Mini-Project 2 Report

Protocol Specification

To ensure reliable and efficient data transfer over UDP, our protocol implements a pipelined sliding window mechanism for flow control and reliability, as well as an Additive Increase Multiplicative Decrease (AIMD) algorithm for congestion control. This protocol extends standard UDP functionality by addressing its lack of reliability and congestion handling.

Flow Control and Reliability

The protocol employs a sliding window mechanism that allows multiple packets to be sent before awaiting acknowledgment, maximizing throughput.

- Window Size: Represents the maximum number of unacknowledged packets in transit.
- Sliding Behavior: As acknowledgments (ACKs) are received, the window slides forward, enabling new packets to be sent.

Packet Loss Handling:

- If a packet is lost or delayed, the sender detects this via a timeout mechanism. Lost packets are retransmitted, ensuring data integrity.
- Fast retransmit is triggered when duplicate ACKs are received, minimizing delays caused by timeouts.

Connection Management:

- The protocol uses SYN, ACK, and FIN flags for connection setup and termination.
    - SYN: Initiates the connection.
    - ACK: Confirms receipt of packets and maintains order.
    - FIN: Gracefully terminates the connection.

Congestion Control

The AIMD algorithm dynamically adjusts the transmission rate based on network conditions:

- Additive Increase: The congestion window size increases by one after each successful transmission round, promoting higher throughput.
- Multiplicative Decrease: On detecting packet loss or timeout, the congestion window is halved, preventing further congestion.

This balance ensures efficient utilization of bandwidth while maintaining fairness across competing traffic flows.

Packet structure

Each UDP packet is augmented with a custom header to enable reliability and flow control. The header fields are:

Field Name	Size (bits)	Purpose
Sequence Number	32	Identifies and orders packets uniquely.
Acknowledgement Number	32	Confirms receipt of packets.
Flags	8	Control signals (e.g., SYN, ACK, FIN).
Window Size	16	Indicates the receiver's available buffer size.

The custom header enables essential protocol functions like sequencing, acknowledgment, and flow management with minimal performance overhead.

Description of Code

The sender code implements a sliding window protocol where it maintains a window of unacknowledged packets. It sends packets up to the window size and starts a timer for the oldest unacknowledged packet. If an acknowledgment is received, the window slides forward, and the sender transmits new packets. In case of a timeout, the sender retransmits all packets in the current window starting from the base sequence number. Metrics such as the number of retransmissions and the congestion window size are tracked to evaluate the protocol's performance.

The receiver code buffers out-of-order packets and sends acknowledgments for the highest in-order sequence number received. When the expected packet arrives, it delivers it to the application layer and checks if subsequent buffered packets can be delivered in order. The receiver also discards corrupted or malformed packets, logging these events for debugging and analysis. This mechanism ensures reliable delivery and proper sequencing of packets over UDP.

The sender implements AIMD (Additive Increase Multiplicative Decrease) congestion control by maintaining a congestion window (cwnd) and a slow start threshold

(ssthresh). Initially, the sender is in the slow start phase, where cwnd increases exponentially with each acknowledgment received. Once cwnd surpasses ssthresh, the sender enters the congestion avoidance phase, incrementing cwnd linearly. Packet loss is simulated using a random function; when a loss is detected (simulating network congestion), ssthresh is set to half of the current cwnd, and cwnd is reset to 1 to restart slow start. This approach allows the sender to adjust its transmission rate based on perceived network conditions, balancing throughput with congestion control.

Test Procedures

To validate the protocol's functionality, reliability, and congestion control, we designed the following tests:

1. **Functional Testing:**
    - Setup: Run sender and receiver programs with no simulated packet loss or corruption.
    - Steps:
        - Initiate a connection using the sender and receiver.
        - Transmit a sequence of packets.
        - Observe acknowledgments and retransmissions.
    - Metrics Collected:
        - Sequence numbers of sent/received packets.
        - Number of retransmissions.
        - Throughput (packets per second).
    - Expected Results:
        - All packets are successfully transmitted and acknowledged in order.
        - No retransmissions occur under ideal conditions.

