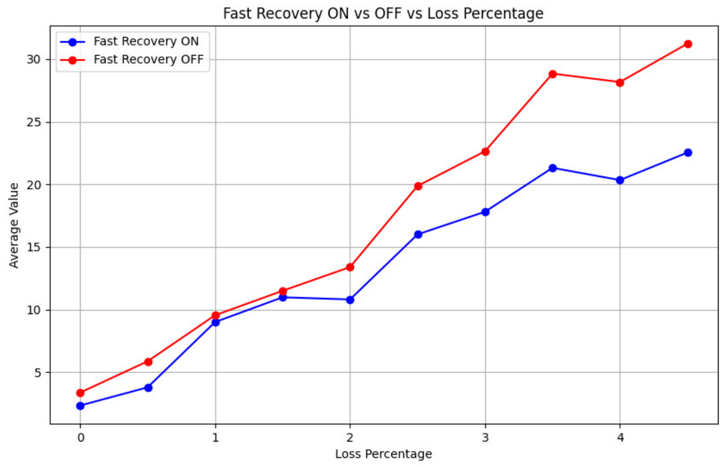
TCP like UDP

Umesh Kumar-2022CS11115
Aditya Sahu-2022CS11113
COL334:- Computer Networks

1 Part 1: Reliability

In this part, reliability mechanisms are implemented over UDP to ensure dependable file transfer from server to client.

1.1 Plot 1



Average time taken vs loss percentage

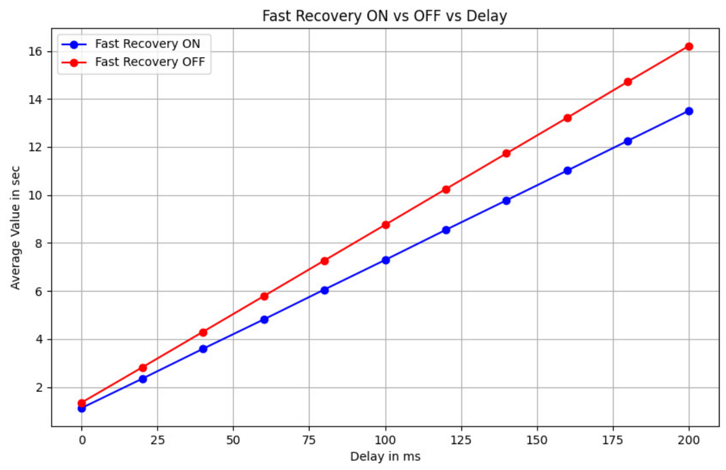
This plot shows the average value of time taken for transfer of file with reliability implemented versus the loss percentage. The size of the file used for making this plot was approximately 2MB and the delay value was kept constant at 20ms.

Some trends and explanations are:

- **Comparison:** Both the graphs are increasing. However, at low loss rates, the difference is small. As loss increases beyond 2%, fast recovery becomes much more effective, keeping transfer times significantly lower.

- **With Fast Recovery (Blue Line):** Transfer time increases gradually as packet loss rises. Fast recovery enables prompt retransmissions, reducing delays caused by packet loss.
- **Without Fast Recovery (Red Line):** Transfer time increases sharply with higher packet loss, as retransmissions rely on timeouts, leading to cumulative delays.

1.2 Plot 2



Average time taken vs link delay

This plot shows the average value of time taken to transfer the file in fast recovery and not fast recovery condition versus the delay imposed.

