COL334 Assignment 4

Abhijeet Choudhary(2022CS11104)

Satwik(2022CS51150)

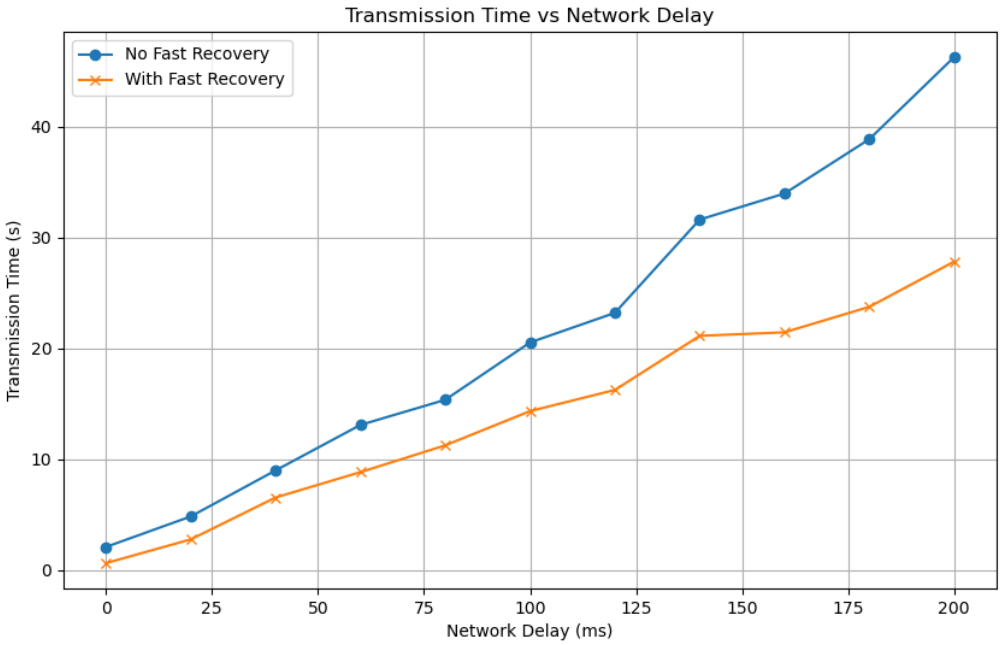
Part 1 Analysis: Reliability

a) Varying Delays from 0 to 200ms, Keeping Loss Fixed at 1%

- Took average of ttc over 5 iterations.
- Checked for both possibilities of enabling and disabling fast recovery.

```
loss,delay,fast_recovery,md5_hash,ttc
1,0,True,2b39c6058615d60985d157d8d95fd67d,0.6462238629659017
1,0,False,2b39c6058615d60985d157d8d95fd67d,2.1038015683492026
1,20,True,2b39c6058615d60985d157d8d95fd67d,2.7895060380299888
1,20,False,2b39c6058615d60985d157d8d95fd67d,4.85591459274292
1,40,True,2b39c6058615d60985d157d8d95fd67d,6.563930511474609
1,40,False,2b39c6058615d60985d157d8d95fd67d,9.003102699915567
1,60,True,2b39c6058615d60985d157d8d95fd67d,8.849786122639975
1,60,False,2b39c6058615d60985d157d8d95fd67d,13.102229515711466
1,80,True,2b39c6058615d60985d157d8d95fd67d,11.263932228088379
1,80,False,2b39c6058615d60985d157d8d95fd67d,15.366202672322592
1,100,True,2b39c6058615d60985d157d8d95fd67d,14.349312623341879
1,100,False,2b39c6058615d60985d157d8d95fd67d,20.53715642293294
1,120,True,2b39c6058615d60985d157d8d95fd67d,16.25616478919983
1,120,False,2b39c6058615d60985d157d8d95fd67d,23.202353636423748
1,140,True,2b39c6058615d60985d157d8d95fd67d,21.133254845937092
1,140,False,2b39c6058615d60985d157d8d95fd67d,31.611337423324585
1,160,True,2b39c6058615d60985d157d8d95fd67d,21.45153872172038
1,160,False,2b39c6058615d60985d157d8d95fd67d,33.96102245648702
1,180,True,2b39c6058615d60985d157d8d95fd67d,23.749366601308186
1,180,False,2b39c6058615d60985d157d8d95fd67d,38.85958456993103
1,200,True,2b39c6058615d60985d157d8d95fd67d,27.824916919072468
1,200,False,2b39c6058615d60985d157d8d95fd67d,46.25843302408854
```

Figure 1: reliability_delay.csv



Clearly, from the plot, multiple trends emerge:

- **Transmission Time Increases with Delay:**
  As the network delay increases, packets take longer to reach the destination, and ACKs take longer to return. Since TCP relies on ACKs to confirm packet receipt, increased delay in ACKs impacts the rate at which new packets can be sent, slowing the transmission and thereby increasing the total transmission time.
- **Linear Relationship Between Transmission Time and Delay:**
  This observed linearity results from the direct impact of RTT on the sending rate in TCP-based protocols. With each increment in delay, RTT increases proportionally, causing each transmission cycle to slow down at a similar rate. Therefore, transmission time grows linearly with network delay as long as there are no extreme congestion or loss events that might otherwise alter this relationship.

