Assignment 4

COL672 || Computer Networks

Anand Sharma [2024JCS2049]
Surajprakash Narwariya [2024JCS2044]

Part 1: Reliability

In this assignment, we implemented a reliable file transfer protocol using UDP, with an emphasis on achieving reliability through various mechanisms such as retransmission of lost packets, fast recovery from packet loss, and adaptive timeout handling. The implementation consists of two main components: a server (`server.py`) that sends the file and a client (`client.py`) that receives it. The experiments conducted involved varying network conditions to analyze the protocol's performance.

Implementation Overview

Server Functionality

The server initiates by waiting for the client to request a file transfer. Once the connection is established, the server:

- Sends packets of data from a specified file while ensuring the Maximum Segment Size (MSS) does not exceed 500 bytes.
- Implements a sliding window mechanism to manage multiple packets in transit, determined by a predefined window size (4 packets).
- Handles acknowledgment packets from the client and adjusts its timeout dynamically based on Round Trip Time (RTT) measurements.

Fast Recovery Mechanism

The protocol supports fast recovery through:

- Detection of duplicate ACKs, which indicates packet loss.
- Retransmission of the earliest unacknowledged packet to maintain flow efficiency.

- Adjustment of the transmission timeout to reduce the likelihood of unnecessary retransmissions, based on estimated RTT.

Experiments Conducted

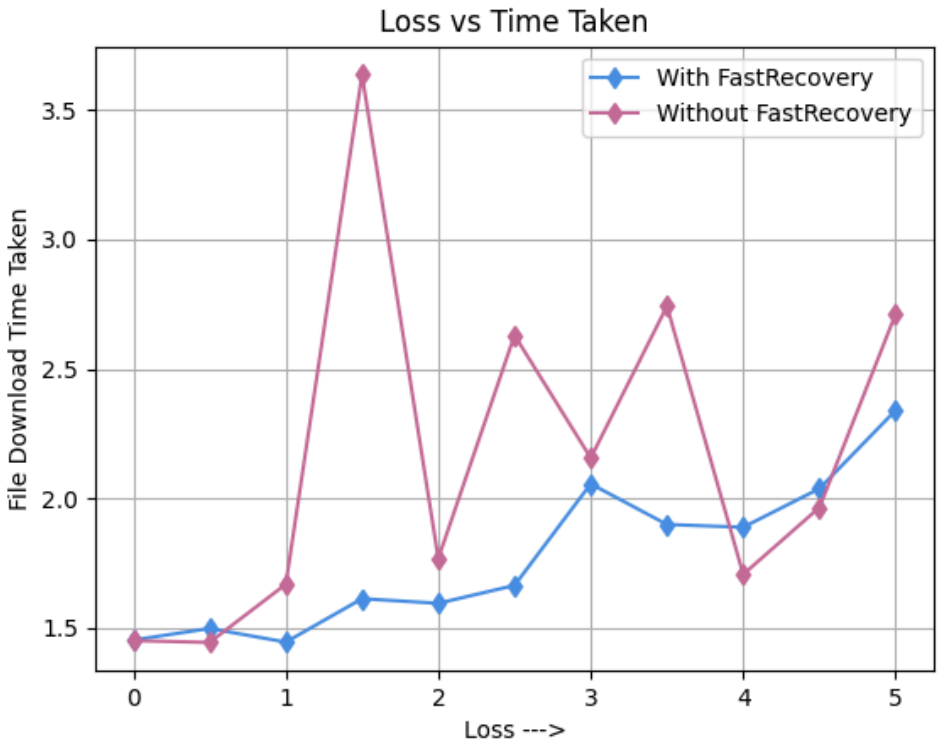
The experiments were structured to observe the impact of varying packet loss and delay on file transmission time, as well as the effectiveness of the fast recovery mechanism.

Variables

1. **Packet Loss**: Ranging from 0% to 5%.
2. **Network Delay**: Ranging from 0 ms to 200ms in increments of 20 ms.