Some general trends and explanations:

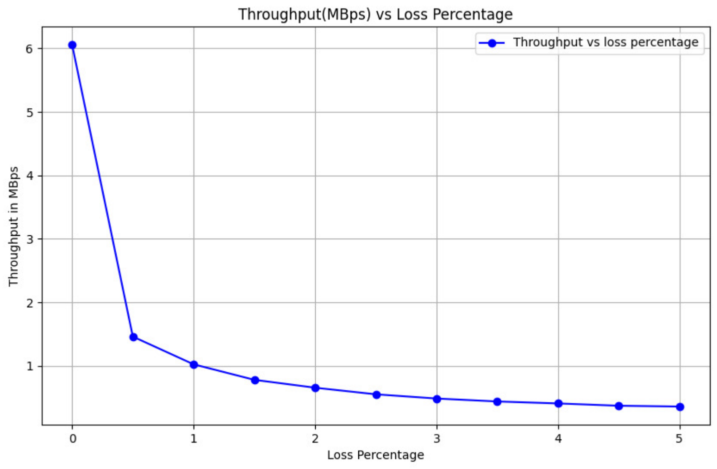
- **Comparison:** The graph is linearly increasing with the delay and file transfer takes less time when Fast recovery is ON. The total file transfer time increases proportionally with network delay because each packet-ACK cycle is directly impacted, leading to a linear relationship in the graph.
- **Effect of Fast Recovery (Blue Line):** Even with increased delay, fast recovery consistently achieves lower transfer times than without fast recovery.

2 Part 2: Congestion Control

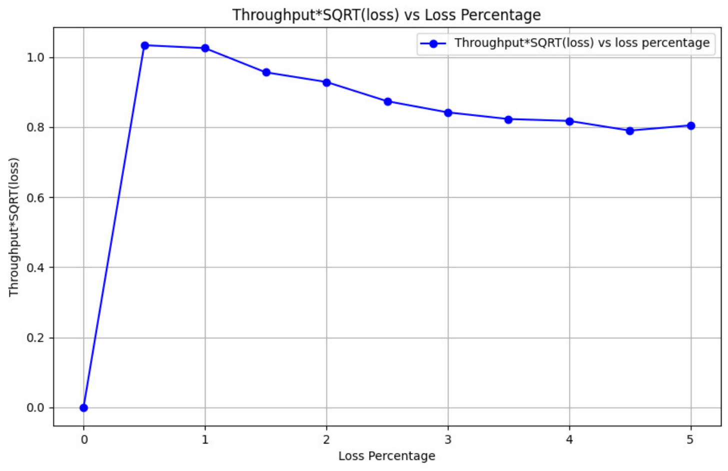
2.1 Efficiency

2.1.1 Plot 1

The graph shows the relationship between throughput (in MBps) and packet loss percentage for a TCP connection implementing TCP Reno-like congestion control.



Throughput in MBps vs loss percentage



Constant value of (Throughput multiplied by $\sqrt{p}$)

Some observable trends and explanations are:

- **High Throughput at Low Packet Loss (0% Loss):** At 0% packet loss, the throughput is approximately 6 MBps, which is the highest observed value in the graph.

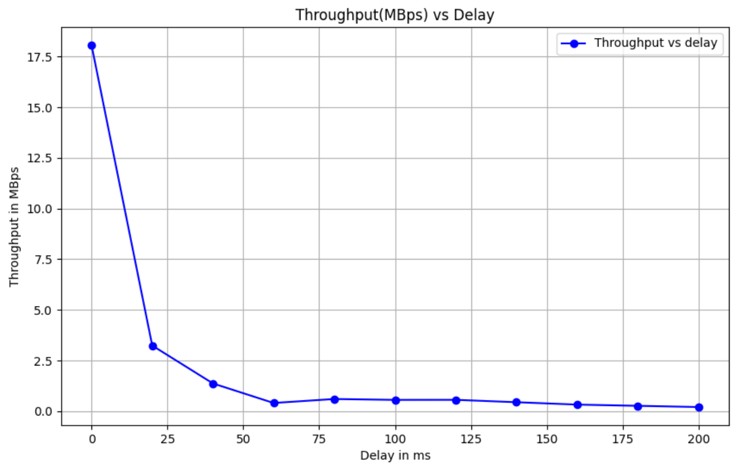
- **Significant Drop with Minimal Loss:** As the packet loss increases slightly to 0.5% and 1%, there is a sharp decrease in throughput. This sharp decrease is a result of TCP Reno's sensitivity to packet loss, which triggers congestion control mechanisms like reducing the congestion window by half and entering congestion avoidance mode, limiting throughput growth.