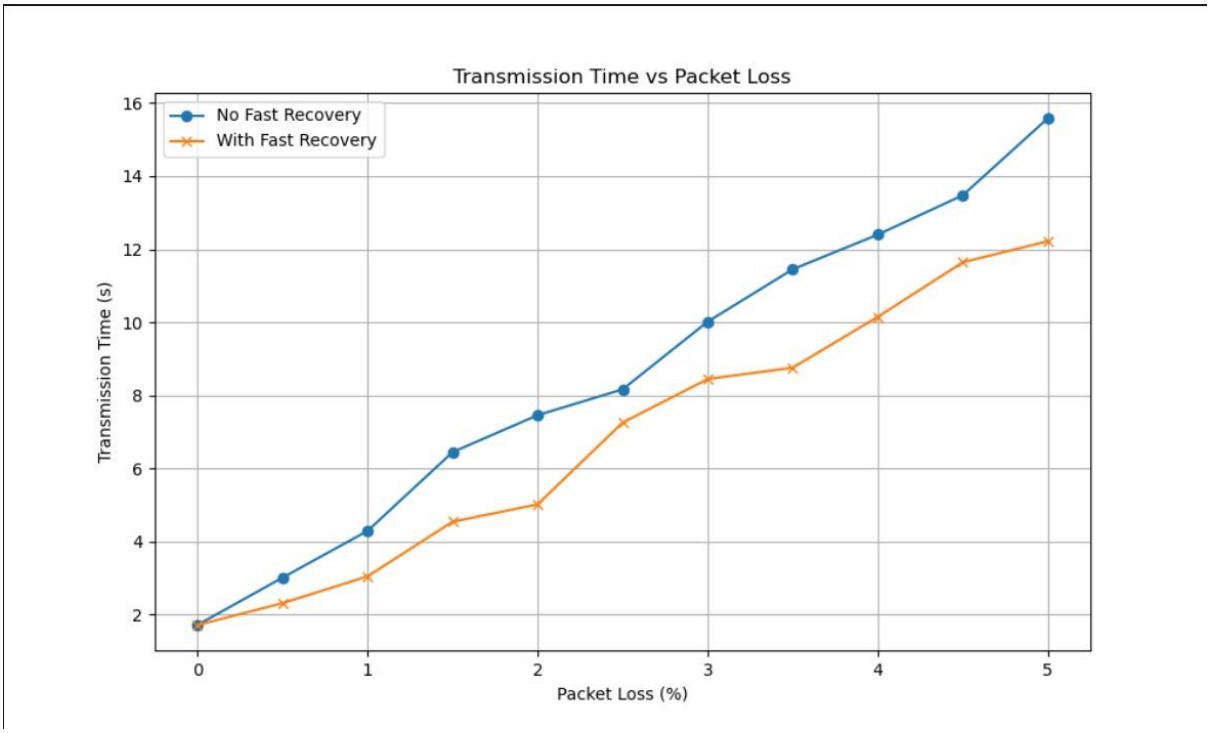
- **Fast Recovery Leads to Faster Transmission:**
  Fast recovery allows the sender to quickly resume data transmission after detecting packet loss, avoiding the need to drop the congestion window drastically. When enabled, it minimizes the impact of minor congestion or packet drops on transmission time. In scenarios with delay, fast recovery helps mitigate the compounded effect of packet loss and delay by ensuring a quicker return to a stable state, reducing total transmission time compared to no fast recovery.

b) Varying Loss Rates from 0% to 5%, Keeping Delay Fixed at 20ms

- **Took average of ttc over 5 iterations.**
- **Checked for both possibilities of enabling and disabling fast recovery.**

```
loss,delay,fast_recovery,md5_hash,ttc
0.0,20,True,3271b7ad8aa55e464692203b442a7313,1.716213544209798
0.0,20,False,3271b7ad8aa55e464692203b442a7313,1.7172893683115642
0.5,20,True,3271b7ad8aa55e464692203b442a7313,2.313824494679769
0.5,20,False,3271b7ad8aa55e464692203b442a7313,3.012310187021891
1.0,20,True,3271b7ad8aa55e464692203b442a7313,3.0483502546946206
1.0,20,False,3271b7ad8aa55e464692203b442a7313,4.291217088699341
1.5,20,True,3271b7ad8aa55e464692203b442a7313,4.542974948883057
1.5,20,False,3271b7ad8aa55e464692203b442a7313,6.447637399037679
2.0,20,True,3271b7ad8aa55e464692203b442a7313,5.018265962600708
2.0,20,False,3271b7ad8aa55e464692203b442a7313,7.458683490753174
2.5,20,True,3271b7ad8aa55e464692203b442a7313,7.254634300867717
2.5,20,False,3271b7ad8aa55e464692203b442a7313,8.165960868199667
3.0,20,True,3271b7ad8aa55e464692203b442a7313,8.448310216267904
3.0,20,False,3271b7ad8aa55e464692203b442a7313,10.018502076466879
3.5,20,True,3271b7ad8aa55e464692203b442a7313,8.758998473485311
3.5,20,False,3271b7ad8aa55e464692203b442a7313,11.450704097747803
4.0,20,True,3271b7ad8aa55e464692203b442a7313,10.14938728014628
4.0,20,False,3271b7ad8aa55e464692203b442a7313,12.399603287378946
4.5,20,True,3271b7ad8aa55e464692203b442a7313,11.642991383870443
4.5,20,False,3271b7ad8aa55e464692203b442a7313,13.477400700251263
5.0,20,True,3271b7ad8aa55e464692203b442a7313,12.219373861948648
5.0,20,False,3271b7ad8aa55e464692203b442a7313,15.584086497624716
```

Figure 2: reliability_loss.csv



From the plot, multiple trends emerge:

- **Transmission Time Increases with Packet Loss:**
  When packet loss occurs, the sender has to retransmit lost packets. Higher loss rates result in more retransmissions, increasing the effective transmission time. In TCP-based protocols, each retransmission cycle involves waiting for a timeout or detecting multiple duplicate ACKs, which slows the overall transmission rate. Both with and without fast recovery, the transmission time increases as the loss rate rises because more packets are lost and need to be retransmitted, delaying successful file delivery.
- **Linear Relationship Between Transmission Time and Loss Rate:**
  A roughly linear trend is observed because each additional percentage of packet loss adds a relatively constant overhead in terms of retransmission time. As packet loss scales, the frequency of retransmission events grows in a roughly linear fashion, thus increasing the overall transmission time proportionally to the loss rate.
- **Fast Recovery Improves Transmission Efficiency:**
  When fast recovery is enabled, upon detecting packet loss through three

duplicate ACKs, the sender immediately retransmits the suspected lost packet (typically the next unacknowledged packet). This proactive retransmission avoids waiting for a timeout, allowing the sender to quickly address the loss.

Part 2 Analysis: Congestion Control

1. **Efficiency:**

a) **Measuring throughput (using ttc) vs. variation of loss rate (0% to 5%)**

- The average transmission time (ttc) was calculated over 5 iterations, analyzing both scenarios with fast recovery enabled and disabled.
- From the plot (Figure 3: Average Throughput $\propto 1/\sqrt{p}$), several trends emerge:
  - Average throughput is inversely proportional to the square root of the packet loss rate.
  - This relationship emphasizes that as packet loss increases, throughput decreases, highlighting the sensitivity of TCP performance to loss rates.
  - Figure 4 demonstrates that the product of throughput and the square root of the packet loss rate remains almost constant, supporting the conclusion that throughput is indeed inversely proportional to the square root of the packet loss rate.

