UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 4: Implementation of flow control and reliable data transfer through management of timeout.

Submitted By:

Name : Sudipto Das Sukanto

Roll No : 29

Name: Ahanaf Ahmed Shawn

Roll No : 47

Submitted On :

Feb 21, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Mr. Md. Redwan Ahmed Rizvee

1 Introduction

TCP flow control and congestion management algorithms are fundamental components of the Transmission Control Protocol (TCP), a popular computer networking protocol. Flow control limits the quantity of data a sender can send to a recipient at a time, whereas congestion control limits the rate of data transmission to avoid network congestion.

An early version of TCP that uses algorithms for both congestion and flow control is called TCP Tahoe. A sliding window method is used in TCP Tahoe to manage flow. The sender keeps a window of unacknowledged data open at all times that the recipient can handle. The sender dynamically modifies the window's size in response to the acknowledgments it gets from the recipient.

Slow start is the mechanism used in TCP Tahoe to control congestion. When the sender first starts transmitting data, it sends data via a window that is progressively smaller until it notices network congestion. In the event that congestion is identified, the sender shrinks the window and enters a congested

1.1 Objectives

The following are the major goals of these lab experiments:

- In a network simulation, put the TCP Tahoe flow control and congestion control algorithms to use.
- Examine the ways in which TCP Tahoe controls data flow and handles network congestion in various network scenarios.
- Examine TCP Tahoe's performance in comparison to other TCP versions.
- Examine the information gathered from the experiment to learn more about how well TCP Tahoe manages flow and congestion.
- Make inferences regarding TCP Tahoe's behavior and its applicability to the development and deployment of dependable and efficient network protocols.

2 Theory

TCP is a transport layer protocol used in network communication. Applications running on hosts interacting via an IP network can reliably, systematically, and error-checkedly transfer a stream of data thanks to TCP. Since TCP is connection-oriented, sending data requires first establishing a connection between the client and server. Before a connection is created, the server needs to be passively open, waiting for clients to send connection requests. Retransmission, active open three-way handshake, and error-detection increase reliability. In order to create flawless communications between two hosts, TCP can thus continue to perform a number of functions, such as flow control, connection management, congestion control, error recovery, and detection and recovery. We will examine dependable data transport and the flow control mechanism in this lab.

Sliding window flow control is used by TCP. The amount of additional received data (in bytes) that the receiver is willing to buffer for the connection is specified by the receiver in the receive window field of each TCP segment. That is the maximum amount of data that the sending host can communicate before having to wait for an acknowledgement and window update from the receiving host.

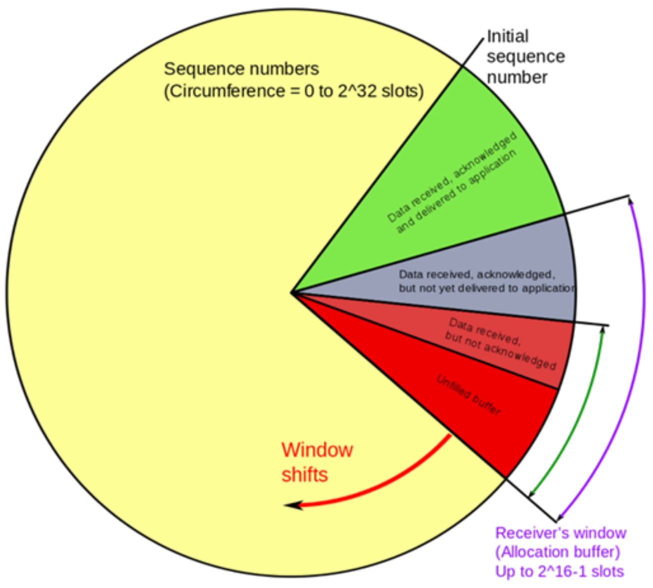

Sequence numbers
(Circumference = 0 to 2^32 slots)
Initial sequence number
Data received, acknowledged and delivered to application
Data received, acknowledged, but not yet delivered to application
Data received, but not acknowledged
Unfilled buffer
Window shifts
Receiver's window (Allocation buffer) Up to 2^16-1 slots

Figure 1: Congestion Window

A congestion management algorithm called TCP Tahoe is used to regulate data flow inside a network and avoid bottlenecks. It is still in widespread usage today and was among the first congestion control algorithms incorporated in TCP.

