COL-334

COMPUTER NETWORKS

ASSIGNMENT-4

KARUNA - 2019CS10364
RACHAPROLU SHREJA - 2022CS51145

Part-1 : Reliability

Introduction

This project aims to implement a reliable data transmission mechanism over UDP by incorporating techniques typically used in TCP to handle packet loss and delays. UDP does not natively guarantee reliability, so we have added mechanisms such as acknowledgments (ACKs), retransmissions, fast recovery, packet numbering, and timeout mechanisms to achieve reliable communication between the server and client.

Server Implementation

The server script sends a file to a client over UDP, with mechanisms to manage retransmissions and track network performance metrics like RTT (Round-Trip Time).

1. Connection Initialization: The server binds to a specified IP and port and waits for a "START" signal from the client to establish a connection.
2. Packet Segmentation and Transmission: The file is read in chunks of `MSS` (Maximum Segment Size) and each chunk is encapsulated in a packet with a sequence number. The server sends packets within a defined window size (`WINDOW_SIZE`), implementing a sliding window protocol for flow control.
3. Acknowledgment and RTT Calculation: For each received acknowledgment (ACK), the server calculates the RTT and updates the retransmission timeout (RTO) based on the Smoothed RTT (`srtt`) and RTT variance (`rttvar`).
4. Fast Recovery: If enabled, fast recovery is triggered when the server detects duplicate ACKs exceeding a threshold (`DUP_ACK_THRESHOLD`). In this mode, the server retransmits the earliest unacknowledged packet without waiting for a timeout.
5. End of Transmission: Upon finishing the file transmission, the server sends an "END" signal to the client, indicating the end of data.

Client Implementation

The client script receives packets from the server, handles packet reordering, and writes the received data to a file in the correct order.

1. Connection Initialization: The client starts by sending a "START" signal to the server, establishing a connection.
2. Packet Reception and Ordering: The client processes incoming packets based on sequence numbers. It maintains a buffer for out-of-order packets and writes packets to the file only when they arrive in the expected order.

3. Acknowledgment and Retransmission: After writing each correctly received packet, the client sends an ACK with the next expected sequence number to the server. In case of timeouts, it retransmits the last ACK, prompting the server to resend lost packets.

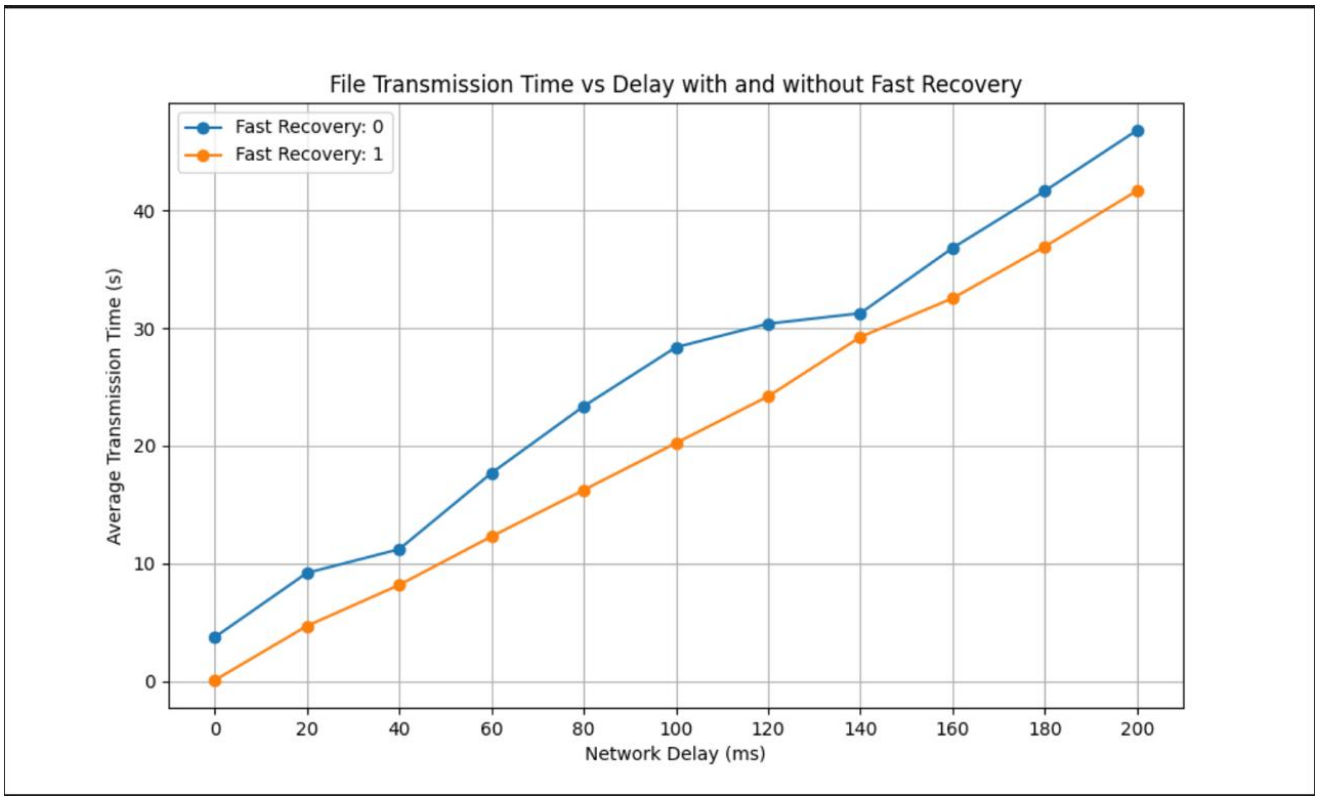
4. End of Transmission: When the client receives the "END" signal from the server, it responds with an "END_ACK" to confirm receipt of the entire file, ending the transfer.

