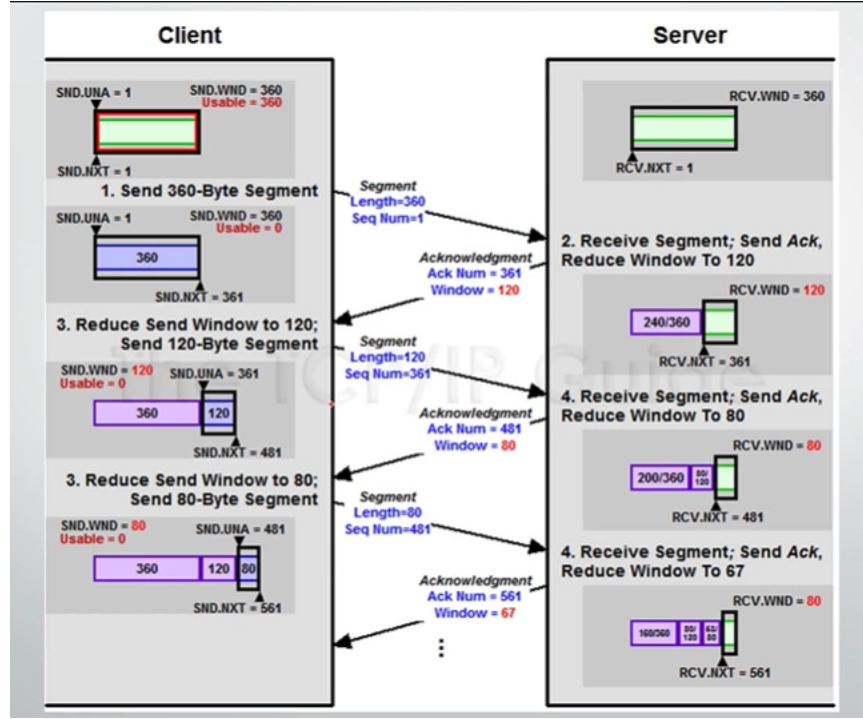


Figure 1: TCP Flow Control

3 Methodology

1. Preparing the network environment: A set of computers are connected with TCP sockets for data communication. The network topology can be either LAN or WAN, depending on the experiment scale.
2. The client and server programs are written using socket programming, with TCP Tahoe algorithms for flow and congestion control. The flow control is based on the sliding window protocol, and the congestion control is based on the slow start technique.
3. The network conditions are varied to create different scenarios, such as high or low bandwidth, latency, and packet loss. A sender and a receiver are assigned for each scenario, and the TCP Tahoe settings are adjusted accordingly.
4. The data transmission between the sender and receiver is carried out over the TCP connection, and the data rate and network status are monitored at regular intervals.
5. The data obtained from the experiments are processed to understand the behavior of TCP Tahoe under different network situations. The TCP Tahoe performance is contrasted with other TCP versions to assess its efficiency and reliability in managing flow and congestion.

