COL334 - Computer Networks
Assignment 4: TCP-like UDP

Nikhil Zawar - 2022CS11106
Yash Bansal - 2022CS51133

Contents

1 Reliability

1.1 Implementation Details

1.1.1 Client

- Initially, the client starts by sending a **START** message to the server, which also contains the client's IP address for further conversation. The client keeps sending the start message until the client receives the server's acknowledgement **CONNECT**.
- After successfully starting the communication, the client waits for the packets from the server and sends the acknowledgement packets as the packets arrive.
- If some packets arrive out of order, the client buffers the packet and sends the acknowledgement of the smallest sequence number of unarrived packets.
- Once the client receives the lost packets when the server sends them again, the client removes the necessary packets from the buffer in order and sends the cumulative acknowledgement of the last in-order packet.
- When the client receives an **END** packet from the server, it sends an end acknowledgement **END_ACK** to the server and closes the connection.

1.1.2 Server

- Initially, the server listens for the **START** message from the client so that it can have its IP address for further communication. When the packet is received, it sends the **CONNECT** to the client and then sends the further packets.
- The server sends the packets according to the space available in the sliding window and then waits for the acknowledgements of the packets.
- When acknowledgements are received and are in the correct order, it updates the sliding window base and sends the further packets.
- In fast recovery mode, when the acknowledgement threshold is reached, the server sends the lowest unacknowledged packet again, hoping that the packets after that must have arrived.
- If the acknowledgement timeout is reached, it resends all the unacknowledged packets again.
- When all the packets are sent, and the acknowledgements of all are received, it sends the **END** packet to the client until its acknowledgement is received and then closes the connection successfully. The end is sent almost three times to prevent getting stuck in an infinite sending loop.

1.2 Graphs, results and explanation

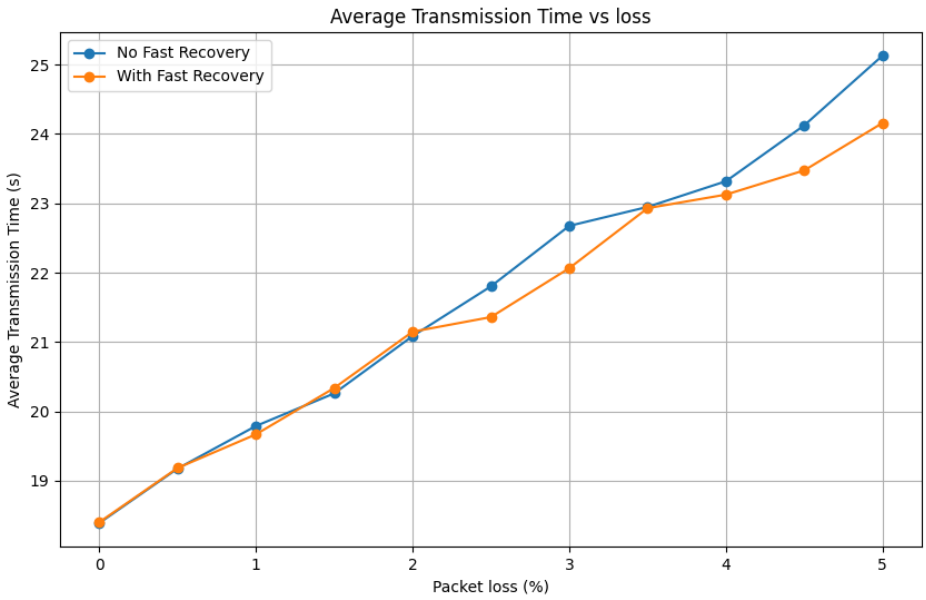


Figure 1: Transmission time vs % loss for reliable transmission

Above is the graph to show the dependence of transmission time on % loss on the link and the fast recovery/ no fast recovery mode of TCP reliable transmission. The link delay is constant at 20ms in each case, while the loss varies from 0 to 5% in increments of 0.5%. Each experiment is conducted 5 times, and the average value is taken to account for noises.