One of the elements that establishes the maximum number of bytes that can be outstanding at any given time is the congestion window. It is a technique for preventing an excessive amount of traffic from overloading a link between a sender and a recipient, and it is determined by evaluating the degree of congestion present in that link. Slow-start is a component of TCP's congestion control technique. In order to prevent delivering more data than the network can handle, or network congestion, slow-start is used in concert with other techniques.

The sender maintains the congestion window (CWND). Keep in mind that this is not the same as the TCP window size, which is controlled by the receiver. The following illustrates an example of TCP Tahoe's visual behavior:

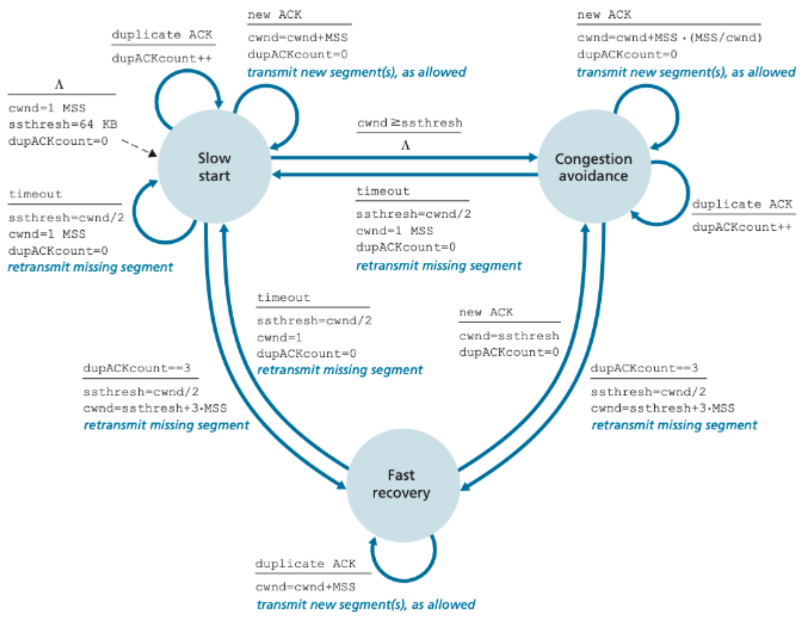


Figure 2: FSM description of TCP congestion

3 General Methodology

1. **Requirement Analysis**:
   - Understand the specific requirements and goals of each task.
   - Identify the functionalities to be implemented and the constraints to be considered.
2. **Design**:
   - Design the architecture and data flow of the client and server applications.
   - Define the protocols and algorithms to be implemented for TCP flow control and reliable data transfer.
3. **Implementation**:
   - Write the code for the server and client applications based on the design specifications.
   - Implement error handling and logging mechanisms to track the behavior of the system during testing.
4. **Testing and Validation**:
   - Conduct thorough testing of the implemented functionalities using test cases and scenarios designed earlier.
   - Validate the correctness, robustness, and performance of the system under different conditions.
   - Collect and analyze the results to ensure that the implemented mechanisms meet the desired objectives.
5. **Documentation**:
   - Document the implementation details, including configuration settings, algorithms, and test results.
   - Provide clear instructions for deploying and running the server and client applications.
6. **Iterative Refinement**:
   - Continuously refine and improve the implementation based on feedback and testing results.
   - Address any identified issues or optimizations to enhance the reliability and efficiency of the system.

4 Experimental Result

4.1 Task 1: Implement TCP Flow Control

4.1.1 Configure Server for TCP Flow Control

- Set up the server application to accept connections from clients.
- Implement socket programming to handle TCP connections.
- Set the receive window size on the server side using appropriate system calls or socket options. This window size determines how much data the server is willing to accept before sending an acknowledgment.

4.1.2 Configure Clients for Cumulative Acknowledgment

- Implement client applications using socket programming.
- Configure the clients to use cumulative acknowledgment, where the receiver sends acknowledgments for the highest sequence number received in order.
- Ensure that the clients are sending data packets to the server with appropriate sequence numbers and handle acknowledgment packets from the server accordingly.