4 Experimental Result

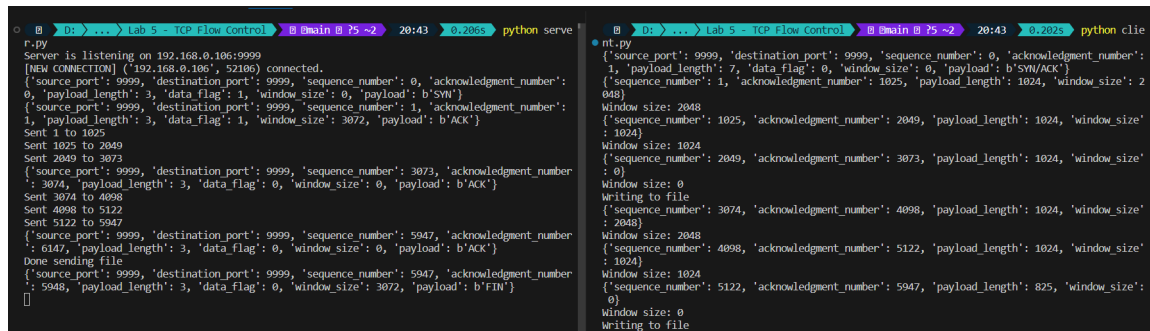
4.1 Task 1: TCP Flow Control

4.1.1 Server

The server is set to wait for client connection at port *9999*. When a client arrives, the three-way handshake is conducted and the client provides its window size. The server sends packets of size *1024* byte payload until the buffer window is filled. When the buffer window gets filled, the client sends an acknowledgment message (*ACK*) as a cumulative acknowledgment for all the received packets. The server continues to send the packets to the client and receives acknowledgment until the whole data has been sent.

4.1.2 Client

The client requests to connect with the server. The server accepts the request and performs the TCP three-way handshake. The client provides its window size to the server. The server sends data considering the window size and when the window gets filled, the client sends a cumulative acknowledgment for the received packets to the server and passes the contents in the buffer to the application layers (writes to the file). Then buffer is emptied and ready to receive more packets. The client continues to receive a packet in such a way until the whole file is received.



```
r.py
Server is listening on 192.168.0.106:9999
[NEW CONNECTION] ('192.168.0.106', 52106) connected.
{'source_port': 9999, 'destination_port': 9999, 'sequence_number': 0, 'acknowledgment_number': 0, 'payload_length': 3, 'data_flag': 1, 'window_size': 0, 'payload': b'SWN'}
{'source_port': 9999, 'destination_port': 9999, 'sequence_number': 1, 'acknowledgment_number': 1, 'payload_length': 3, 'data_flag': 1, 'window_size': 3072, 'payload': b'ACK'}
Sent 1 to 1025
Sent 1025 to 2049
Sent 2049 to 3073
{'source_port': 9999, 'destination_port': 9999, 'sequence_number': 3073, 'acknowledgment_number': 3074, 'payload_length': 3, 'data_flag': 0, 'window_size': 0, 'payload': b'ACK'}
Sent 3074 to 4098
Sent 4098 to 5122
Sent 5122 to 5947
{'source_port': 9999, 'destination_port': 9999, 'sequence_number': 5947, 'acknowledgment_number': 6147, 'payload_length': 3, 'data_flag': 0, 'window_size': 0, 'payload': b'ACK'}
Done sending file
{'source_port': 9999, 'destination_port': 9999, 'sequence_number': 5947, 'acknowledgment_number': 5948, 'payload_length': 3, 'data_flag': 0, 'window_size': 3072, 'payload': b'FIN'}
[]

nt.py
{'source_port': 9999, 'destination_port': 9999, 'sequence_number': 0, 'acknowledgment_number': 1, 'payload_length': 7, 'data_flag': 0, 'window_size': 0, 'payload': b'SWN/ACK'}
{'sequence_number': 1, 'acknowledgment_number': 1025, 'payload_length': 1024, 'window_size': 2048}
Window size: 2048
{'sequence_number': 1025, 'acknowledgment_number': 2049, 'payload_length': 1024, 'window_size': 1024}
Window size: 1024
{'sequence_number': 2049, 'acknowledgment_number': 3073, 'payload_length': 1024, 'window_size': 0}
Window size: 0
Writing to file
{'sequence_number': 3074, 'acknowledgment_number': 4098, 'payload_length': 1024, 'window_size': 2048}
Window size: 2048
{'sequence_number': 4098, 'acknowledgment_number': 5122, 'payload_length': 1024, 'window_size': 1024}
Window size: 1024
{'sequence_number': 5122, 'acknowledgment_number': 5947, 'payload_length': 825, 'window_size': 0}
Window size: 0
Writing to file
```

