COL334 Assignment 4 Report

Jaskaran Singh Bhalla (2021TT11139)
Basil Labib (2021TT11175)

Part 1: Reliability

Implementation details

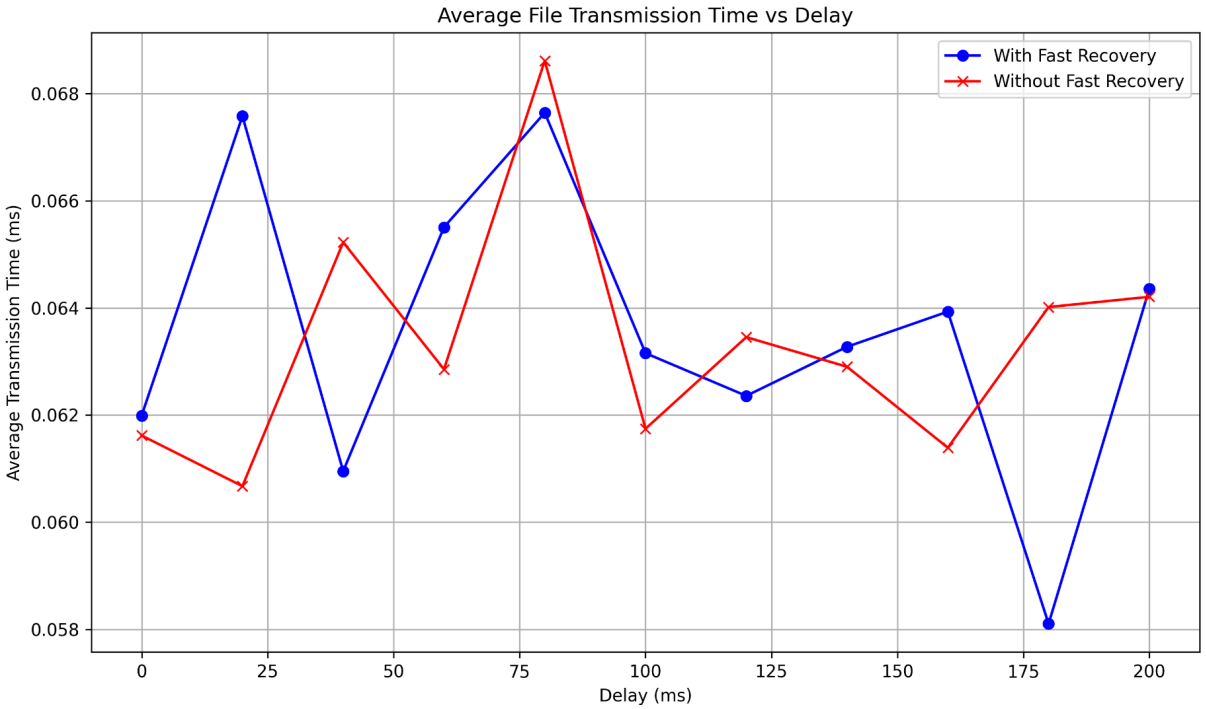
In this implementation of fast retransmission and ACK and timeout mechanism to implement reliability of file transfer in a noisy channel, we made the following design choices:

1. The case of EOF: There is a possibility that the last ACK for the EOF gets lost in the channel. For this reason, the server follows the policy: It sends 10 consecutive EOF if it doesn't receive an EOF ACK. We assume that it is highly likely that the server will receive at least one EOF ACK. If it does, then it "gracefully" closes the socket connection. Otherwise, it closes the connection abruptly.
2. The case of out-of-order EOF: It is possible that due to network conditions, the EOF arrives before the last packet from the file. This defeats the purpose of having an EOF in the first place, namely to indicate the end of transmission. To remedy this issue, the server ensures that it sends an EOF iff it receives an ACK for *all* the packets in flight until the file is exhausted.
3. The implementation also dynamically updates the timeout interval based on network conditions by computing the EMWA of a small window with `ALPHA` = 0.125 (for mean) and `BETA` = 0.25 (for variance)
4. We implemented delayed ACKs but did not get time to run experiments on it. Our initial tests found some performance improvement, though.