4.1.3 Test TCP Flow Control

1. **Setting Receive Window Size (rwnd)**:
   - The variable `rwnd` represents the receive window size, which determines the maximum number of unacknowledged packets the receiver is willing to accept before its buffer gets full.
2. **Sending Data Packets**:
   - The code reads data from a file (`file_to_send.txt`) and sends it to the client in packets.
   - It divides the data into chunks of 1460 bytes (payload) and constructs TCP segments for each chunk, including transport layer headers with sequence numbers, acknowledgment numbers, window size (`rwnd`), and checksums.
3. **Flow Control Mechanism**:

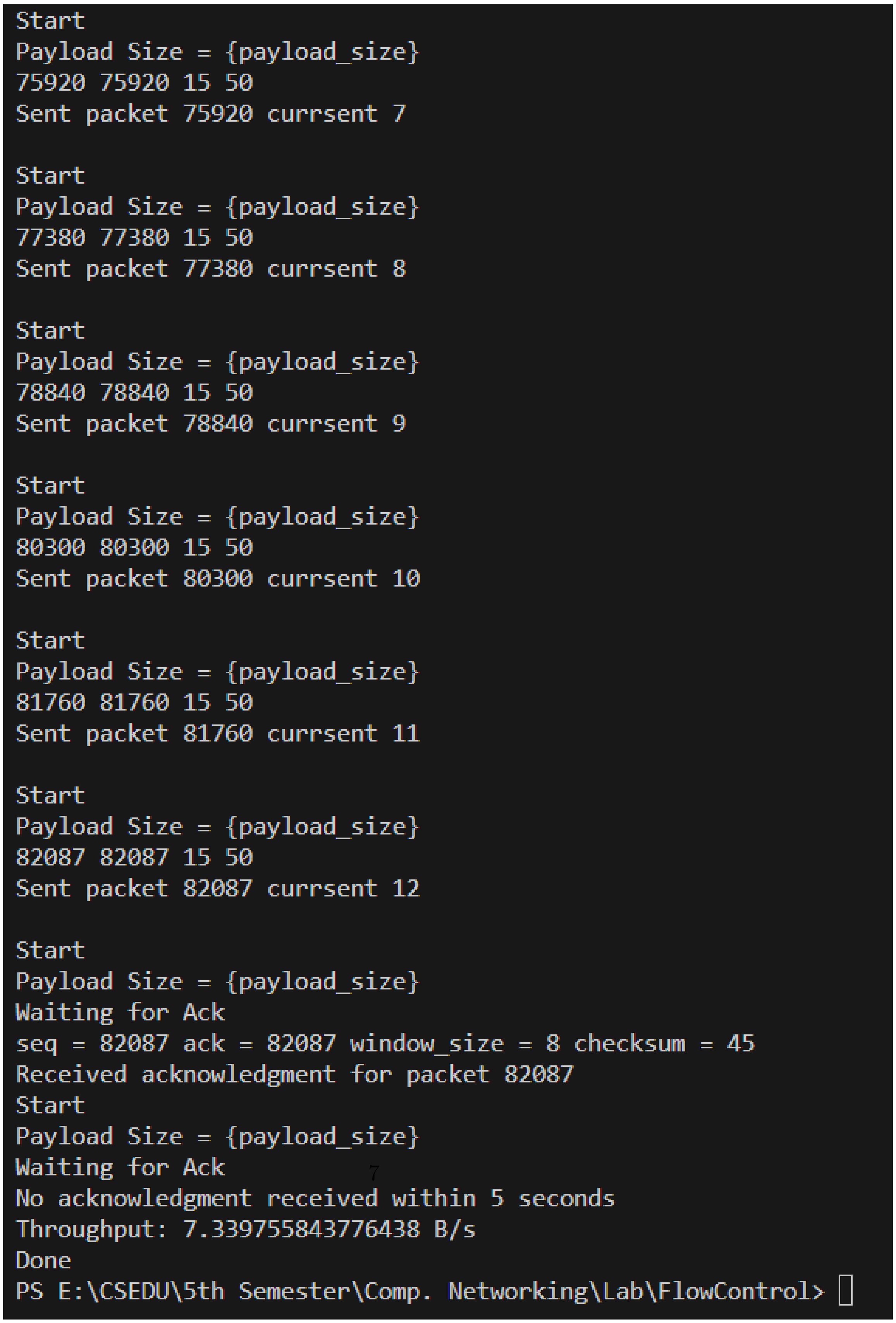
- The client implicitly controls the flow of data by sending acknowledgments back to the server, indicating the next expected sequence number (cumulative acknowledgment).
- The server adjusts its sending behavior based on the acknowledgments received from the client and respects the receiver's window size (`rwnd`).
- In case of acknowledgment timeouts, the server assumes packet loss and retransmits the packet.

4. **Error Handling**:

    - The server handles potential errors, such as timeout during acknowledgment reception (`socket.timeout` exception).