- Sliding Window Protocol: The server maintains a window of unacknowledged packets, sliding forward as ACKs are received. This controls the data flow and prevents overloading the network.
- Adaptive Timeout Calculation: The timeout dynamically adjusts based on RTT measurements, using factors `ALPHA` and `BETA` to balance RTT stability and responsiveness.
- Fast Recovery: Fast recovery retransmits packets before timeouts when duplicate ACKs are received, potentially reducing retransmission delays.
- Error and Timeout Handling: The client and server handle unexpected packet loss and out-of-order packets gracefully, ensuring data integrity.

This system mimics reliable data transfer features over UDP, providing a foundation for scenarios requiring reliability and congestion control without using TCP.

Results

1. File Transmission Time vs Network Delay



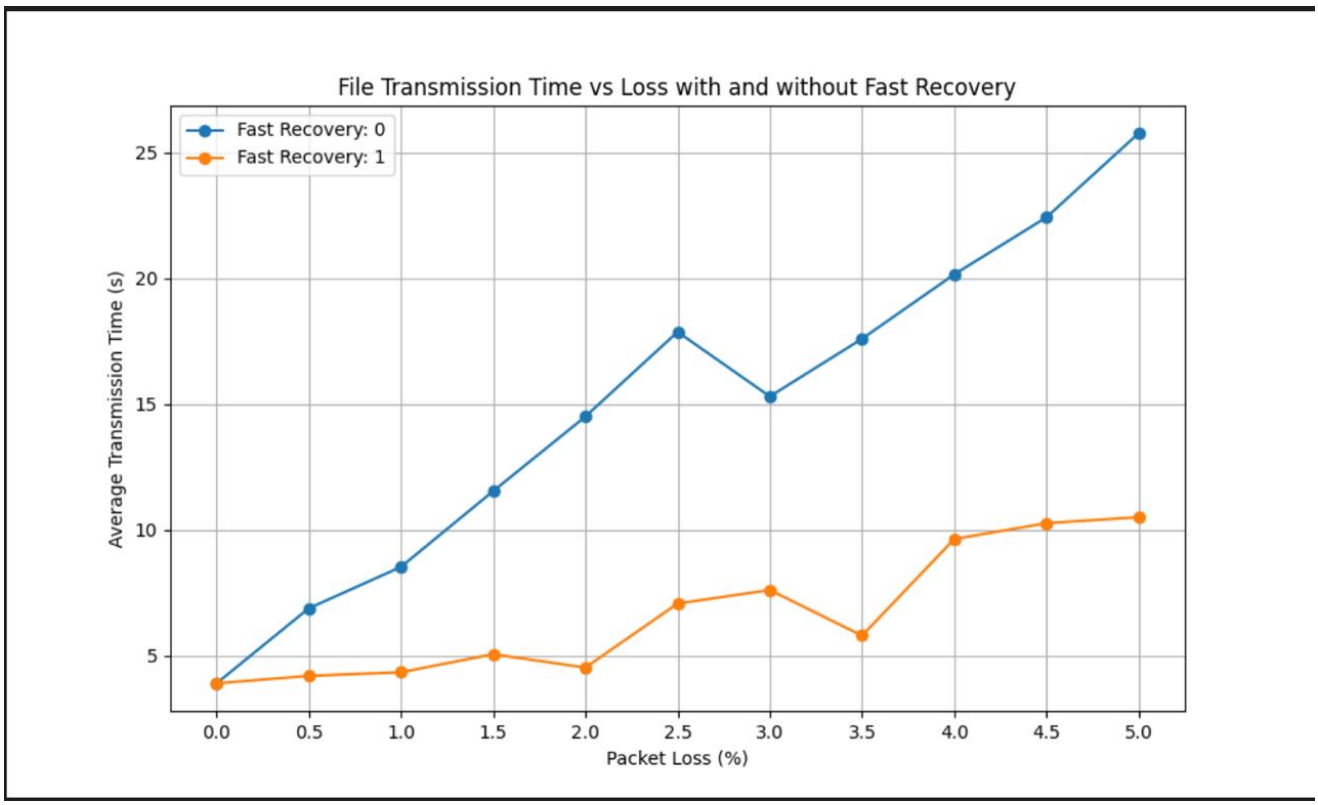
This plot compares the average file transmission time as network delay increases, with and without Fast Recovery enabled.