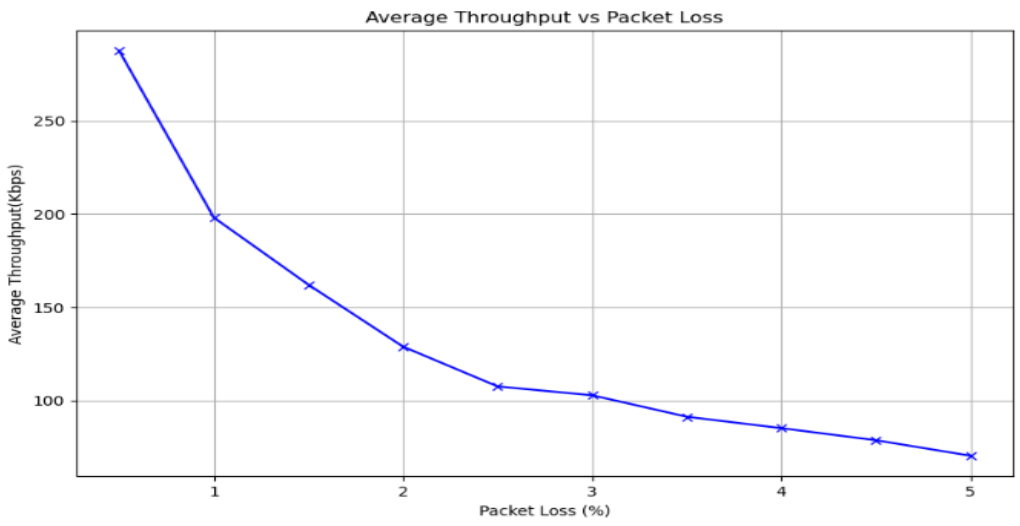


Figure 3:Average Throughput $\propto 1/\sqrt{p}$

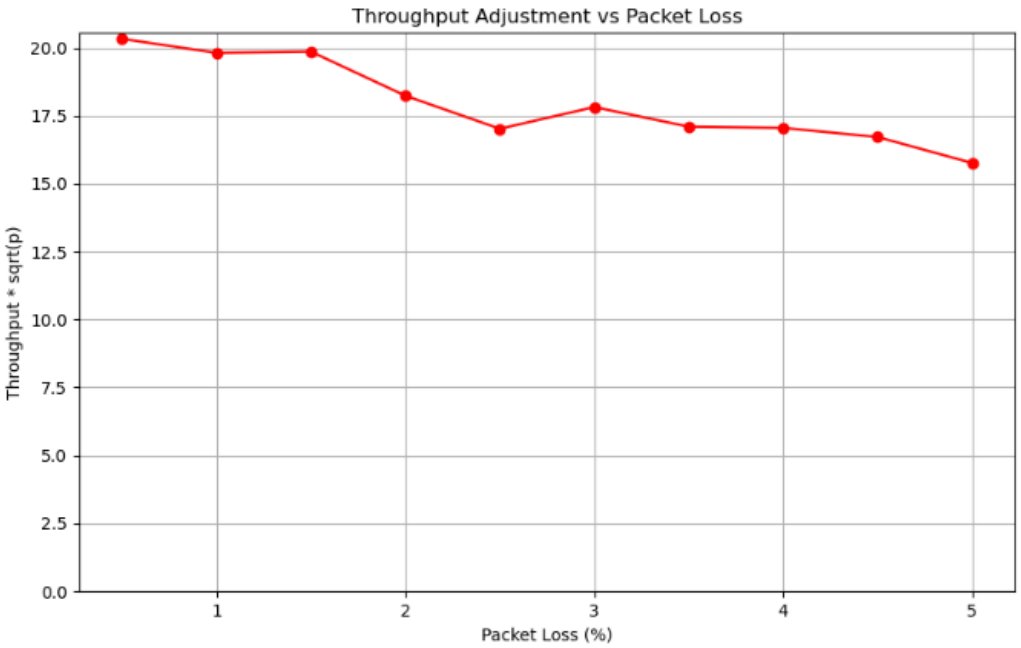


Figure 4: Throughput√p is almost constant*

```
loss,delay,fast_recovery,md5_hash,ttc
0.0,20,True,2b39c6058615d60985d157d8d95fd67d,1.430166244506836
0.5,20,True,2b39c6058615d60985d157d8d95fd67d,7.11229544878006
1.0,20,True,2b39c6058615d60985d157d8d95fd67d,10.320896863937378
1.5,20,True,2b39c6058615d60985d157d8d95fd67d,12.611188888549805
2.0,20,True,2b39c6058615d60985d157d8d95fd67d,15.85766077041626
2.5,20,True,2b39c6058615d60985d157d8d95fd67d,18.997997164726257
3.0,20,True,2b39c6058615d60985d157d8d95fd67d,19.877558290958405
3.5,20,True,2b39c6058615d60985d157d8d95fd67d,22.3764306306839
4.0,20,True,2b39c6058615d60985d157d8d95fd67d,23.985372245311737
4.5,20,True,2b39c6058615d60985d157d8d95fd67d,25.952711760997772
5.0,20,True,2b39c6058615d60985d157d8d95fd67d,29.016217529773712
```

Figure 5: csv file storing ttc for every loss rate from 0-5%

b) **Measuring throughput (using ttc) vs. variation of delay (0ms to 200ms)**

- The average transmission time (ttc) was again calculated over 5 iterations while analyzing both fast recovery enabled and disabled scenarios.
- Observations indicate that increased network delays significantly impact throughput:
    - As network delay increases, overall transmission time also increases, as packets take longer to reach their destination, which in turn affects the ACK return time.
    - The plot illustrates that throughput decreases with increasing delay, demonstrating the negative impact of higher round-trip times on TCP performance.