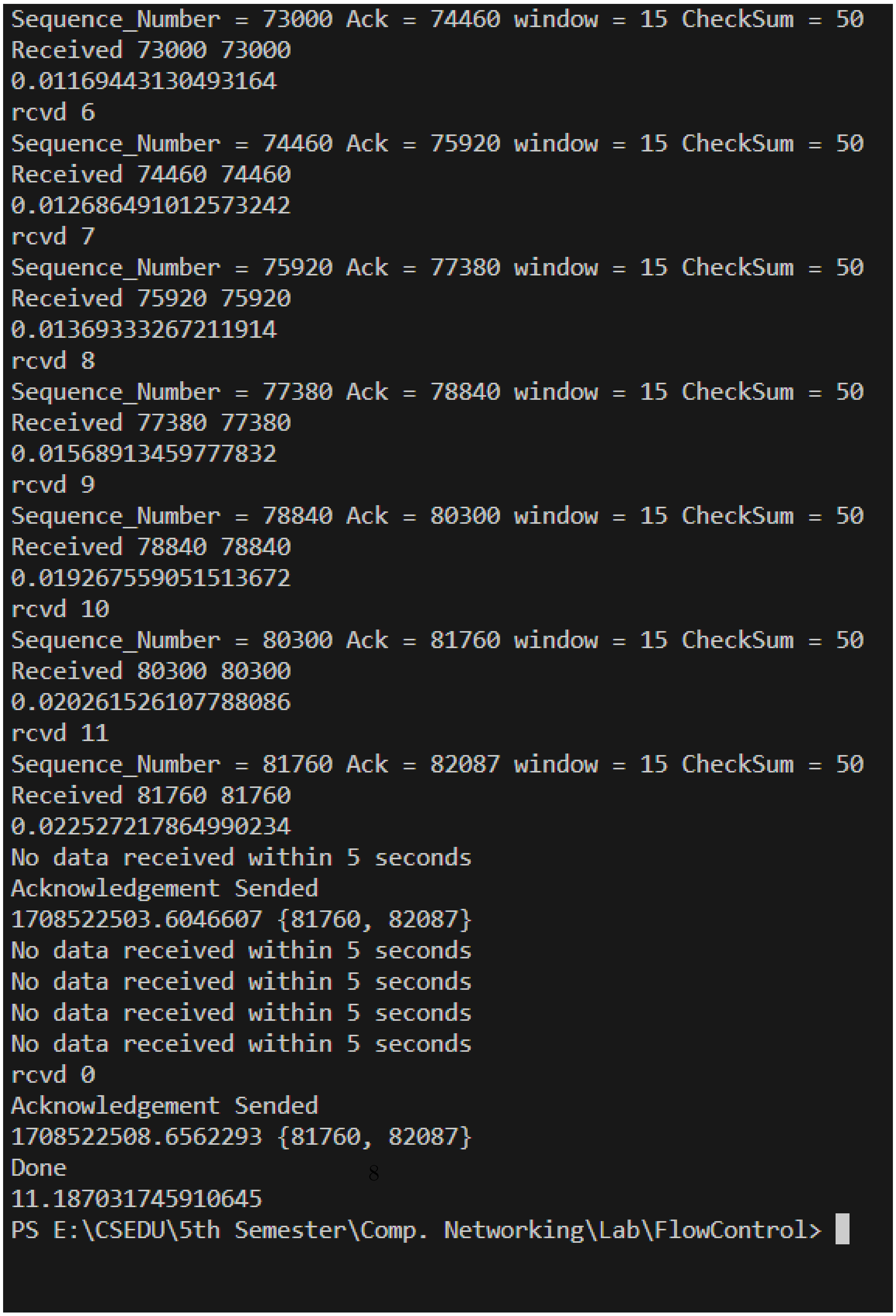
5. **Throughput Calculation**:

    - The code calculates throughput by dividing the total file size by the time taken for transmission.



```
Start
Payload Size = {payload_size}
75920 75920 15 50
Sent packet 75920 currsent 7

Start
Payload Size = {payload_size}
77380 77380 15 50
Sent packet 77380 currsent 8

Start
Payload Size = {payload_size}
78840 78840 15 50
Sent packet 78840 currsent 9

Start
Payload Size = {payload_size}
80300 80300 15 50
Sent packet 80300 currsent 10

Start
Payload Size = {payload_size}
81760 81760 15 50
Sent packet 81760 currsent 11

Start
Payload Size = {payload_size}
82087 82087 15 50
Sent packet 82087 currsent 12

Start
Payload Size = {payload_size}
Waiting for Ack
seq = 82087 ack = 82087 window_size = 8 checksum = 45
Received acknowledgment for packet 82087
Start
Payload Size = {payload_size}
Waiting for Ack
No acknowledgment received within 5 seconds
Throughput: 7.339755843776438 B/s
Done
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\FlowControl>
```