- **Gradual Decline with Higher Loss Rates:** Beyond 1% packet loss, the decline in throughput becomes more gradual, but the throughput remains consistently low, approaching around 0.5 MBps by the time the loss rate reaches 5%. TCP Reno's performance is heavily affected by packet loss due to its additive increase, multiplicative decrease (AIMD) approach, which restricts throughput growth in lossy networks.

The throughput is approximately inversely proportional to the square root of the loss percentage, which aligns with the classic TCP Reno throughput model: $T \propto \frac{1}{\sqrt{p}}$.

2.1.2 Plot 2

This graph shows the relationship between throughput and link delay at constant 1% packet loss.



Throughput in MBps vs link delay

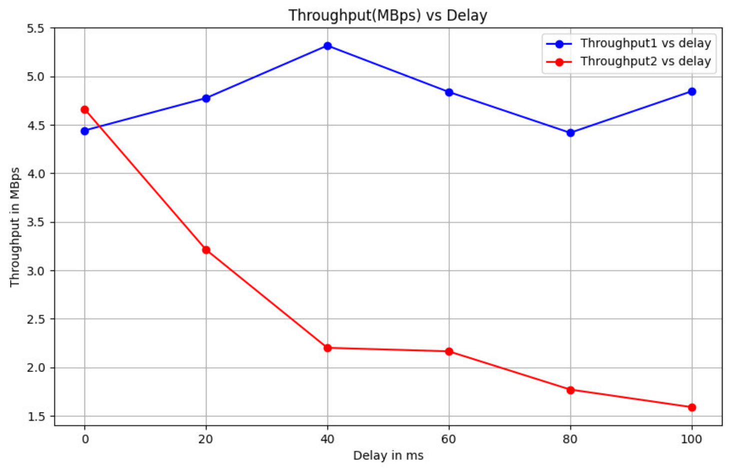
- **High Throughput at Low Delay:** At 0 ms delay, the throughput is approximately 18 MBps, which is the highest observed value in this graph. This reflects ideal conditions for TCP Reno, where the network allows quick transmission and acknowledgment cycles, maximizing throughput.
- **Significant Drop in Throughput with Small Delay Increase:** Even with a small increase in delay to around 25 ms, throughput decreases sharply to below 3 MBps.
- **Minimal Throughput with High Delay:** Beyond 50 ms of delay, the throughput stabilizes at a low value, remaining below 1 MBps up to 200 ms delay.

As link delay increases, RTT also increases, causing throughput to decrease. This is consistent with TCP Reno's behavior in environments with high latency.. Since TCP Reno relies on receiving ACKs to increase the congestion window, longer delays mean slower feedback, limiting the congestion window's growth and consequently reducing throughput. TCP Reno's throughput is inversely proportional to the Round-Trip Time (RTT), so throughput T can be approximated as $T \propto \frac{1}{RTT}$.

2.2 Fairness

Plot 1

This graph is the plot of the throughput of 2 servers when the second server is put on varying link delay from 0ms to 100ms.



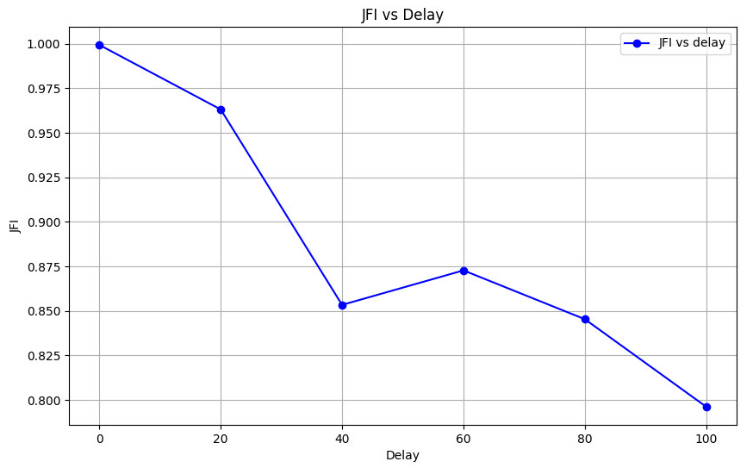
Throughput comparison for server 1 and server 2