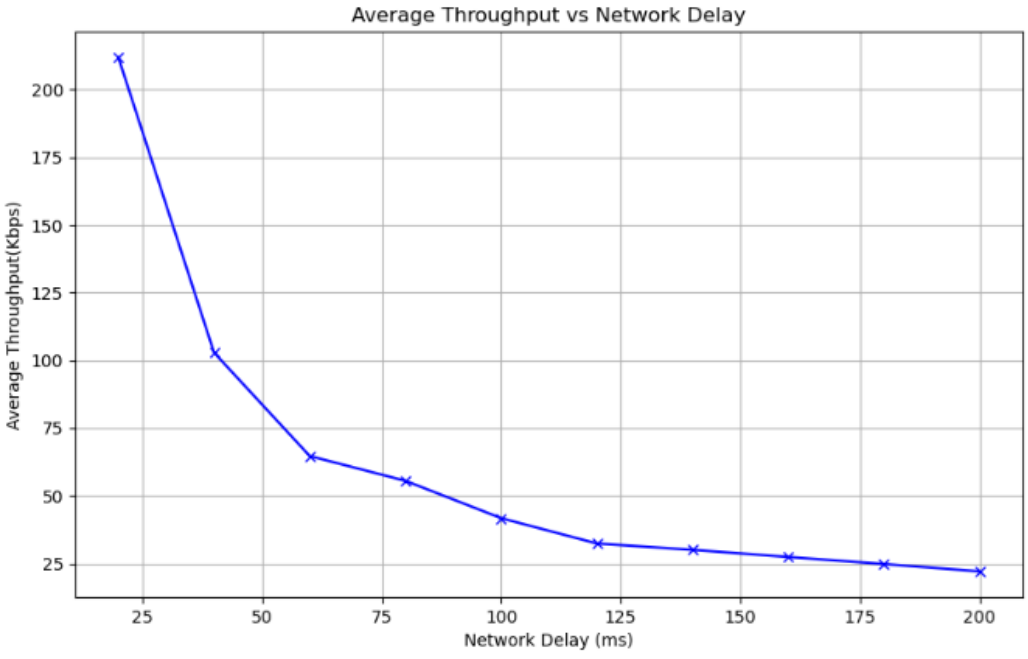


Figure 6: Average Throughput ∝ 1/RTT

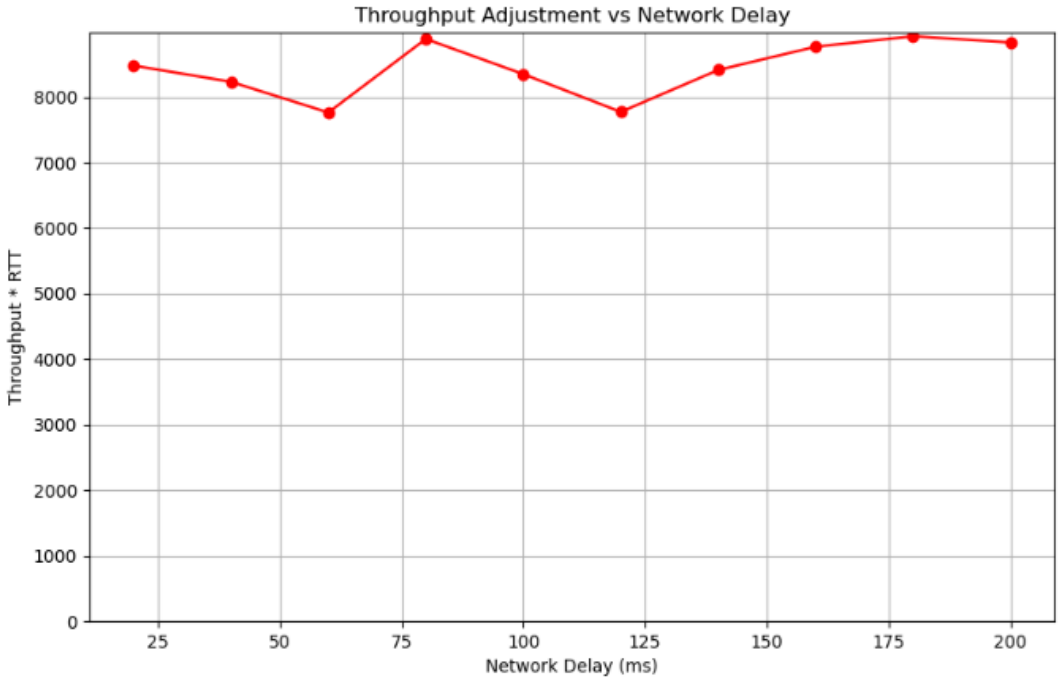


*Figure 7: Throughput*RTT is almost constant*

```
loss,delay,fast_recovery,md5_hash,ttc
1,0,True,2b39c6058615d60985d157d8d95fd67d,0.387368487358093 26
1,20,True,2b39c6058615d60985d157d8d95fd67d,9.64959329366684
1,40,True,2b39c6058615d60985d157d8d95fd67d,19.880209147930145
1,60,True,2b39c6058615d60985d157d8d95fd67d,31.621588587760925
1,80,True,2b39c6058615d60985d157d8d95fd67d,36.83545243740082
1,100,True,2b39c6058615d60985d157d8d95fd67d,49.0054093003273
1,120,True,2b39c6058615d60985d157d8d95fd67d,63.16932129859924
1,140,True,2b39c6058615d60985d157d8d95fd67d,68.09022510051727
1,160,True,2b39c6058615d60985d157d8d95fd67d,74.6649394035339 36
1,180,True,2b39c6058615d60985d157d8d95fd67d,82.49172914028168
1,200,True,2b39c6058615d60985d157d8d95fd67d,92.65224367380142
```

Figure 8:csv file storing ttc for every delay from 0-200ms

Theoretical Explanation of the above results:

The Square Root Law of TCP Throughput

Mathematical Formulation:

The Square Root Law can be expressed as:

$$\text{Throughput} \propto \frac{1}{\sqrt{p}}$$

Where:

- **Throughput** is the effective data transfer rate over a TCP connection.
- p is the packet loss rate (the fraction of packets lost during transmission).

Derivation of the Square Root Law

Modeling Assumptions

1. **Single TCP Connection**: We assume only one TCP connection, with no interference from other connections.
2. **Fixed RTT**: The round-trip time (RTT) is constant, and network delay does not fluctuate.
3. **Ignoring Slow-Start**: We assume the connection is in a steady state and ignore the TCP slow-start phase.

Average Window Size over a Congestion Control Cycle

1. **Behavior of the Congestion Window**: The congestion window W grows from half its maximum value to $W_{\max}$ over each cycle. Given TCP's additive increase and multiplicative decrease behavior, the average window size during this cycle can be approximated as:

$$\text{Average Window Size} = \frac{3}{4} W_{\max}$$

2. **Throughput Expression**: TCP throughput T can be estimated by dividing the average window size by RTT:

$$T = \frac{\text{Average Window Size} \times \text{MSS}}{\text{RTT}} = \frac{\frac{3}{4} W_{\max} \cdot \text{MSS}}{\text{RTT}}$$

.

The total number of packets sent in a cycle (from the beginning of window increase until the next loss) can therefore be approximated as:

$$\text{Packets Sent} = \left(\frac{1}{2} W_{\max}\right) \times \left(\frac{3}{4} W_{\max}\right) = \frac{3}{8} W_{\max}^2$$

Defining Packet Loss Rate p: The packet loss rate p is defined as the ratio of lost packets to packets sent. Since one packet is lost per cycle, we have:

$$p = \frac{1}{\frac{3}{8} W_{\max}^2} = \frac{8}{3 W_{\max}^2}$$

Solving for $W_{\max}$ in terms of p:

$$W_{\max} = \sqrt{\frac{8}{3p}}.$$

Step 3: Substitute $W_{\max}$ Back into the Throughput Expression:
Now, we substitute $W_{\max}$ in terms of p back into the initial throughput expression:

$$T = \frac{3}{4} W_{\max} \cdot \frac{MSS}{RTT}.$$

Substituting for $W_{\max}$:

$$T = \frac{3}{4} \left(\sqrt{\frac{8}{3p}} \right) \cdot \frac{MSS}{RTT}.$$

Thus, the final expression for throughput T is given by:

$$T = \sqrt{\frac{3}{2}} \times \frac{MSS}{RTT\sqrt{p}}.$$

From this, we conclude that throughput T is inversely proportional to both the square root of the packet loss rate p and the round-trip time RTT:

$$T \propto \frac{1}{\sqrt{p}} \quad \text{and} \quad T \propto \frac{1}{RTT}.$$

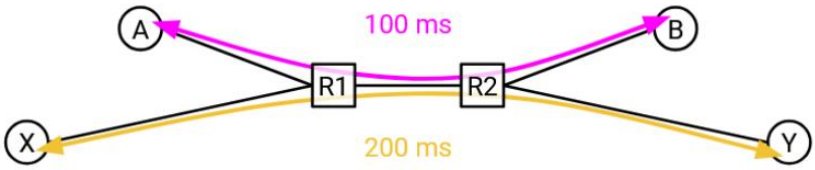
This relationship emphasizes that higher packet loss leads to lower throughput, and lower RTT enhances throughput, indicating that TCP's performance is sensitive to both loss rates and network delays.