- **Dependency on % loss:-** As the % loss increases in the links, the total transmission time for both fast recovery and no fast recovery modes increases. This is due to more packet losses.
- **Dependency on fast recovery mode:-** The transmission time for fast recovery mode is generally lower than that of no fast recovery mode. The difference is enhanced as the % loss of the links increases.
- This is because in fast recovery, after detecting 3 duplicate acknowledgements, the server senses that just the next packet might be lost, and further packets have arrived successfully. So, rather than waiting for timeout occurrence to retransmit the unacknowledged packets, it resends the packets immediately, thus taking overall less transmission time than fast recovery mode.

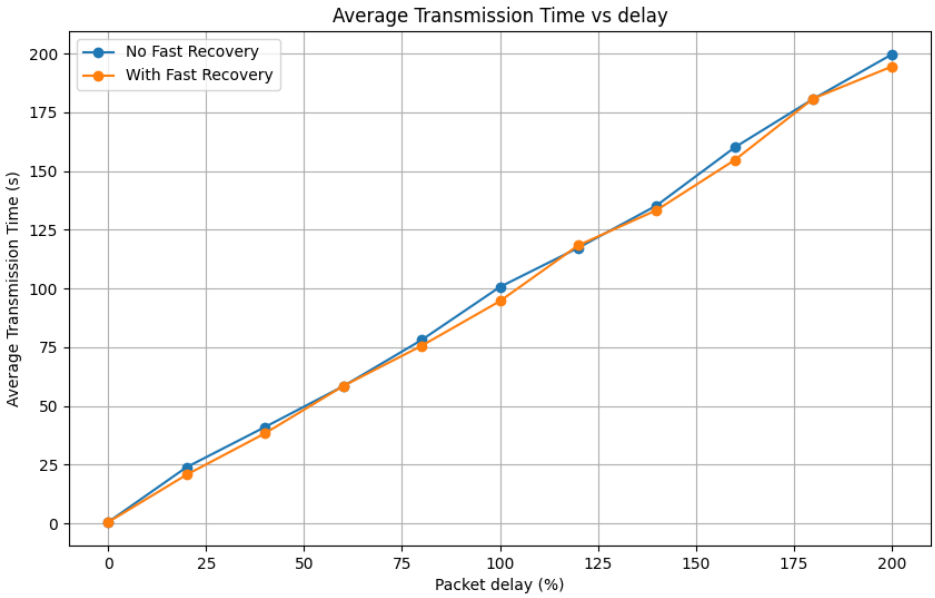


Figure 2: Transmission time vs link delay for reliable transmission

Above is the graph showing the dependence on transmission time vs link delay in the circuit and on fast recovery/ no fast recovery mode of TCP reliable transmission. Link loss is constant at 1%, while the delay varies from 0 - 200 ms, incrementing by 20 ms. Each experiment is conducted 5 times, and average transmission time is taken to account for noises.

- **Dependency of link delays:-** As the link delay varies, the transmission time varies almost linear with the link delay for both fast recovery and no fast recovery mode. This is because the time taken by each packet should vary linearly with the link delay. Thus, overall time also varies linearly.
- **Dependency on fast recovery mode:-** As in the previous case, fast recovery mode generally performs better than no fast recovery mode in TCP reliable transmission.

2 Congestion Control:- TCP Reno

2.1 Implementation Guidelines

The implementation for congestion control is inspired by the TCP Reno protocol. There are three states that we shuttle between for the sake of congestion control - slow start, congestion avoidance and fast recovery.

Since this is an end-to-end protocol, there are no network layer services that tell the transport layer about how congested the network is. The transport layer has to identify this on its own, using RTTs, timeouts, or packet losses.

2.1.1 The following are congestion control states:

Initially, CNWD starts small and grows as ACKs confirm successful packet delivery. A variable named 'mode' is used to denote the state of the network at any given moment.