The throughput 1 corresponds to 5ms link delay, which is kept constant throughout the 6 iterations. The other server's link delay is varied from 0ms to 100ms at gap of 20ms. The throughput 2 also varies accordingly.

General observations and explanations:

- The closer the link delay of the 2nd server is to 5ms, the closer the throughputs of the 2 servers will be.
- As the delay of the 2nd server link increases, the throughput of the two servers differ largely because the one with lower link delay has greater throughput.
- However the difference in throughput is not in correspondence with the inverse proportional rule for delay. For example, the ratio of throughput 80ms and 5ms is not 16, rather it is much less than that. This shows a fairness in link congestion of the bottleneck link.

The below graph shows the Jain's fairness index plot vs the link delay.



Jain's fairness index vs link delay

The JFI for 0ms and 5ms link delay is very close to 1 which is justified as both the server's are able to send their files in very less time, taking almost equal time. The JFI decreases as the difference between the link delay of the servers increase. As the link delay of server 2 increases its file transmission becomes slow. Thanks to the congestion control, the JFI still remains around 0.8 even when the link delay for sever 2 is 100ms.

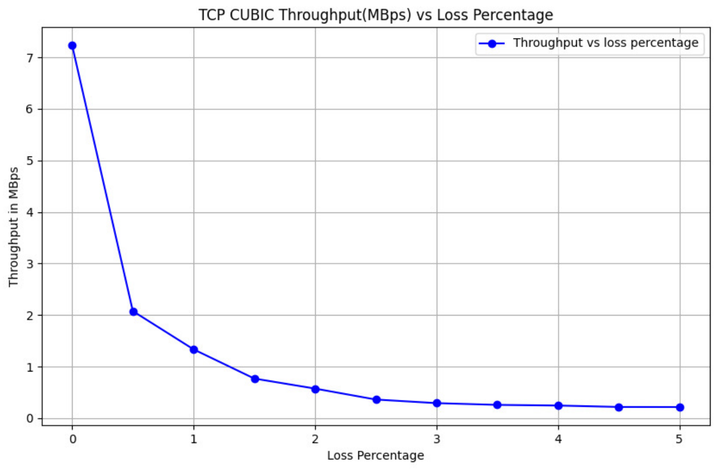
3 Part 3: Cubic Congestion Control

TCP CUBIC is a modern congestion control algorithm optimized for high-bandwidth, high-latency networks. Its cubic function allows for a more aggressive window growth than TCP Reno,particularly in long-delay environments.

3.1 Efficiency

Plot 1

This graph shows the relationship between Throughput and loss percentage for delay 20ms and losses in range 0% to 5%.



TCP CUBIC throughput vs loss percentage

Some general observations and their explanations are:

- The graph for TCP CUBIC congestion control is similar to the TCP RENO, it decreases rapidly with the increasing loss percentage as the time taken for file transfer increases with increasing packet loss percentage.

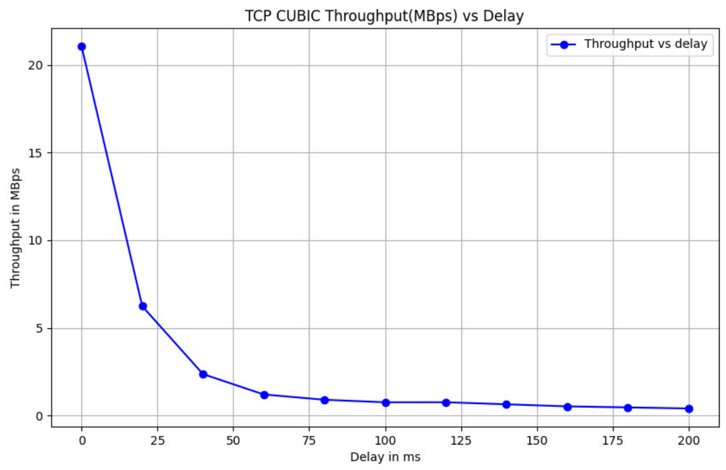
- However TCP CUBIC congestion control in general performs better when the loss percentage of packets is low. The algorithm's cubic growth function allows for aggressive window expansion, which benefits from low-loss conditions.