Figure 3: Server output for the Flow control



```
Sequence_Number = 73000 Ack = 74460 window = 15 CheckSum = 50
Received 73000 73000
0.011694431304931641
rcvd 6
Sequence_Number = 74460 Ack = 75920 window = 15 CheckSum = 50
Received 74460 74460
0.012686491012573242
rcvd 7
Sequence_Number = 75920 Ack = 77380 window = 15 CheckSum = 50
Received 75920 75920
0.01369333267211914
rcvd 8
Sequence_Number = 77380 Ack = 78840 window = 15 CheckSum = 50
Received 77380 77380
0.01568913459777832
rcvd 9
Sequence_Number = 78840 Ack = 80300 window = 15 CheckSum = 50
Received 78840 78840
0.019267559051513672
rcvd 10
Sequence_Number = 80300 Ack = 81760 window = 15 CheckSum = 50
Received 80300 80300
0.020261526107788086
rcvd 11
Sequence_Number = 81760 Ack = 82087 window = 15 CheckSum = 50
Received 81760 81760
0.022527217864990234
No data received within 5 seconds
Acknowledgement Sended
1708522503.6046607 {81760, 82087}
No data received within 5 seconds
No data received within 5 seconds
No data received within 5 seconds
No data received within 5 seconds
rcvd 0
Acknowledgement Sended
1708522508.6562293 {81760, 82087}
Done
11.187031745910645
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\FlowControl>
```

Figure 4: Client output for the Flow control

4.2 Task 2: Implement Reliable Data Transfer

4.2.1 Start Timer and Calculate SampleRTT

- Implement a timer mechanism in the client application that starts upon sending each data packet.
- Calculate the SampleRTT values for each packet by measuring the time elapsed between sending the packet and receiving its acknowledgment from the server.

4.2.2 Configure Clients for EWMA Calculation

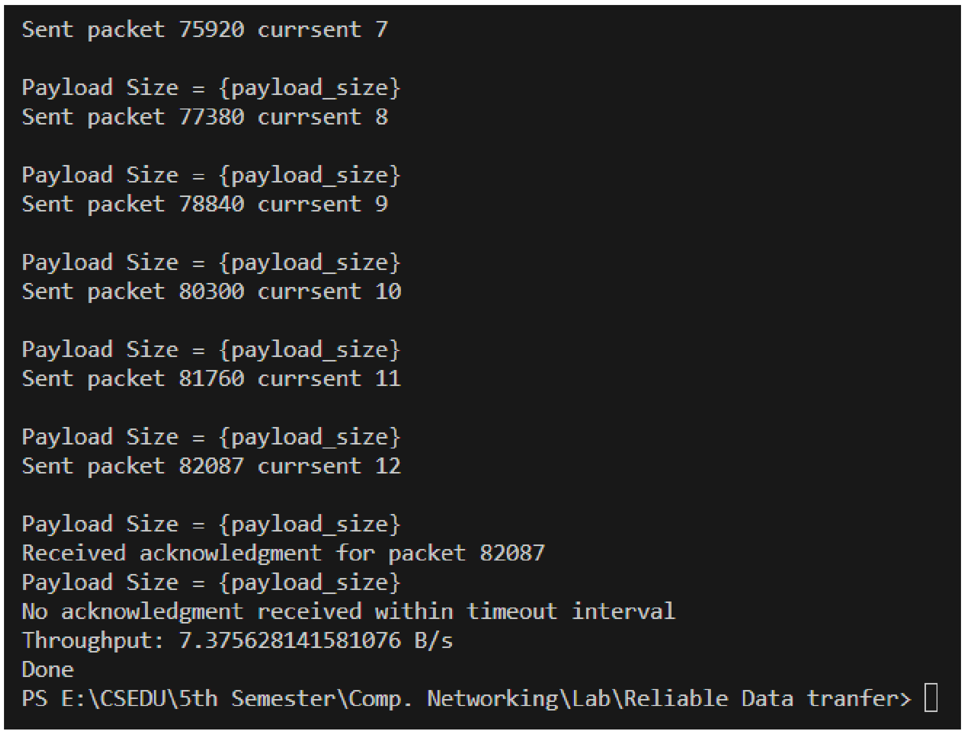
- Implement the Exponential Weighted Moving Average (EWMA) equation in the client application to calculate the Timeout value based on SampleRTT values.
- Adjust the Timeout value dynamically to adapt to changes in network conditions and RTT estimates.