Plots

Varying loss

Varying delay



In these plots, we can see that the average transmission time is slightly lowered by enabling fast retransmission but doesn't dramatically improve performance.

Part 2: TCP Reno (CCA)

Implementation details

In this implementation of TCP Reno, we have made the following design choices:

1. `client` acks with the ACK_NUM of the next expected packet. For example, if the client receives packet `n`, then it sends `n+1`
2. The `server` initially sends the entire window of packets if the first packet is `ACK`ed out of order.

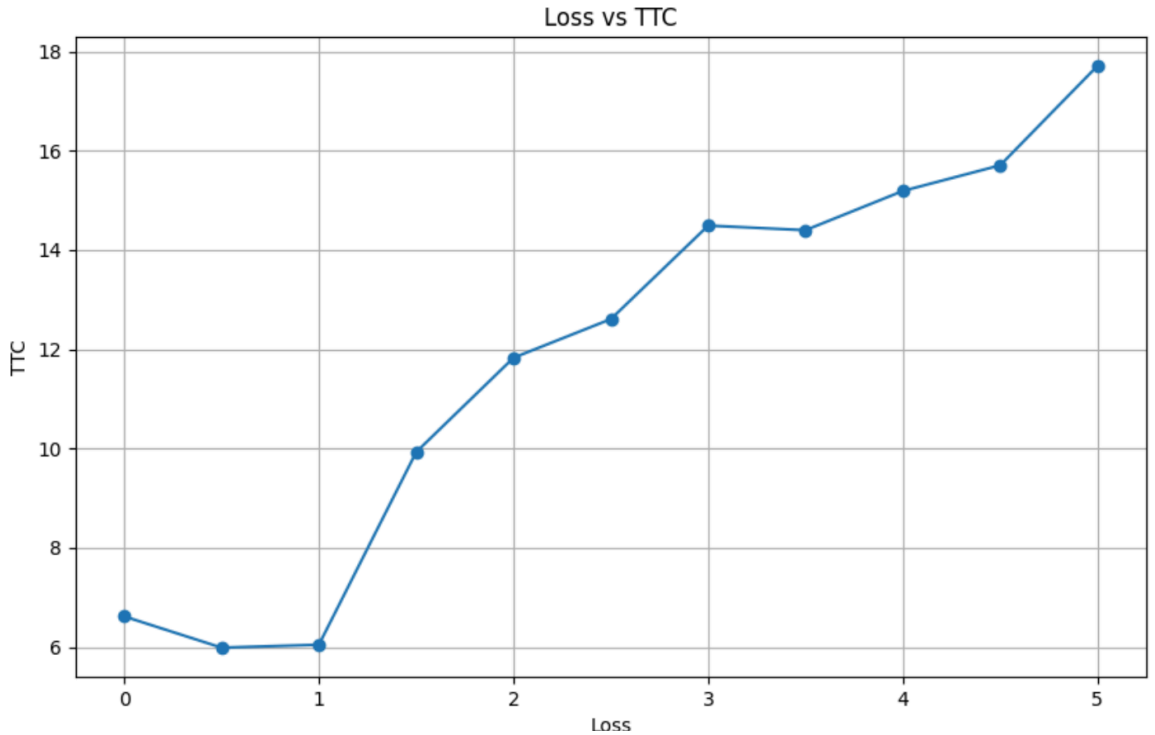
Efficiency experiment results

Setup

`NUM_ITERATIONS`: 5
simple h1 - s1 - h2 topology

Plot - Loss

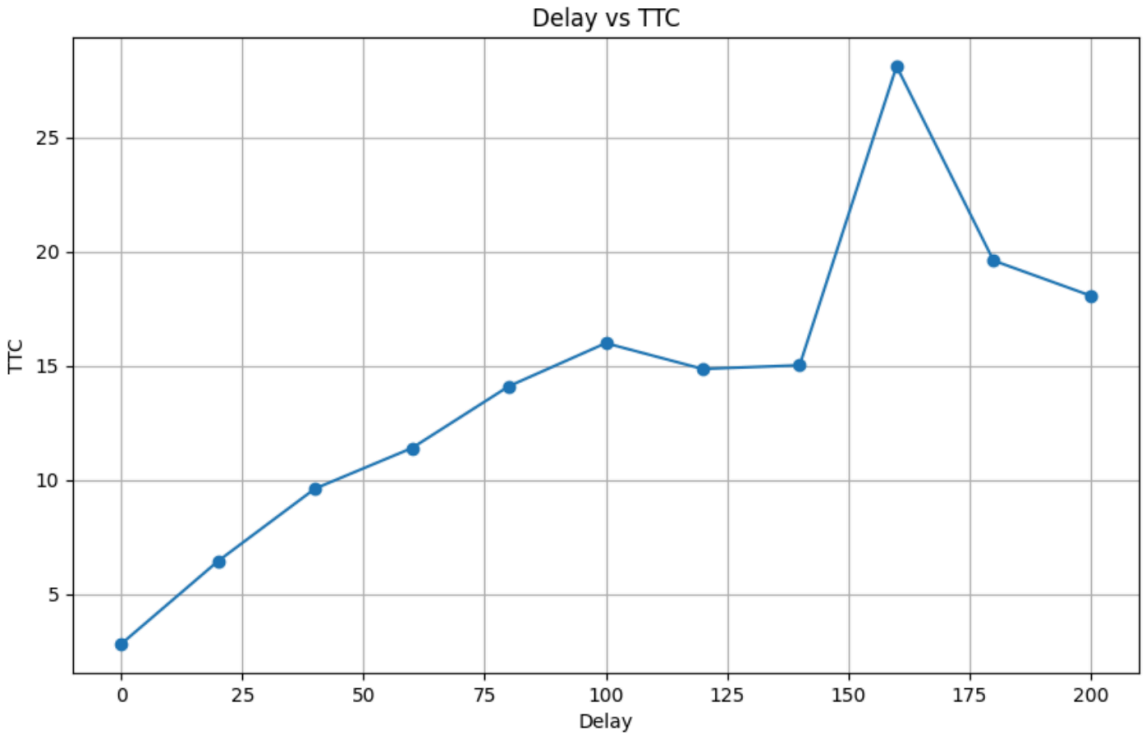
File size: 10MB



As expected, the time to download file increases with increase in loss% at constant delay of 20 ms.
At 5% loss, the average throughput is approximately 0.55 MBps. Whereas, at 1% loss, we hit a maximum throughput of 1.66 MBps