- This is apparent from the graph where in low loss condition the throughput is more in comparision to TCP RENO. The difference between the throughput of TCP CUBIC congestion control and TCP RENO decreases as loss percentage increases.

- In TCP CUBIC, throughput is inversely proportional to $\frac{1}{p^{0.75}}$, meaning as packet loss increases, throughput decreases, but not as drastically as in TCP Reno. This behavior comes from CUBIC's cubic window growth function, which is designed to recover quickly from congestion while being less sensitive to packet loss.

Plot 2

This graph shows the relationship between Throughput and delay for loss 0% and delays in range 0ms to 200ms.



TCP CUBIC throughput vs link delay

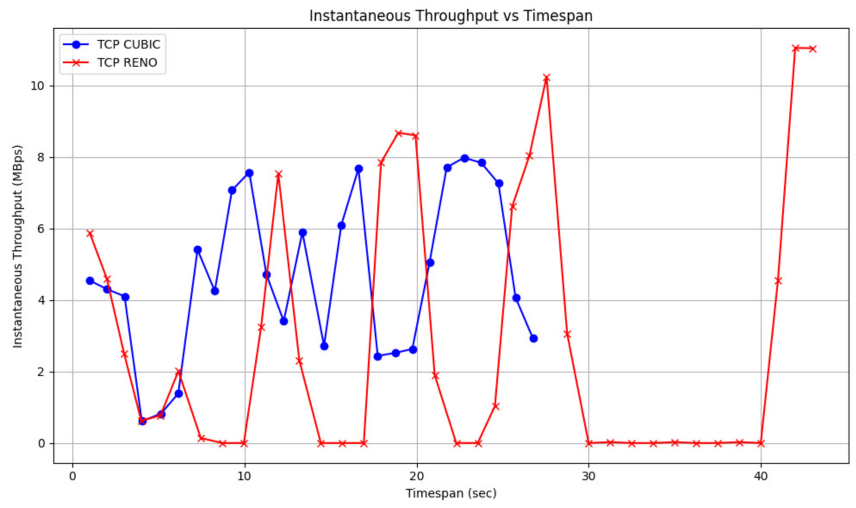
Some general observations and their explanations are:

- The throughput vs Delay graph for TCP CUBIC congestion control is similar to that obtained for TCP RENO apart from the fact that the throughput values a little bit more for the same parameters. This is primarily because TCP CUBIC uses a cubic window growth function, which allows it to more efficiently utilize available bandwidth, especially on high-delay, high-bandwidth networks
- The throughput values are almost hyperbolic , i.e. they are inversely proportional to the link delay values.
- TCP CUBIC is specifically optimized for networks with higher delays (such as long-distance or high-bandwidth-delay product networks). This can also be seen in the graph as the throughput values in the higher delay segment are relatively more in comparison to TCP RENO. This is because TCP CUBIC's cubic growth function is designed to be more aggressive in expanding the congestion window, allowing it to make better use of available bandwidth even in high-delay scenarios.

3.2 Fairness

Plot 1

This is the graph for instantaneous throughput vs the timespan for which the file transfer takes place. The red one is for TCP RENO, and the blue one is for TCP CUBIC congestion control. The graph was taken by keeping the link delays 2ms for each link.