Implications of Equation

We now have an equation for throughput, expressed in terms of RTT and loss rate. What does it tell us?

1. **Throughput is inversely proportional to the square root of the loss rate**: Intuitively, if the loss rate is higher, then the throughput is lower. This makes sense because losing more packets means that the window size gets halved more often.
2. **Throughput is inversely proportional to RTT**: Intuitively, if the RTT is lower, then the throughput is higher. This makes sense because the window size increases every time we receive an acknowledgment (ack), and a lower RTT means we get more acks more frequently.

This relationship between RTT and throughput can be problematic if we have multiple connections with different RTTs.



- The connection with the lower RTT will receive acks more quickly, allowing it to increase its window size and send packets faster. In this scenario, it turns out that the lower-RTT connection can utilize twice as much bandwidth as the higher-RTT connection.

Fundamentally, TCP is unfair when RTTs are heterogeneous (not the same). A shorter RTT improves propagation time, but it also enables TCP to ramp up its rate faster. We accept this as a feature of TCP, and there's little we can do about this in practice.

2)Fairness:

Fairness in networking, especially with TCP, refers to how network resources, such as bandwidth, are shared among multiple competing connections. Ideally, a fair TCP system should ensure that all active connections receive equal access to bandwidth, promoting efficient resource utilization while preventing any single connection from monopolizing the network.

Jain's Fairness Index: $$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2} = \frac{\overline{\mathbf{x}}^2}{\overline{\mathbf{x}^2}}$$

1. Fairness in TCP Connections:

- TCP's congestion control mechanisms, such as AIMD (Additive Increase, Multiplicative Decrease), are designed to improve the fairness of resource distribution.
- In heterogeneous networks where connections have varying RTTs or are subject to different packet loss rates, fairness can be affected. For instance:
  - Connections with shorter RTTs receive acknowledgments faster, allowing them to increase their congestion window size more quickly.
  - This may lead to faster data transmission for short RTT connections at the expense of longer RTT connections.
- Such inequalities are exacerbated when multiple TCP flows are competing, leading to imbalances where some flows gain bandwidth advantage over others, impacting fairness across the network.

Analysis of Congestion Control Effectiveness Based on Fairness Values

To evaluate the effectiveness of the congestion control mechanism, we analyzed fairness values using Jain's Fairness Index (JFI) for different bottleneck delays, ranging from 0ms to 100ms. The fairness values averaged over five iterations are: `[0.988, 0.974, 0.967, 0.962, 0.912, 0.857]`. Here's a detailed analysis of what these values indicate about the performance of the congestion control mechanism.

1. Fairness Values Close to 1 Indicate Effective Congestion Control:

- The values are close to 1, with a high fairness index ranging from 0.988 to 0.857. Values near 1 signify that resources are being distributed nearly equally among competing connections, indicating an effective congestion control mechanism.
- The high fairness suggests that TCP's congestion control algorithm, which uses strategies like additive increase and multiplicative decrease, effectively prevents any one connection from monopolizing the available bandwidth. This results in a balanced sharing of resources, even under increasing traffic loads.
- High JFI values also suggest that the congestion control mechanism quickly adapts to packet loss or delays by adjusting the congestion window size, helping to maintain consistent throughput across connections.

2. Decreasing Fairness with Increasing Delays:

- The fairness index gradually decreases from 0.988 at 0ms delay to 0.857 at 100ms delay. This trend suggests that as network delay increases, the effectiveness of congestion control in maintaining fairness reduces.
    - Higher delays lead to increased round-trip times (RTTs), slowing down the feedback loop that TCP relies on for congestion management. With longer RTTs, connections with shorter delays are able to receive acknowledgments and increase their congestion window size faster than those with longer delays.
    - This discrepancy creates an imbalance in bandwidth allocation, as connections with lower delays continue to grow their window size more frequently than those facing higher delays. This behavior demonstrates TCP's sensitivity to RTT differences, which can diminish fairness when there is a significant variation in delays.

3. Implications of Reduced Fairness with Delay on Network Performance:

- As fairness values drop with higher delays, congestion control mechanisms may be less efficient at balancing resources. Lower fairness could mean certain connections experience significantly lower throughput, leading to underutilization of network resources and potential congestion if other connections overuse available bandwidth.
- This reduction in fairness as delay increases highlights a known limitation of TCP, where connections with shorter RTTs receive a disproportionate share of bandwidth, an effect known as "RTT unfairness."