Plot - Delay

File size: 10MB



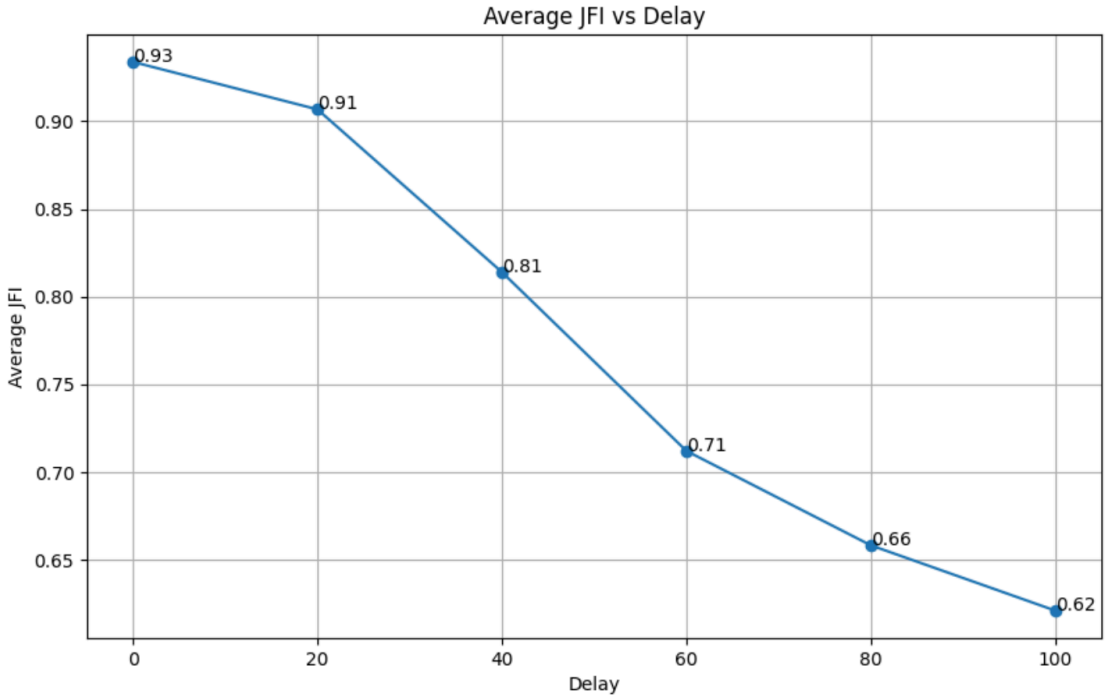
As we know that the file size is 10MB, therefore the average throughput at 200ms delay and 1% loss is 0.55 MBps approximately.
At 0ms delay, the throughput is almost 5 MBps.