4.2.3 Implement Cumulative ACK and Fast Retransmit

- Modify the client application to handle cumulative acknowledgments from the server.
- Implement the fast retransmit algorithm to detect and recover from packet loss by retransmitting a packet upon receiving three duplicate acknowledgments for the same packet.

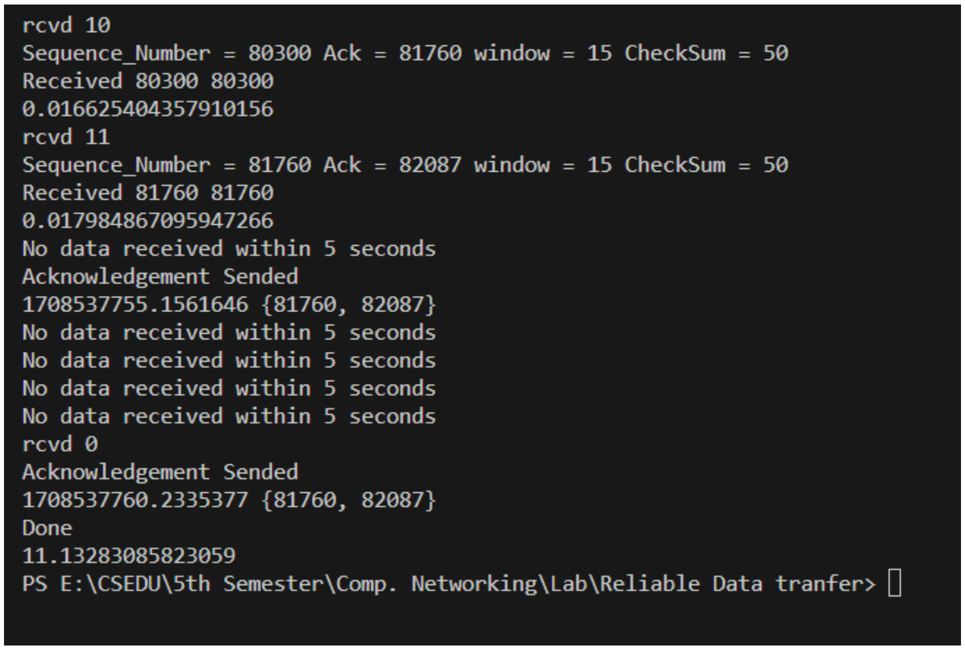
4.2.4 Test Reliable Data Transfer Control Mechanisms

- Design test scenarios to evaluate the reliability and efficiency of the data transfer mechanisms under various network conditions, including different RTT values, packet loss rates, and congestion scenarios.
- Verify that the client can successfully recover from packet loss using fast retransmit and that data transfer is reliable and efficient.



```
Sent packet 75920 currsent 7

Payload Size = {payload_size}
Sent packet 77380 currsent 8

Payload Size = {payload_size}
Sent packet 78840 currsent 9

Payload Size = {payload_size}
Sent packet 80300 currsent 10

Payload Size = {payload_size}
Sent packet 81760 currsent 11

Payload Size = {payload_size}
Sent packet 82087 currsent 12

Payload Size = {payload_size}
Received acknowledgment for packet 82087
Payload Size = {payload_size}
No acknowledgment received within timeout interval
Throughput: 7.375628141581076 B/s
Done
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\Reliable Data tranfer>
```

Figure 5: Server output for the Reliable Data transfer



```
rcvd 10
Sequence_Number = 80300 Ack = 81760 window = 15 CheckSum = 50
Received 80300 80300
0.016625404357910156
rcvd 11
Sequence_Number = 81760 Ack = 82087 window = 15 CheckSum = 50
Received 81760 81760
0.017984867095947266
No data received within 5 seconds
Acknowledgement Sended
1708537755.1561646 {81760, 82087}
No data received within 5 seconds
No data received within 5 seconds
No data received within 5 seconds
No data received within 5 seconds
rcvd 0
Acknowledgement Sended
1708537760.2335377 {81760, 82087}
Done
11.13283085823059
PS E:\CSEDU\5th Semester\Comp. Networking\Lab\Reliable Data tranfer>
```

Figure 6: client output for the Reliable Data transfer