- **Slow Start (Mode 0):** Begins when a connection is established, with CNWD set to 1. Each ACK received increases CNWD by 1 until reaching ss_threshold, aiming for rapid growth to utilize the available bandwidth. The net increase in this case is the exponential increase. For example, if the present cwnd = 4, then 4 packets are sent, and when corresponding 4 ACKs are received, for each ACK the cwnd size is incremented by 1. So final size of cwnd = 8, which implies that the size of cwnd doubles every iteration.
- **Congestion Avoidance (Mode 1):** Activated when CNWD reaches or exceeds ss_threshold. Here, CNWD increases more slowly (by 1/CNWD per ACK), reflecting a cautious approach as network capacity is likely near saturation.
- **Fast Recovery (Mode 2):** Entered upon receiving three duplicate ACKs, indicating potential packet loss. The ss_threshold is set to half the current CNWD, and CNWD is set to ss_threshold + 3, allowing for a quicker recovery by retransmitting unacknowledged packets. The 3 is added to keep in account the actual packets corresponding to the 3 duplicate acknowledgments.

2.1.2 State Transition

The states of congestion control - slow start, congestion avoidance, and fast recovery – are all visited by different actions in the network.

The fast recovery state is only reached when three duplicate ACKs are received, denoting the network is heavily congested and the server should go back to sending just one packet(cwnd = 1).

The cnwd controls the rate of transmission of the server. The change in the value of cwnd is controlled by both - slow start and congestion avoidance states. When the congestion is less(cwnd * 2 $\leq$ ss_threshold), then the network is in the slow start state, where the size of the cwnd increases exponentially. When the congestion is near full(cwnd $\geq$ ss_threshold) then the increase in the size of cwnd is slower - incremented unit-by-unit.

The following is the **Finite-State-Machine** used for state transition:

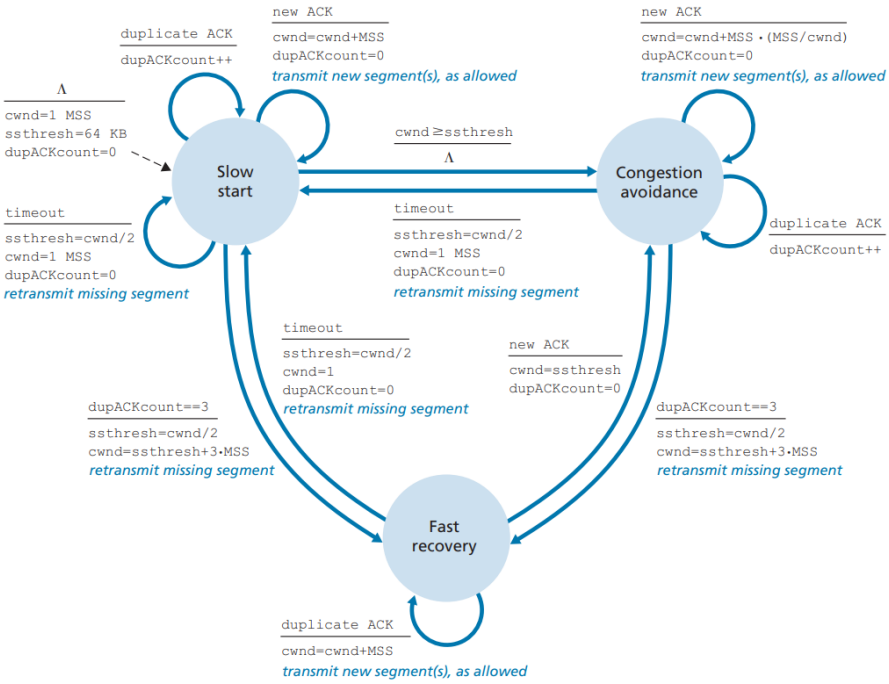
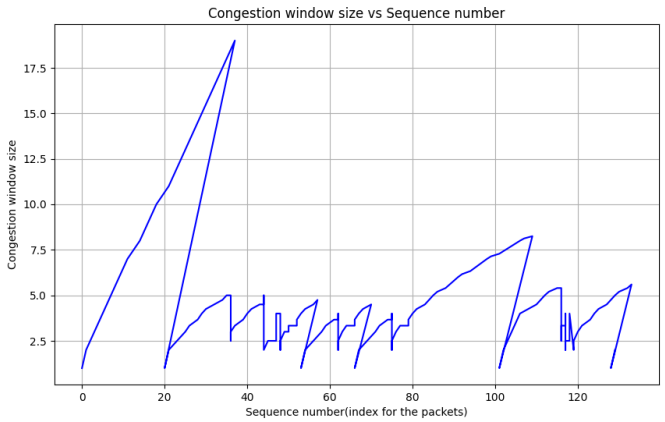