2. **Packet Loss Simulation Testing:**
    - Setup: Enable packet loss and corruption simulations using LOSS_PROBABILITY and CORRUPTION_PROBABILITY set to 10%.
    - Steps:
        - Configure the sender to simulate random packet losses.
        - Observe retransmissions triggered by timeouts or duplicate ACKs.
        - Validate the receiver's ability to reassemble out-of-order packets.
    - Metrics Collected:
        - Retransmission count.
        - Throughput.
        - Correctness of reassembled packets.
    - Expected Results:
        - Lost packets are retransmitted.

    - Receiver buffers out-of-order packets and reassembles the original sequence.
    - Throughput decreases proportionally to the loss rate.

3. **Edge Case Testing for Congestion Control**
    - Setup: Simulate heavy packet loss with LOSS_PROBABILITY set to 30%.
    - Steps:
        - Observe AIMD behavior under high loss rates.
        - Capture network activity using Wireshark.
    - Metrics Collected:
        - Congestion window size changes (cwnd).
        - Throughput under heavy loss conditions.
    - Expected Results:
        - The congestion window decreases after losses and increases in the absence of congestion.
        - The protocol maintains reliable delivery despite high packet loss.

Results and Analysis

Logs and Observations

The logs demonstrate the protocol's connection-oriented behavior:

- Reliable Transmission: Sequence numbers and ACKs confirm packet delivery in order.
- Congestion Control:
    - The congestion window (cwnd) grows exponentially during the slow-start phase and linearly in the congestion-avoidance phase.
    - On detecting loss, cwnd reduces multiplicatively and restarts slow-start.

Throughput Analysis

Assuming each packet is 100 bytes:

- Ideal Conditions (0% Loss): Throughput = 19,046.75 bytes/sec.
- 10% Loss: Throughput = 18,978.75 bytes/sec.
- 50% Loss: Throughput = 326.68 bytes/sec.

Graph Analysis

The graph generated using iperf3 shows:

- Both our protocol and competing traffic fairly share network bandwidth.

- The congestion window adjusts dynamically based on packet loss, preventing either flow from dominating.

Conclusion

Our protocol extends UDP with features for reliability, flow control, and congestion management.

- Reliable Transmission: Achieved via sliding window and retransmission mechanisms.
- Congestion Management: AIMD ensures fair bandwidth usage and responsiveness to network conditions.
- Robust Testing: Functional and simulated tests validate the protocol's behavior under various conditions.

Logs

Sent: 0:Hello Receiver!
Sent: 1:Hello Receiver!
Received ACK: -1, Window Size: 3
Sent: 2:Hello Receiver!
Received ACK: -1, Window Size: 4
Sent: 3:Hello Receiver!
Sent: 4:Hello Receiver!
Received ACK: -1, Window Size: 2
Fast retransmit triggered for packet 0
Sent: 0:Hello Receiver!
Received ACK: 2, Window Size: 3
Sent: 6:Hello Receiver!
Received ACK: 2, Window Size: 4
Timeout occurred. Resending unacknowledged packets.
Socket timeout. Triggering timeout handler.
Sent: 3:Hello Receiver!
Sent: 4:Hello Receiver!
Sent: 6:Hello Receiver!
Timeout occurred. Resending unacknowledged packets.
Sent: 3:Hello Receiver!
Sent: 4:Hello Receiver!
Sent: 6:Hello Receiver!
Received ACK: 2, Window Size: 3
Received ACK: 4, Window Size: 2
Simulated loss: Packet 7 not sent

```
Received ACK: 4, Window Size: 2
Received ACK: 4, Window Size: 3
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 6:Hello Receiver!
Sent: 7:Hello Receiver!
Timeout occurred. Resending unacknowledged packets.
Simulated loss: Packet 6 not sent
Sent: 7:Hello Receiver!
Received ACK: 4, Window Size: 4
Fast retransmit triggered for packet 5
Sent: 5:Hello Receiver!
Received ACK: 4, Window Size: 5
Received ACK: 4, Window Size: 6
Received ACK: 7, Window Size: 7
Sent: 15:Hello Receiver!
Received ACK: 7, Window Size: 8
Timeout occurred. Resending unacknowledged packets.
Socket timeout. Triggering timeout handler.
Sent: 15:Hello Receiver!
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Received ACK: 7, Window Size: 9
Received ACK: 7, Window Size: 10
Fast retransmit triggered for packet 8
Sent: 8:Hello Receiver!
Received ACK: 8, Window Size: 10
Sent: 19:Hello Receiver!
Received ACK: 8, Window Size: 10
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Simulated loss: Packet 15 not sent
Sent: 19:Hello Receiver!
Received ACK: 8, Window Size: 10
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Received ACK: 8, Window Size: 10
Fast retransmit triggered for packet 9
```