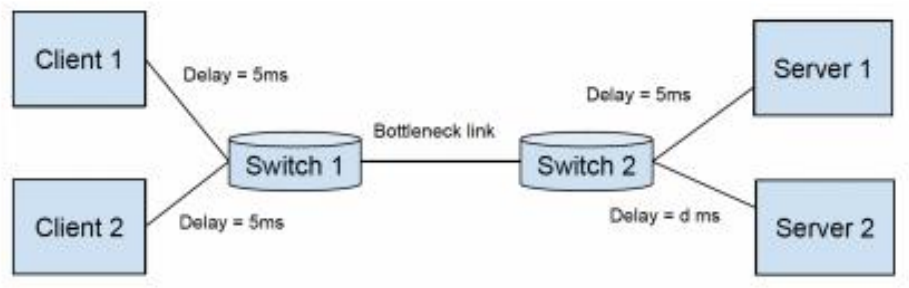


Figure 9:The network topology on which congestion control is run

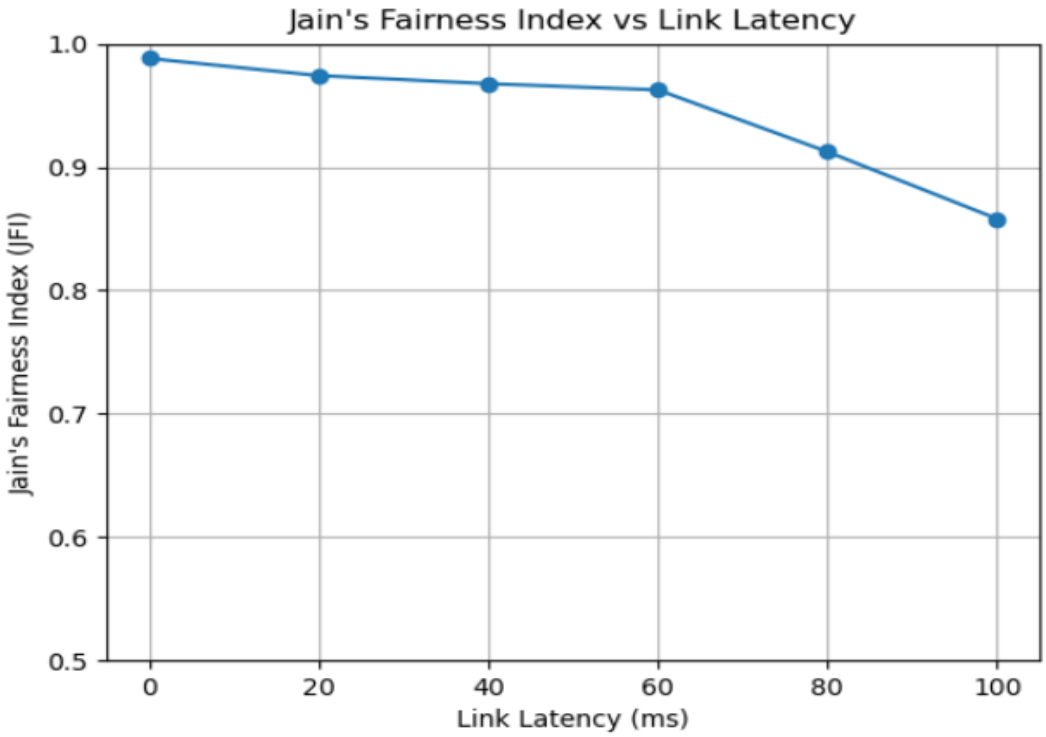
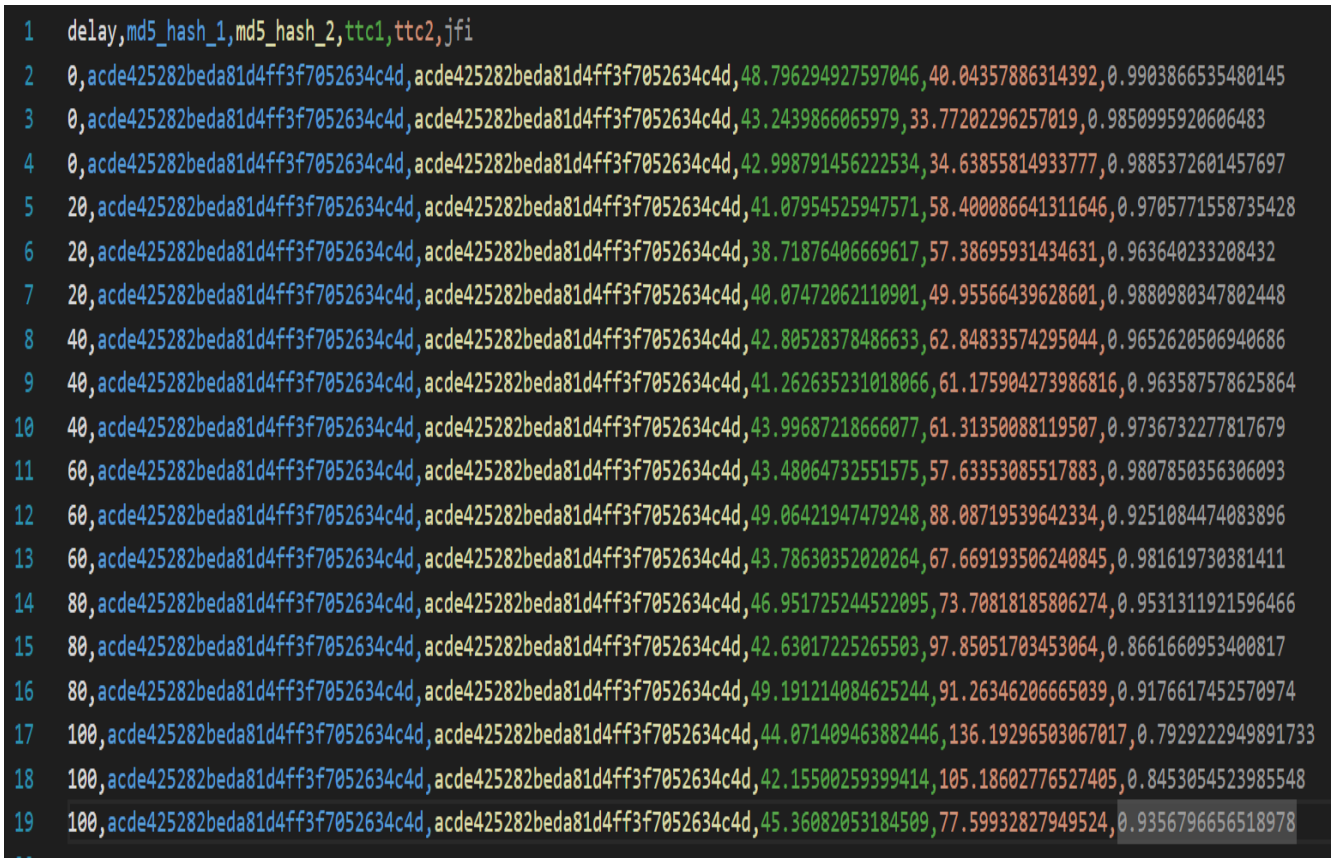


Figure 10: Plot of JFI vs Latency of bottleneck link

JFI Values:

Delay(ms)	Jain's Fairness Index
0	**0.988**
20	**0.974**
40	**0.967**
60	**0.962**
80	**0.912**
100	**0.857**



```
delay,md5_hash_1,md5_hash_2,ttc1,ttc2,jfi
0,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,48.796294927597046,40.04357886314392,0.9903866535480145
0,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,43.2439866065979,33.77202296257019,0.9850995920606483
0,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,42.998791456222534,34.63855814933777,0.9885372601457697
20,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,41.07954525947571,58.400086641311646,0.9705771558735428
20,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,38.71876406669617,57.38695931434631,0.963640233208432
20,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,40.07472062110901,49.95566439628601,0.9880980347802448
40,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,42.80528378486633,62.84833574295044,0.9652620506940686
40,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,41.262635231018066,61.175904273986816,0.963587578625864
40,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,43.99687218666077,61.31350088119507,0.9736732277817679
60,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,43.48064732551575,57.63353085517883,0.9807850356306093
60,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,49.06421947479248,88.08719539642334,0.9251084474083896
60,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,43.78630352020264,67.669193506240845,0.981619730381411
80,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,46.951725244522095,73.70818185806274,0.9531311921596466
80,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,42.63017225265503,97.85051703453064,0.8661660953400817
80,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,49.191214084625244,91.26346206665039,0.9176617452570974
100,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,44.071409463882446,136.19296503067017,0.7929222949891733
100,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,42.15500259399414,105.18602776527405,0.8453054523985548
100,acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,45.36082053184509,77.59932827949524,0.9356796656518978
```