Image Source: [Computer Networking: Top-Down approach by Jim Kurose]

2.1.3 Observing the CWND vs Sequence Number

The sequence number is the indication of the index in the input test file. The way the congestion window increases and decreases with time is shown below. This graph matched the expected pattern of the size of the congestion window size.



2.2 Throughput Analysis

2.2.1 Average Throughput vs Link Delay

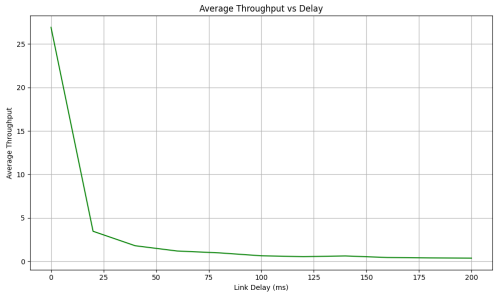


Figure 3: Average throughput(MB/s) vs Link Delay(s)

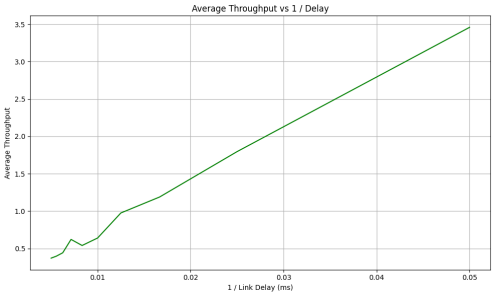


Figure 4: Average throughput(MB/s) vs 1/Link Delay(s)

Observation and Analysis As packet delay increases, packets take longer to travel across the link, this will reduce the rate at which data is transmitted. This slower transmission rate results to reduced throughput since the link can handle less data per unit time.

$$throughput \propto \frac{1}{delay}$$

The trend of reduced throughput at higher delays is particularly steep at lower delay values because, at these levels, even small increases in delay cause relatively significant impacts on throughput. When you plot throughput against 1/delay, the resulting linear line indicates that throughput is inversely proportional to delay. This plot confirms that as delay increases, throughput falls in a way that is predictable and aligns with the

With rising packet delay, congestion control starts slowing down the sending rate to avoid further congestion. This mechanism further contributes to the decrease in throughput.

2.2.2 Average Throughput vs Packet Loss(%)

Plot

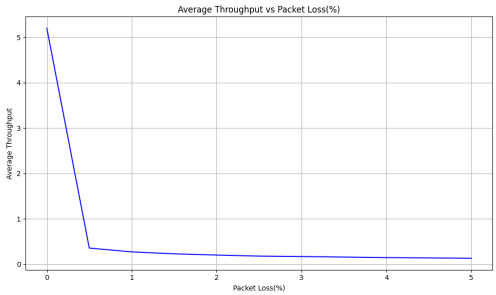


Figure 5: Average throughput(MB/s) vs Packet Loss %

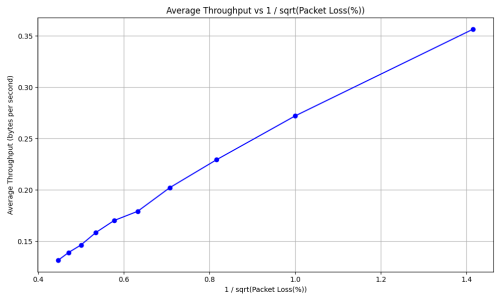


Figure 6: Average throughput(MB/s) vs $1/\sqrt{(PacketLoss\%)}$

Observation and Analysis

As the packet loss percentage increases, the average throughput decreases. This trend highlights an inverse relationship between throughput and packet loss rate, indicating that higher packet loss significantly reduces the efficiency of data transfer.