Sent: 9:Hello Receiver!
Received ACK: 8, Window Size: 10
Received ACK: 9, Window Size: 10
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Received ACK: 9, Window Size: 10
Received ACK: 9, Window Size: 10
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Received ACK: 9, Window Size: 10
Fast retransmit triggered for packet 10
Sent: 10:Hello Receiver!
Received ACK: 9, Window Size: 10
Received ACK: 10, Window Size: 10
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Received ACK: 10, Window Size: 2
Received ACK: 10, Window Size: 2
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Received ACK: 10, Window Size: 2
Fast retransmit triggered for packet 11
Sent: 11:Hello Receiver!
Received ACK: 10, Window Size: 2
Received ACK: 11, Window Size: 3
Sent: 15:Hello Receiver!
Received ACK: 11, Window Size: 4
Timeout occurred. Resending unacknowledged packets.

Socket timeout. Triggering timeout handler.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Received ACK: 11, Window Size: 5
Received ACK: 11, Window Size: 6
Fast retransmit triggered for packet 12
Simulated loss: Packet 12 not sent
Received ACK: 11, Window Size: 7
Received ACK: 11, Window Size: 8
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Received ACK: 11, Window Size: 9
Fast retransmit triggered for packet 12
Sent: 12:Hello Receiver!
Received ACK: 12, Window Size: 5
Sent: 18:Hello Receiver!
Received ACK: 12, Window Size: 5
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Simulated loss: Packet 18 not sent
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Received ACK: 12, Window Size: 5
Received ACK: 12, Window Size: 5
Fast retransmit triggered for packet 13
Simulated loss: Packet 13 not sent
Received ACK: 12, Window Size: 5
Received ACK: 12, Window Size: 5
Received ACK: 12, Window Size: 5
Fast retransmit triggered for packet 13
Sent: 13:Hello Receiver!