- Observation: Without Fast Recovery (blue line), the transmission time increases significantly as delay increases. This is expected, as higher delays mean longer wait times for ACKs, resulting in additional retransmissions due to timeout.

- With Fast Recovery (orange line): Fast Recovery helps reduce the impact of delay, as the server retransmits missing packets more quickly upon detecting duplicate ACKs. This keeps transmission times relatively lower compared to the scenario without Fast Recovery.

Conclusion: Fast Recovery is beneficial in high-delay networks by reducing transmission times through quicker recovery from packet losses.

2. File Transmission Time vs Packet Loss Rate



This plot illustrates how transmission time varies with different packet loss rates, with and without Fast Recovery.

- Observation: As packet loss rate increases, transmission time increases more dramatically without Fast Recovery (blue line). When Fast Recovery is enabled (orange line), the transmission time grows more slowly and remains significantly lower across loss rates.

- Explanation: Fast Recovery enables the server to retransmit lost packets upon receiving duplicate ACKs instead of waiting for a timeout. This minimizes the retransmission delay caused by packet loss, thereby keeping the file transmission time more stable.

Conclusion: Fast Recovery shows a clear improvement in maintaining lower transmission times under high packet loss conditions, making it effective in unreliable network environments.

PART-2 :congestion control

Server Implementation

The server's primary role is to send a file over UDP, ensuring reliable delivery with congestion control.

1. Socket Initialization: The server creates a UDP socket(server_socket), binds it to the given IP address and port, and starts listening for incoming client connections.
2. Congestion Control:
    - The server employs TCP-like congestion control with slow start, congestion avoidance, and fast recovery. Initially, the server uses a congestion window (`cwnd`) of 1 MSS (Maximum Segment Size) and increases it with each acknowledged packet.
    - Slow Start: The `cwnd` is doubled as packets are acknowledged.
    - Congestion Avoidance: Once `cwnd` reaches a threshold, it increases more gradually to avoid congestion.
    - Fast Recovery: If three duplicate ACKs are received (indicating packet loss), the server reduces the threshold and the `cwnd` to quickly recover.
3. Packet Sending:
    - The file is read in chunks of MSS-sized packets.
    - The server sends each packet, maintains a list of unacknowledged packets, and retransmits lost packets based on timeouts or duplicate ACKs.
4. RTT and Timeout: The server computes the Round-Trip Time (RTT) for each packet and adjusts the retransmission timeout (RTO) accordingly using Exponential Weighted Moving Average (EWMA).
5. End Signal: When the file has been fully transmitted, the server sends an "END" signal to the client, indicating the completion of the transfer.