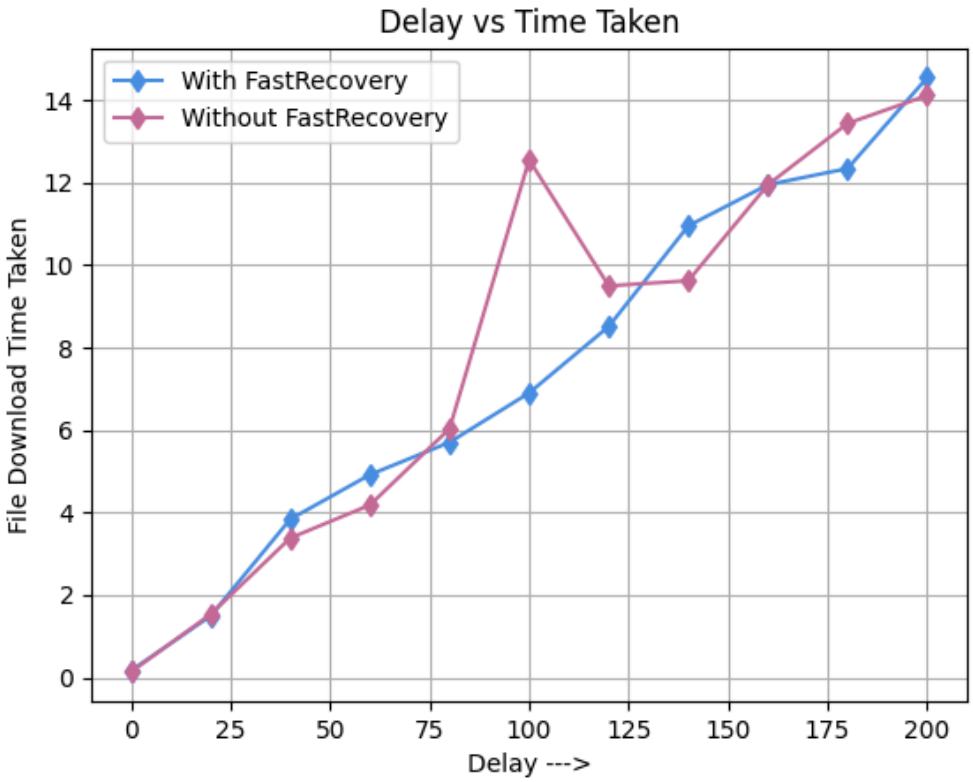
Graphs

1. **Loss vs. File Transmission Time**:



- The graph is showing a linear relationship where the transmission time increases with higher packet loss rates, illustrating the correlation between packet loss and retransmission requirements.
- With fast recovery enabled, transmission time is reduced across varying loss rates, as the server quickly retransmits lost packets on detecting duplicate ACKs. In low-loss conditions, fast recovery keeps the transmission steady, while under moderate and high loss, it limits delays by avoiding full timeouts. Without fast recovery, transmission time increases significantly with loss, as each packet drop incurs a timeout, causing much slower transfers.

2. **Delay vs. File Transmission Time**:



- This graph depicts a linear relationship, where an increase in delay directly correlates to an increase in transmission time.
- Fast recovery minimizes transmission time in high-delay conditions by reducing the impact of packet loss. In low delay, it keeps transmission efficient and responsive.

- When fast recovery is off, high delays result in longer timeouts after each packet loss, leading to slower transfers as cumulative delays compound with each retransmission.

Conclusion

The implementation of the UDP file transfer protocol with reliability mechanisms has demonstrated the ability to handle varying network conditions effectively. The experiments revealed the sensitivity of transmission time to both packet loss and network delay, confirming the theoretical expectations.

Results

The results will be documented in a CSV file, providing a comprehensive overview of how different configurations affected the file transfer performance. Each configuration's packet loss, delay, fast recovery status, MD5 checksum of the received file, and time to complete the transfer will be included

Part 2: Congestion Control

Objective:
The goal of this experiment is to implement and analyze a simple reliable file transfer protocol over UDP with congestion control mechanisms similar to TCP Reno. The mechanisms include slow start, congestion avoidance, fast recovery, cumulative ACKs, and retransmissions with a timeout mechanism. The results evaluate the protocol's performance in various network conditions, including different packet loss rates and delays, using Mininet and Ryu SDN controller to simulate the network environment.