Figure 11: tcc1 and tcc2 values ,and the calculated JFI(3 iterations)

Throughput = File Size/Transmission Time

Some example of throughput log in terminal:

```
Delay:  20
Throughput Server1(No congestion control):  1738.9056923475246
Kbps
Throughput Server2(TCP Reno):  939.1318093736429  Kbps
```

Part 3(Bonus) : TCP CUBIC

TCP CUBIC is a congestion control algorithm optimized for high-speed networks. Instead of linear growth, it uses a **cubic function** to adjust the congestion window:

$$W(t) = C(t - K)^3 + W_{\max}$$

where:

- $W_{\max}$ is the previous maximum window size,
- C (0.4) controls the growth rate, and
- K is the time to reach $W_{\max}$ again, scaled by a factor β (usually 0.5).

Main Features

1. **Faster Growth**: The cubic function enables quick growth when far from $W_{\max}$, improving bandwidth usage.
2. **RTT Fairness**: CUBIC's growth isn't dependent on RTT, allowing fairer bandwidth distribution across flows with different RTTs.
3. **Quick Recovery**: After loss, the window increases quickly back to near $W_{\max}$, allowing efficient recovery without long delays.

TCP CUBIC is widely adopted for its ability to handle high-bandwidth and high-latency environments efficiently.

3.1) EFFICIENCY

In TCP CUBIC, the average throughput is impacted by both packet loss and network delay due to the congestion window's behavior:

1. **Increasing Packet Loss Rate**: When packet loss occurs, CUBIC reduces the congestion window, slowing data transmission. Higher packet loss rates mean more frequent window reductions, which restricts throughput since CUBIC has to wait before it can grow the window back to optimal levels.
2. **Increasing Network Delay**: CUBIC's growth rate is independent of RTT, but higher delay still reduces the effective throughput. With longer delays, ACKs take longer to return, slowing the feedback loop and ultimately limiting how fast CUBIC can increase the window, resulting in reduced throughput.

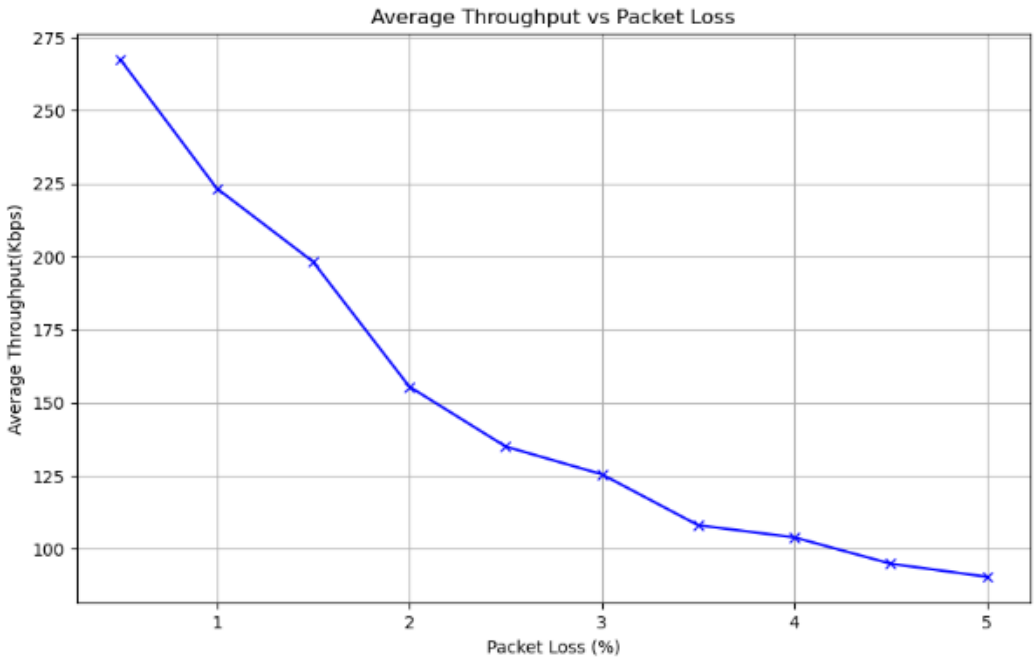


Figure 12: Average Throughput decreases with increase in Loss Rate

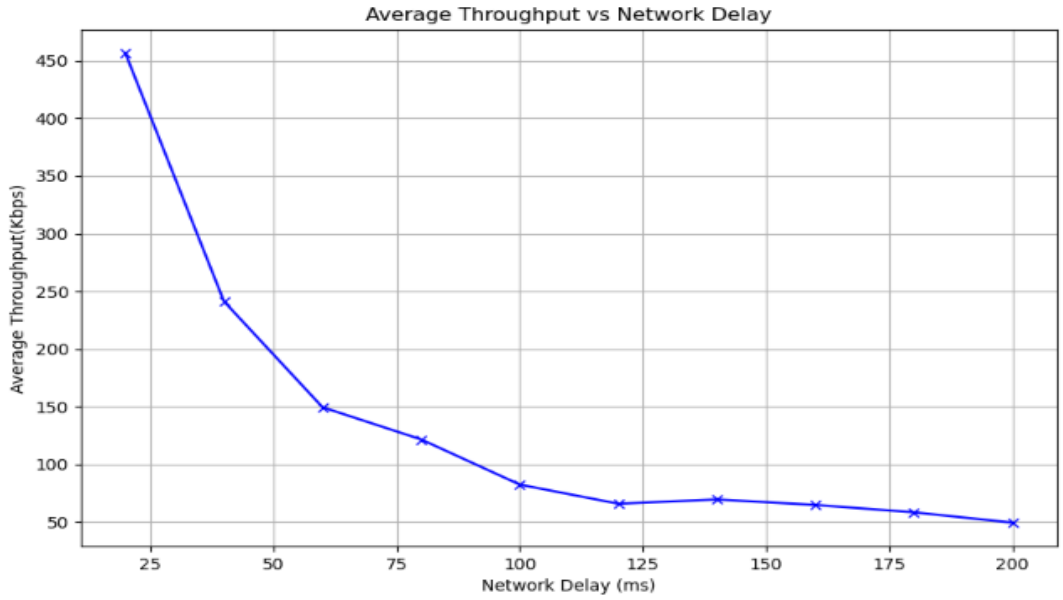


Figure 13: Figure 12: Average Throughput decreases with increase in Network Delay

```
loss,delay,fast_recovery,md5_hash,ttc
1,0,True,2b39c6058615d60985d157d8d95fd67d,4.35786509513855
1,20,True,2b39c6058615d60985d157d8d95fd67d,4.479536294937134
1,40,True,2b39c6058615d60985d157d8d95fd67d,8.474744319915771
1,60,True,2b39c6058615d60985d157d8d95fd67d,13.678957462310791
1,80,True,2b39c6058615d60985d157d8d95fd67d,16.818626642227173
1,100,True,2b39c6058615d60985d157d8d95fd67d,24.802945375442505
1,120,True,2b39c6058615d60985d157d8d95fd67d,31.05662202835083
1,140,True,2b39c6058615d60985d157d8d95fd67d,29.40265464782715
1,160,True,2b39c6058615d60985d157d8d95fd67d,31.558335304260254
1,180,True,2b39c6058615d60985d157d8d95fd67d,35.02583694458008
1,200,True,2b39c6058615d60985d157d8d95fd67d,41.368866205215454
```