Client Implementation

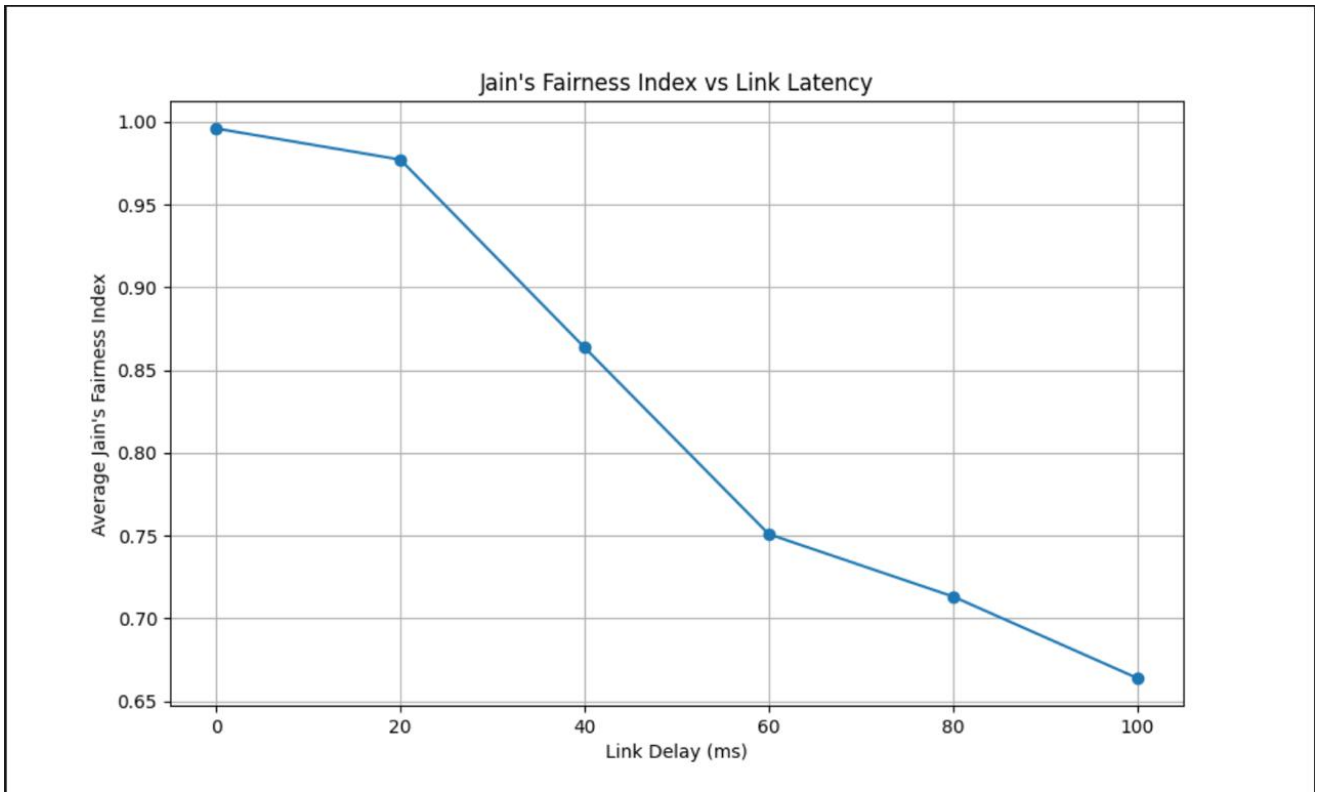
The client is responsible for receiving the file from the server and ensuring reliable receipt by handling out-of-order packets, timeouts, and retransmissions.

1. Socket Initialization: The client creates a UDP socket(client_socket) and sends a "START" signal to establish the connection with the server.
2. Receiving Packets:
    - The client listens for packets and acknowledges the last successfully received packet using cumulative ACKs.
    - It processes each incoming packet, verifies the sequence number, and writes the data to a file. If packets arrive out of order, the client buffers them for later processing.
3. Out-of-Order Handling: The client stores out-of-order packets in a buffer and only writes them when the expected sequence number is received. It ensures that any gaps in the data stream are filled before writing the buffered packets.
4. ACK Handling:
    - The client sends an **ACK** for each packet, indicating the expected sequence number.
    - If a packet is not received within a timeout period, the client retransmits the last acknowledged sequence number.
5. End Signal: Upon receiving the "END" signal from the server, the client sends an "END_ACK" to acknowledge the completion of the file transfer.

- Congestion Control: Both the server and the client work together using a **sliding window** mechanism to avoid network congestion and ensure efficient data transfer.
- Reliable Delivery: The use of cumulative ACKs, retransmissions, and buffer management on the client side ensures that the file is reliably received, even in the presence of packet loss or delays.
- Timeout and Retransmission: The server dynamically adjusts retransmission timeouts based on RTT measurements, and the client retransmits ACKs if timeouts occur.