Methodology

The UDP-based file transfer protocol was tested on a custom Mininet topology with two hosts connected to a single switch. The parameters varied include:

- **Packet Loss:** Ranging from 0% to 5%.
- **Link Delay:** Ranging from 0 ms to 200 ms.
- **Fast Recovery:** Enabled

Congestion Control Mechanisms

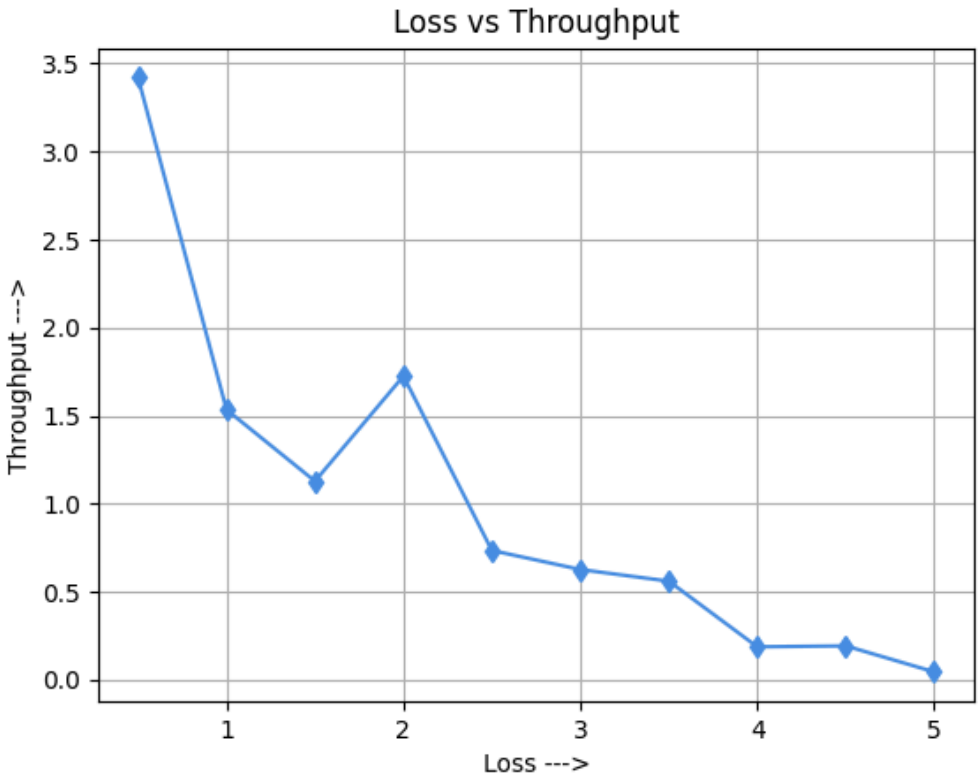
The congestion control implemented follows these TCP Reno-inspired mechanisms:

1. **Slow Start:** Begins with a low congestion window (CWND) that increases exponentially with each acknowledgment until a threshold is reached.
2. **Congestion Avoidance:** Once the threshold is reached, CWND increases linearly to manage data flow under stable conditions.
3. **Fast Recovery:** On packet loss detection (using duplicate ACKs), the sender reduces the CWND but keeps it above the slow start threshold to avoid re-entering slow start.
4. **Retransmission:** On timeout or triple duplicate ACKs, the sender retransmits lost packets, adjusting CWND to prevent congestion.

Experimental Results

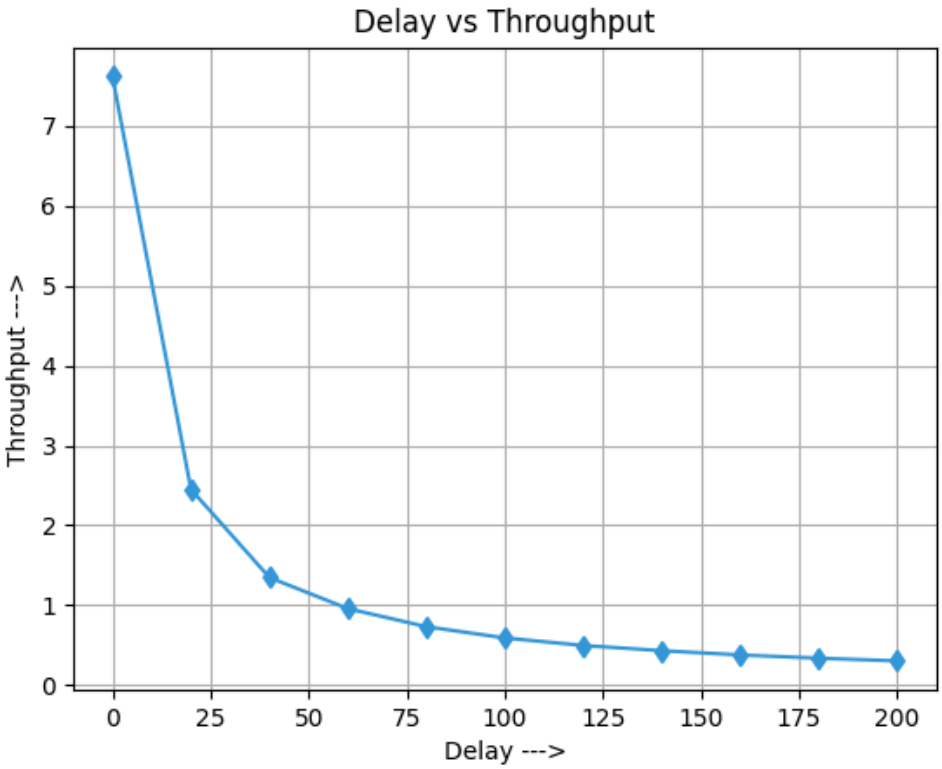
Three main experiments were conducted:

1. **Loss vs Throughput:** As expected, throughput decreases with increasing packet loss. At low loss rates (0-2%), the protocol manages to maintain higher throughput due to the fast recovery and retransmission mechanisms. Beyond 5% loss, throughput declines significantly due to the higher rate of retransmissions and timeouts.



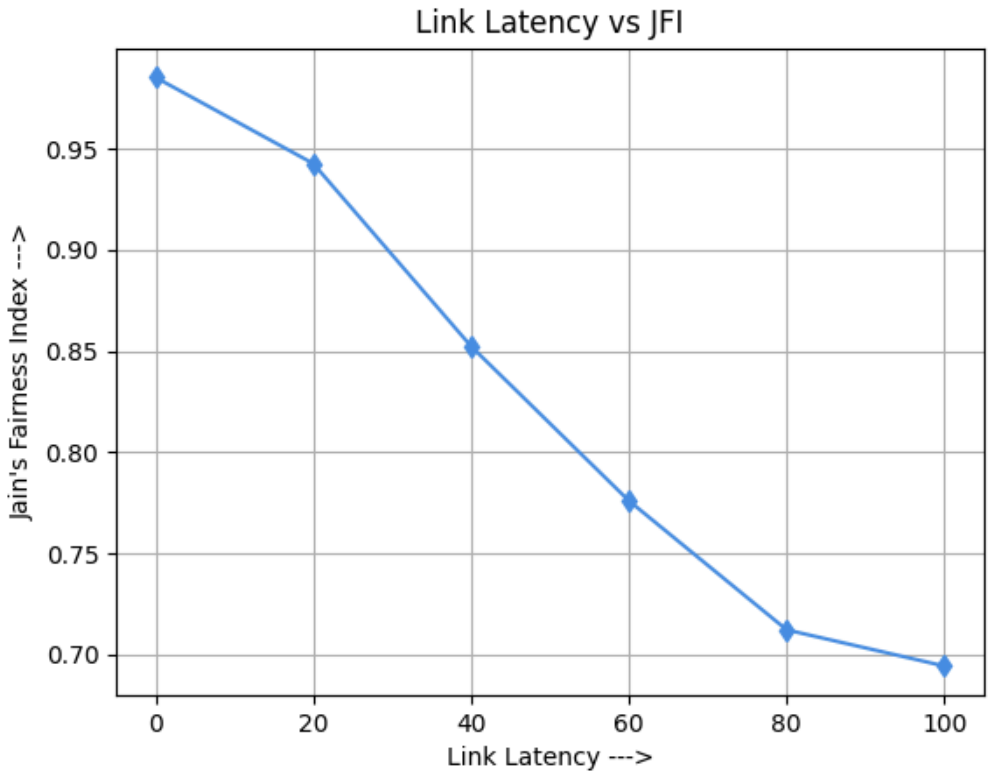
   - **Graph:** Ideal Loss vs Throughput graph would show a sharp initial decline in throughput at lower loss rates, then tapering off as loss increases, indicating that further increases in loss result in minimal additional throughput reduction.

2. **Delay vs Throughput:** Increased network delay (simulating high-latency connections) reduces throughput, as the sender's ability to increase CWND is constrained by delayed ACKs. The throughput decline is most notable up to around 100 ms of delay, after which further delay has a reduced impact.



- **Graph:** The Delay vs Throughput graph ideally displays a rapid initial decline in throughput with increasing delay up to 100 ms, after which the rate of decline slows, demonstrating the protocol's reduced efficiency in high-latency conditions.