Instantaneous throughput for TCP CUBIC and TCP RENO at 2 ms

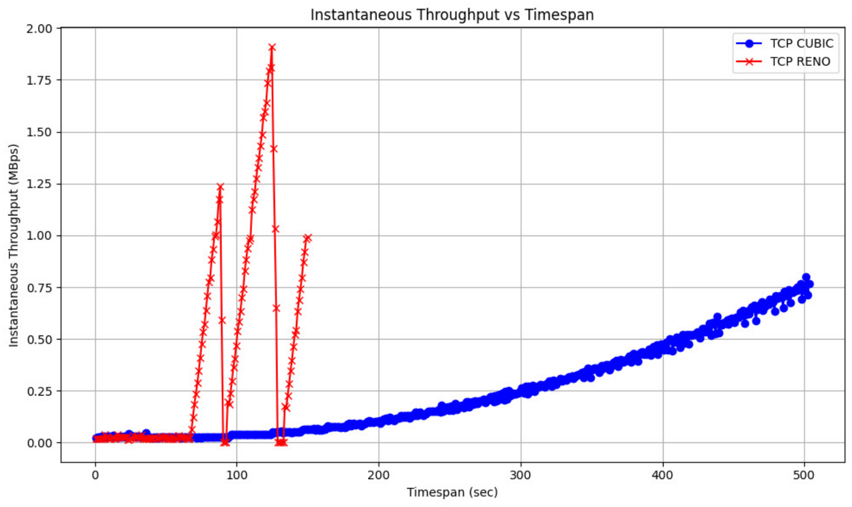
- **Throughput Fluctuations in TCP Reno (Red):** TCP Reno shows a sawtooth pattern in its throughput over time, reflecting its Additive Increase, Multiplicative Decrease (AIMD) congestion control approach. When packet loss occurs, Reno reduces its window size drastically (multiplicative decrease), causing sharp drops in throughput. It then gradually increases the window size additively until it encounters loss again, leading to another drop.

- **Cubic Throughput Behavior (Blue):** TCP CUBIC exhibits a smoother, cubic-shaped progression. Its window growth is faster than Reno's when far from maximum, but it slows as it approaches the congestion window maximum. This behavior results in fewer sudden drops, providing a more stable throughput than Reno. However, CUBIC's throughput does fluctuate, as it periodically adjusts to congestion signals.

- **Delay Impact:** With the low 2 ms delay, TCP CUBIC can better exploit the available bandwidth, recovering from throughput dips quickly. Reno, on the other hand, struggles to maintain high throughput, frequently dropping and increasing its window in response to even minor congestion.

PLOT2

This is the graph for instantaneous throughput vs the timespan for which the file transfer takes place. The red one is for TCP RENO, and the blue one is for TCP CUBIC congestion control. The graph was taken by keeping the link delays 25ms for each link.



Instantaneous throughput for TCP CUBIC and TCP RENO at 25 ms

General observations and explanations:

- **TCP CUBIC(Blue):** TCP CUBIC shows a steady, gradual increase in throughput over time, consistent with its cubic growth function. This smooth growth is characteristic of CUBIC's behavior, as it aims to achieve higher throughput while minimizing abrupt changes. The cubic window adjustment allows it to probe the network more cautiously in high-delay scenarios, which helps maintain consistent growth without drastic fluctuations.

- **TCP RENO(Red):** TCP Reno displays sudden spikes followed by rapid drops, typical of its Additive Increase, Multiplicative Decrease

(AIMD) mechanism. The throughput increases additively until packet loss occurs, after which Reno reduces its congestion window sharply, resulting in a significant throughput drop. This repetitive cycle limits Reno's ability to maintain steady growth

- The increased delay amplifies Reno's throughput oscillations because the protocol's response time to network feedback is slower. In high-delay conditions, Reno struggles to recover quickly after drops, leading to inconsistent performance with lower sustained throughput.
- In contrast, TCP CUBIC's gradual, cubic-based window adjustment is better suited to higher delay scenarios, as it minimizes sudden changes and more effectively exploits the available bandwidth over time.