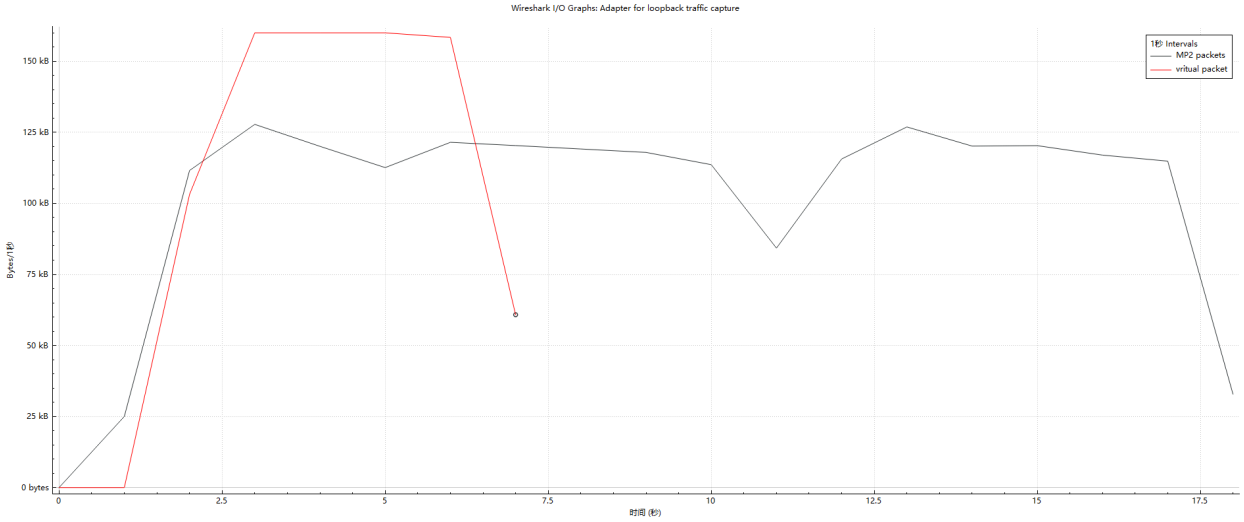
Received ACK: 13, Window Size: 5
Sent: 19:Hello Receiver!
Received ACK: 13, Window Size: 5
Timeout occurred. Resending unacknowledged packets.
Socket timeout. Triggering timeout handler.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Timeout occurred. Resending unacknowledged packets.
Sent: 15:Hello Receiver!
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Received ACK: 13, Window Size: 5
Received ACK: 13, Window Size: 5
Fast retransmit triggered for packet 14
Sent: 14:Hello Receiver!
Received ACK: 13, Window Size: 5
Received ACK: 13, Window Size: 5
Received ACK: 13, Window Size: 1
Fast retransmit triggered for packet 14
Sent: 14:Hello Receiver!
Received ACK: 15, Window Size: 2
Sent: 18:Hello Receiver!
Received ACK: 15, Window Size: 2
Received ACK: 15, Window Size: 2
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Timeout occurred. Resending unacknowledged packets.
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Received ACK: 15, Window Size: 2
Fast retransmit triggered for packet 16
Simulated loss: Packet 16 not sent
Received ACK: 15, Window Size: 2
Received ACK: 15, Window Size: 2
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 19:Hello Receiver!

```
Sent: 18:Hello Receiver!
Received ACK: 15, Window Size: 2
Fast retransmit triggered for packet 16
Simulated loss: Packet 16 not sent
Received ACK: 15, Window Size: 2
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Received ACK: 15, Window Size: 2
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Received ACK: 15, Window Size: 2
Fast retransmit triggered for packet 16
Sent: 16:Hello Receiver!
Received ACK: 15, Window Size: 2
Received ACK: 16, Window Size: 3
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Received ACK: 16, Window Size: 4
Received ACK: 16, Window Size: 5
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Simulated loss: Packet 19 not sent
Sent: 18:Hello Receiver!
Socket timeout. Triggering timeout handler.
Timeout occurred. Resending unacknowledged packets.
Sent: 19:Hello Receiver!
Sent: 18:Hello Receiver!
Received ACK: 16, Window Size: 2
Fast retransmit triggered for packet 17
Sent: 17:Hello Receiver!
Received ACK: 16, Window Size: 2
Received ACK: 19, Window Size: 3
All packets acknowledged. Exiting...
Sender completed transmission.
```

Logs Breakdown

The logs show the sequence numbers, ACKs, and retransmissions. Which prove that our protocol has Connection-Oriented Behavior.
Assuming each packet is 100 bytes, the throughput is 19046.75 bytes/sec with 0 possibility of loss and corruption. 18978.75 bytes/sec with a 10% possibility of loss and corruption. 326.68 bytes/sec with a 50% possibility of loss and corruption. The timeout time in those tests is one second.
Also from the Logs, we can find that the window size is dynamically adjusted based on the virtual error we add to the function. Which prove for the flow control. For congestion control, in the code, cwnd grows exponentially in the slow start phase and linearly in congestion avoidance. On timeout or loss, cwnd reduces multiplicatively and starts slow start again.

Bouns



iperf3 -c 127.0.0.1 -u -p 10001 -b 102400 -t 5 -l 128 -i 1
We use this command to generate a virtual UCP load in the network. And my protocol runs around 6 seconds, and the iperf data stream runs for 5 seconds.
From the graph, can tell, that both our protocol and the competing traffic are active, with neither completely dominating the bandwidth. The interaction between the two flows suggests that both are adjusting their behavior to share the network fairly.