Result:

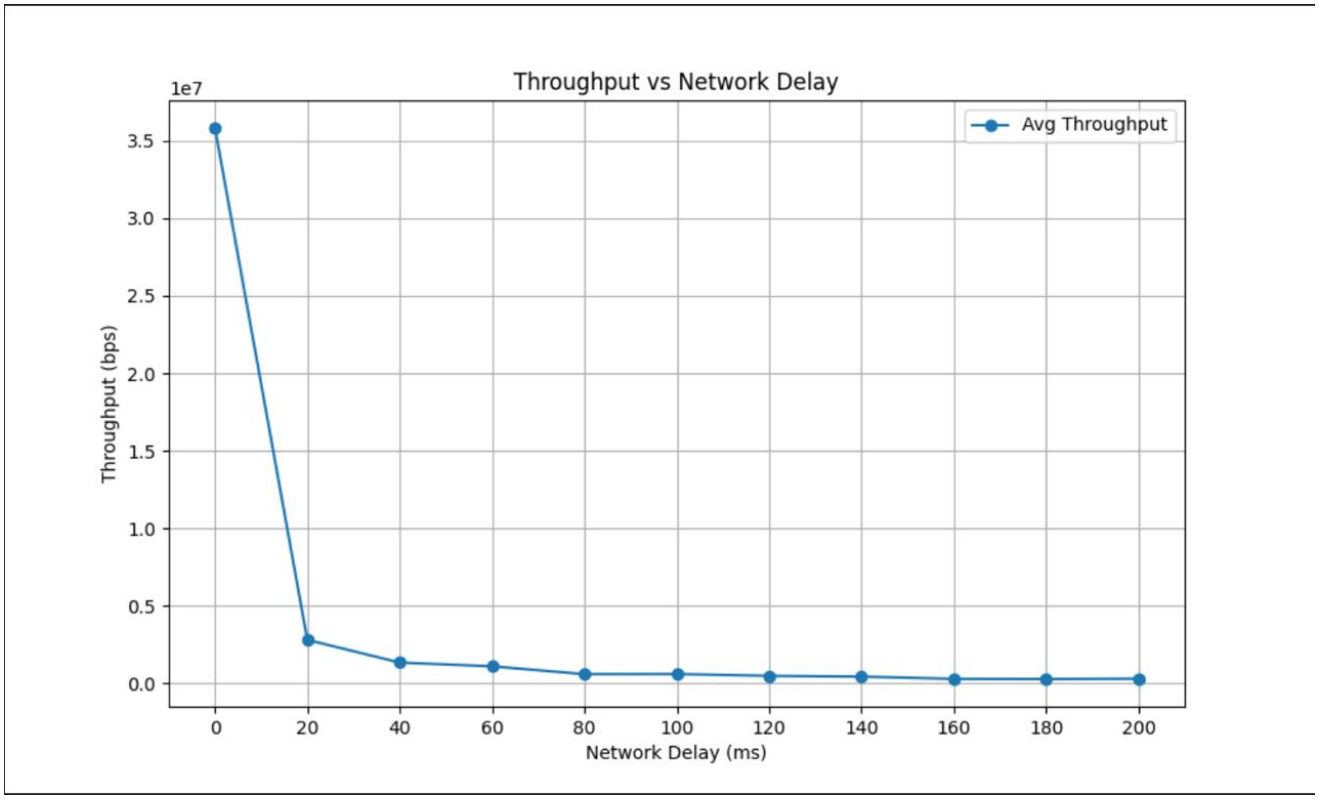
1.



As the link delay increases from 0 ms to 100 ms, the Jain's fairness index decreases, indicating that the fairness of resource allocation between the client-server pairs worsens. This happens because the increased delay affects the congestion control algorithm's ability to adjust its congestion window in a timely manner. With higher delays, the feedback loop becomes slower, causing inconsistent growth in the congestion windows of the flows. One flow may experience slower throughput due to delayed acknowledgments, while the other flow may ramp up faster, leading to an unfair distribution of resources. This results in lower fairness as the link delay increases.

2.