Figure 2: Server and client side of transferring a text file

```

D:\... > Lab 5 - TCP Flow Control  0 0main 0 75 ~3  20:56  0.196s  python serve
r.py
Server is listening on 192.168.0.106:9999
[NEW CONNECTION] ('192.168.0.106', 52257) connected.
{'source port': 9999, 'destination port': 9999, 'sequence number': 0, 'acknowledgment number': 0, 'payload length': 3, 'data flag': 1, 'window size': 0, 'payload': b'SYN'}
{'source port': 9999, 'destination port': 9999, 'sequence number': 1, 'acknowledgment number': 1, 'payload length': 3, 'data flag': 1, 'window size': 3072, 'payload': b'ACK'}
Sent 1 to 1025
Sent 1025 to 2049
Sent 2049 to 3073
{'source port': 9999, 'destination port': 9999, 'sequence number': 3073, 'acknowledgment number': 3074, 'payload length': 3, 'data flag': 0, 'window size': 0, 'payload': b'ACK'}
Sent 3074 to 4098
Sent 4098 to 5122
Sent 5122 to 6146
{'source port': 9999, 'destination port': 9999, 'sequence number': 6146, 'acknowledgment number': 6147, 'payload length': 3, 'data flag': 0, 'window size': 0, 'payload': b'ACK'}
Sent 6147 to 7171
Sent 7171 to 8195
Sent 8195 to 9219
{'source port': 9999, 'destination port': 9999, 'sequence number': 9219, 'acknowledgment number': 9220, 'payload length': 3, 'data flag': 0, 'window size': 0, 'payload': b'ACK'}
Sent 9220 to 10244
Sent 10244 to 11268
Sent 11268 to 12292
{'source port': 9999, 'destination port': 9999, 'sequence number': 12292, 'acknowledgment number': 12293, 'payload length': 3, 'data flag': 0, 'window size': 0, 'payload': b'ACK'}
Sent 12293 to 13317
Sent 13317 to 14341
Sent 14341 to 15365
{'source port': 9999, 'destination port': 9999, 'sequence number': 15365, 'acknowledgment number': 15366, 'payload length': 3, 'data flag': 0, 'window size': 0, 'payload': b'ACK'}
Sent 15366 to 16390
Sent 16390 to 17414
Sent 17414 to 18438
{'source port': 9999, 'destination port': 9999, 'sequence number': 18438, 'acknowledgment number': 18439, 'payload length': 3, 'data flag': 0, 'window size': 0, 'payload': b'ACK'}

D:\... > Lab 5 - TCP Flow Control  0 0main 0 75 ~3  20:56  0.186s  python client
nt.py
{'source port': 9999, 'destination port': 9999, 'sequence number': 0, 'acknowledgment number': 1, 'payload length': 7, 'data flag': 0, 'window size': 0, 'payload': b'SYN/ACK'}
{'sequence number': 1, 'acknowledgment number': 1025, 'payload length': 1024, 'window size': 2048}
Window size: 2048
{'sequence number': 1025, 'acknowledgment number': 2049, 'payload length': 1024, 'window size': 1024}
Window size: 1024
{'sequence number': 2049, 'acknowledgment number': 3073, 'payload length': 1024, 'window size': 0}
Window size: 0
Writing to file
{'sequence number': 3074, 'acknowledgment number': 4098, 'payload length': 1024, 'window size': 2048}
Window size: 2048
{'sequence number': 4098, 'acknowledgment number': 5122, 'payload length': 1024, 'window size': 1024}
Window size: 1024
{'sequence number': 5122, 'acknowledgment number': 6146, 'payload length': 1024, 'window size': 0}
Window size: 0
Writing to file
{'sequence number': 6147, 'acknowledgment number': 7171, 'payload length': 1024, 'window size': 2048}
Window size: 2048
{'sequence number': 7171, 'acknowledgment number': 8195, 'payload length': 1024, 'window size': 1024}
Window size: 1024
{'sequence number': 8195, 'acknowledgment number': 9219, 'payload length': 1024, 'window size': 0}
Window size: 0
Writing to file
{'sequence number': 9220, 'acknowledgment number': 10244, 'payload length': 1024, 'window size': 2048}
Window size: 2048

```

Figure 3: Server and client side of transferring an image

References

- [1] Geeks For Geeks. Tcp tahoe and tcp reno.
- [2] Nidhi Kumari. Tcp flow control.
- [3] Jim Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson; 8th edition (2020), 2020.