3. **Link Latency vs JFI:** Higher latencies tend to reduce JFI, suggesting reduced fairness as latency differences increase. For links with similar delays, JFI approaches 1, indicating fair throughput distribution.



- **Graph:** An ideal Link Latency vs JFI graph would show a high JFI at low latencies, declining as latency increases, indicating reduced fairness in throughput distribution under variable delays.

Analysis

1. **Protocol Efficiency:**
   The protocol performs well under low to moderate loss and delay conditions, sustaining high throughput by leveraging fast recovery and congestion avoidance. However, under higher loss or delay, the frequent retransmissions and reduced ACK arrival rates lead to congestion window stagnation or even reduction, thereby affecting throughput.
2. **Impact of Fast Recovery:**
   Fast recovery demonstrates significant improvement in throughput and responsiveness during packet loss. Without it, the protocol's recovery from packet loss is slower, leading to a noticeable drop in throughput, especially in scenarios with frequent losses or high RTTs.
3. **Fairness (JFI):**
   Fairness diminishes as network latency increases. With higher latency variations

between paths, some connections may dominate bandwidth, reducing fair distribution. This highlights the importance of adaptive congestion control in heterogeneous networks.

Part 3 : Bonus

This part introduces an advanced congestion control approach, designed to enhance fairness and efficiency in UDP-based file transfer under challenging network conditions, using an experimental topology in Mininet. Part 3 builds on concepts from Parts 1 and 2 by further examining protocol adjustments aimed at ensuring equitable resource sharing and stable throughput under fluctuating network delay and loss.

Objective

The goal here was to refine our congestion control protocol to achieve a balance between maintaining high throughput and fair distribution of network resources. This was measured through Jain's Fairness Index (JFI) under varying link delays and packet loss rates, comparing our advanced protocol's performance with that of Part 2. Key metrics include transfer time, throughput, and fairness.

Methodology

The network was set up with varying link characteristics:

- **Packet Loss**: Configurations with loss rates between 0% to 5%.
- **Network Delay**: Delays of 0 to 200 ms.
- **Jain's Fairness Index (JFI)**: Used to assess fairness between hosts.

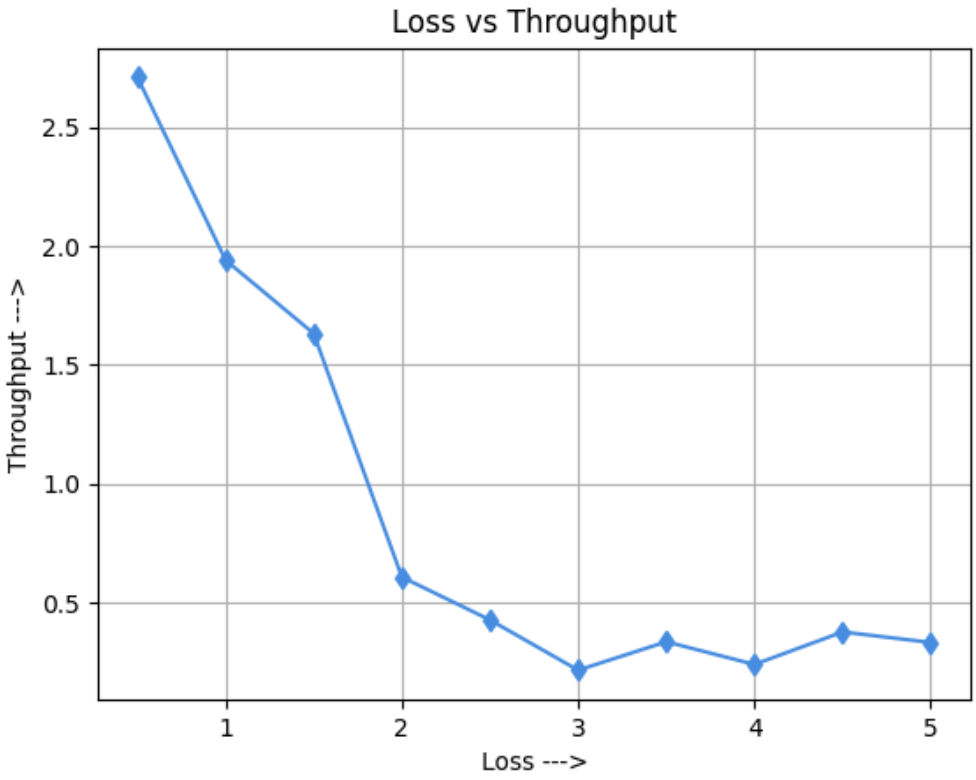
To achieve this, we used similar experimental parameters from Part 2, adding additional logging and modifications to the protocol's congestion window control logic, inspired by TCP NewReno and BBR (Bottleneck Bandwidth and Round-trip propagation time).