Packet loss in a network forces the transmission protocol to re-send lost packets to ensure complete data delivery. Each retransmission occupies bandwidth, decreasing the effective throughput. TCP congestion control adjust their sending rate in response to packet loss, which further contributes to the reduction in throughput. We will try to plot the throughput and the reciprocal of square root of the packet loss rate. We can conclude on the following trend

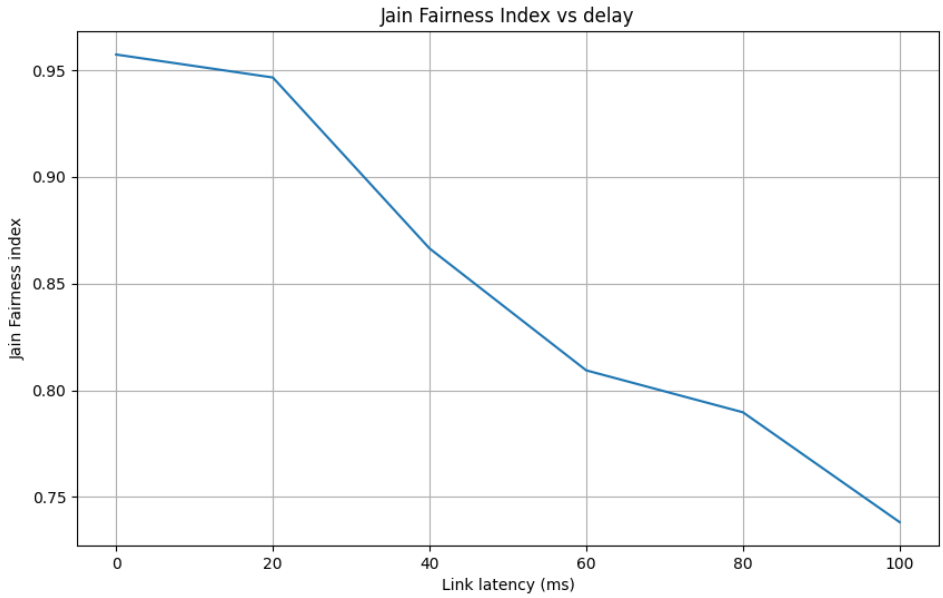
$$throughput \propto \frac{1}{\sqrt{loss}}$$

The resulting linear trend in the plot supports the assumption, showing a clear inverse square-root relationship between throughput and packet loss.

This trend is more important at lower packet loss rates, where even a small increase in loss causes a considerable decrease in throughput. At higher loss rates, the throughput reduction continues but at a slower rate, reflecting the non-linear nature of the inverse-square-root dependency.

2.3 Jain's Fairness Index

2.3.1 Plot



2.3.2 Observation and Analysis

The plot for the Jain's Fairness Index vs the Network Delay, shows that as the network delay increases, the fairness reduces.

The fairness index is a value between 0 and 1, where 1 indicates perfect fairness, meaning equal distribution among flows, and 0 indicates large inequality.

At lower delays, the JFI is generally large and near 1. This suggests that when the link delay is small in the switch2-s2 connection, then the network flows have a fair distribution of resources.

As the delay increases, the JFI drops to a low value of 0.6, which suggests that now there is a disparity in the utilization of network resources. Some flows experience significantly more delay than others.

This same thing can be concluded otherwise also if we look at the Time to Completion(TTC). The TTC1 and TTC2 values differ from a very high magnitude when the link delay is 100ms. This difference increases the unfairness among flows, reducing the JFI.