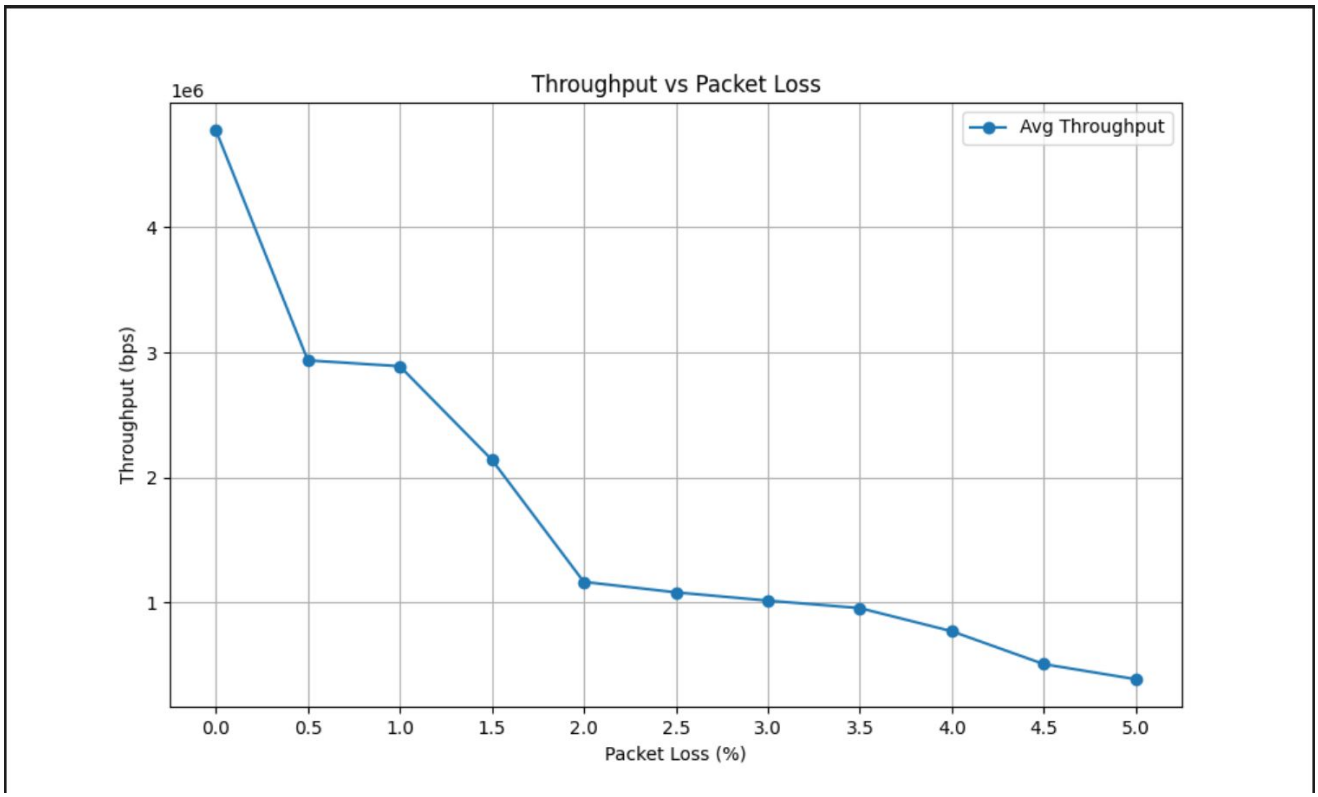
2.a.



Delay Experiment: As the link delay increases, the throughput decreases, This is because higher delays result in slower round-trip times (RTTs), which lead to slower feedback for the congestion control algorithm. This causes TCP to wait longer for acknowledgments, reducing the amount of data sent during each round of transmission. However, the throughput doesn't decrease in a simple linear fashion because other factors, such as TCP's congestion window adjustments and the interaction between delay and bandwidth, also influence the throughput.

2.b.

Packet Loss Experiment: The throughput also decreases with higher packet loss, and the relationship roughly follows the expected pattern, but again not exactly in a linear fashion. According to the literature, throughput is inversely proportional to the square root of the packet loss rate, and you observed that as packet loss increases, throughput decreases as expected, but the rate of decrease slows down. This happens because higher packet loss causes TCP to enter congestion control mechanisms, such as reducing the congestion window, but as losses increase, the impact of additional loss on throughput becomes less significant due to the TCP congestion control mechanisms already being in place.


Throughput vs Packet Loss
1e6
Avg Throughput
Throughput (bps)
4
3
2
1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Packet Loss (%)