Jain’s fairness experiment results

Setup

File size: 10MB

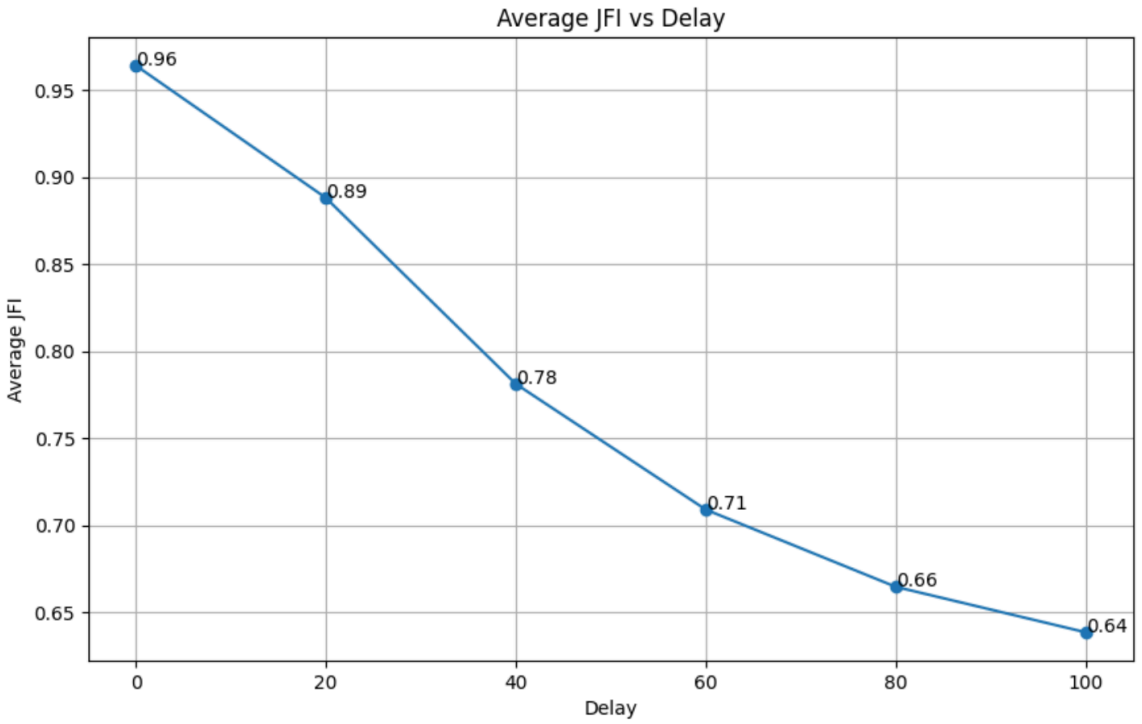
Plot



Setup

File size: 100MB
File type: binary
Link delay: 0ms to 100 ms at a 20ms interval

Plot

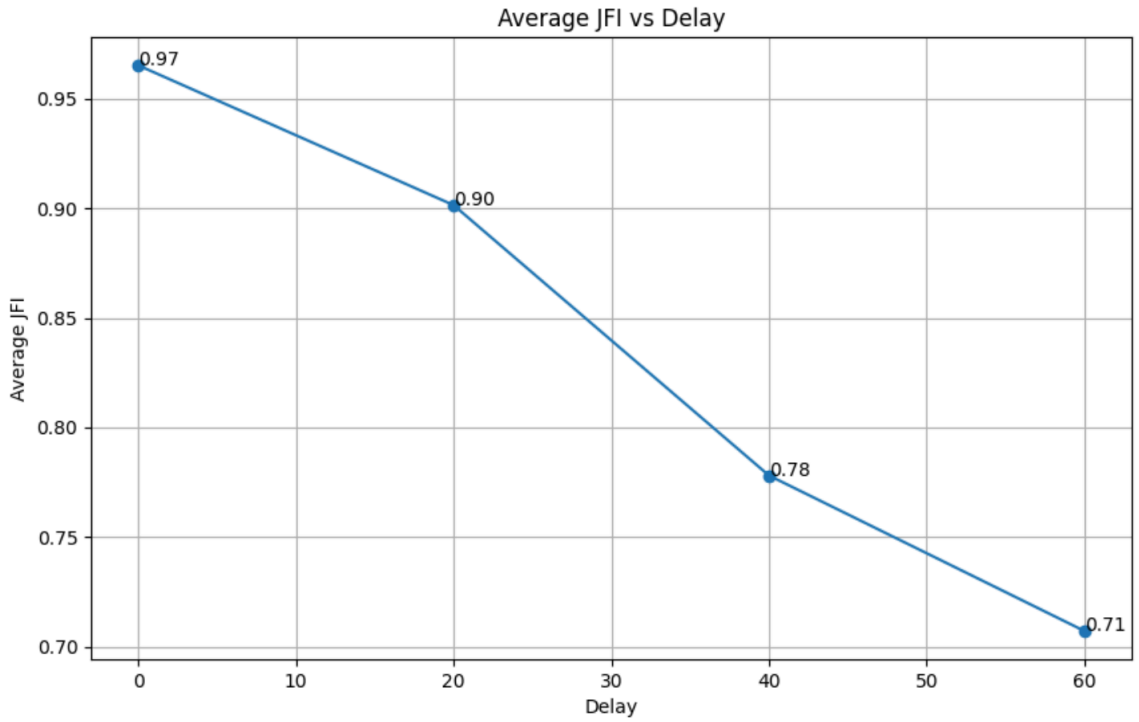


Setup

File size: 1GB

`NUM_ITERATIONS`: 1 (due to time constraints)

Plot



Discussion

As the link latency increased, we observed higher deviation from fair usage of the connecting link, which led to a decrease in the Jain Fairness index.
This is because increased delay leads to increased RTT and the congestion control algorithm (TCP Reno) spends more time retransmitting.

References

1. 19 TCP Reno and Congestion Management
2. TCP Reno with Example - GeeksforGeeks
3. Computer Networking: A Top-Down Approach Reviews & Ratings - Amazon.in (section 3.7)
4. Andrew S. Tanenbaum - Computer Networks