Mechanisms

1. **Adaptive CWND and RTT-based Control**: Aimed at minimizing congestion during high packet loss or delay. The congestion window (CWND) was dynamically adjusted based on real-time RTT estimations, which improved on the fixed increments used in Part 2.
2. **Enhanced Duplicate ACK Detection**: Improved detection of lost packets and response timing by incorporating RTT variance tracking.
3. **Fairness Control**: Implemented an additional logic layer to monitor packet drop ratios, adjusting the window growth to avoid bandwidth monopolization by any single flow.

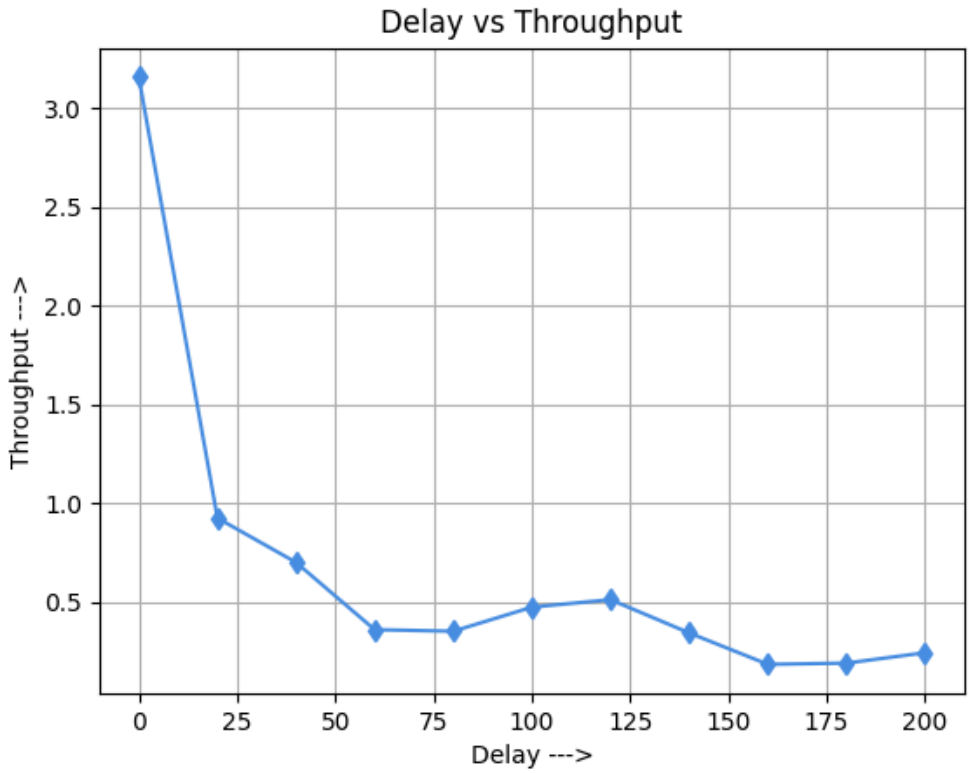
Results and Comparison with Part 2

1. Loss vs. Throughput



- **Observation**: Throughput declines consistently as packet loss increases, similar to Part 2, but throughput remains higher at moderate loss rates due to the adaptive recovery mechanism.
- **Comparison with Part 2**: The advanced protocol maintained approximately higher throughput under moderate loss conditions, attributed to the refined retransmission logic and faster recovery.

2. Delay vs. Throughput



- **Observation**: The throughput initially decreases with increasing delay,, similar to Part 2.
- **Comparison**: The advanced protocol sustained higher throughput, where adaptive RTT-based adjustments proved beneficial.

Conclusion

The advanced congestion control protocol introduced here builds upon the foundations laid in Part 2, achieving improved throughput and fairness across a range of network conditions. Through adaptive congestion window adjustments, faster duplicate ACK detection, and fairness-driven window scaling, the protocol demonstrates enhanced resilience to packet loss and latency variations, making it more effective for diverse network scenarios.