Figure 14: TTC for different delays from 0 to 200ms(TCP CUBIC)

```
loss,delay,fast_recovery,md5_hash,ttc
0.0,20,True,2b39c6058615d60985d157d8d95fd67d,6.196461200714111
0.5,20,True,2b39c6058615d60985d157d8d95fd67d,7.645967245101929
1.0,20,True,2b39c6058615d60985d157d8d95fd67d,9.153276443481445
1.5,20,True,2b39c6058615d60985d157d8d95fd67d,10.312638521194458
2.0,20,True,2b39c6058615d60985d157d8d95fd67d,13.145873069763184
2.5,20,True,2b39c6058615d60985d157d8d95fd67d,15.142560482025146
3.0,20,True,2b39c6058615d60985d157d8d95fd67d,16.289978504180908
3.5,20,True,2b39c6058615d60985d157d8d95fd67d,18.915643453598022
4.0,20,True,2b39c6058615d60985d157d8d95fd67d,19.672447443008423
4.5,20,True,2b39c6058615d60985d157d8d95fd67d,21.538715362548828
5.0,20,True,2b39c6058615d60985d157d8d95fd67d,22.60566210746765
```

Figure 15:TTC values for loss rates from 0% to 5%(TCP CUBIC)

The average throughput for TCP CUBIC is generally higher than TCP Reno across most values of loss rate and network delay due to the fundamental differences in how each protocol handles window growth:

1. **Loss Rate**: TCP CUBIC adjusts the congestion window using a cubic function, allowing for more aggressive growth even after packet loss events. This approach enables it to recover faster and maintain higher throughput compared to TCP Reno, which uses a linear increase and aggressively halves the congestion window upon loss detection. Consequently, CUBIC's more gradual reduction and rapid recovery contribute to better performance in lossy networks.
2. **Network Delay**: TCP CUBIC is less sensitive to RTT variations because its window growth rate is independent of RTT. This feature allows it to maintain higher throughput in networks with variable or high delays. In contrast, TCP Reno's window growth is tightly bound to RTT, causing its throughput to diminish significantly with increasing delay.

3.2)

1. TCP Reno Growth Pattern

- TCP Reno increases the congestion window linearly in the congestion avoidance phase, adding one segment per RTT.
- Upon packet loss, Reno halves the congestion window, creating a distinct "sawtooth" pattern as throughput ramps up slowly and drops sharply on loss detection.

2. TCP CUBIC Growth Pattern

- TCP CUBIC follows a cubic growth function, which initially increases the window slowly (concave growth) and accelerates as it nears the previous maximum (convex growth).
- This cubic behavior allows faster recovery after loss, especially beneficial in high-BDP (Bandwidth-Delay Product) networks.

3. Loss Response

- TCP Reno reacts strongly to loss by halving the window, leading to slower recovery and lower throughput in high-loss scenarios.
- TCP CUBIC reduces the window less drastically after loss, allowing for a smoother and more efficient ramp-up, improving throughput in lossy environments.

4. RTT Sensitivity

- TCP Reno's linear growth is highly sensitive to RTT, as a high RTT delays the rate increase, impacting overall performance.
- TCP CUBIC's RTT independence lets it achieve higher throughput on high-latency networks since the cubic growth shape compensates for RTT delays.

5. Throughput Comparison

- TCP CUBIC generally sustains higher throughput than Reno, thanks to its adaptive cubic function and faster recovery, making it favorable in networks with higher loss rates or delays.

Graphs typically show Reno's throughput with a sharp, jagged sawtooth, while CUBIC exhibits smoother, cubic curves that reach higher throughput more consistently.

Plots:

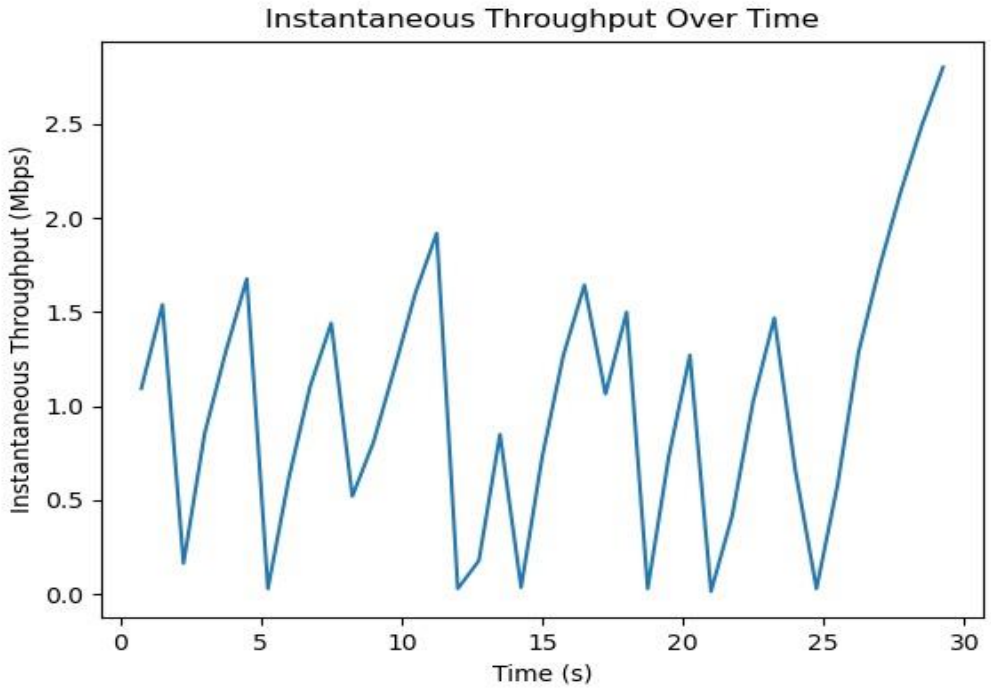


Figure 16:TCP Reno, link delay= 2ms

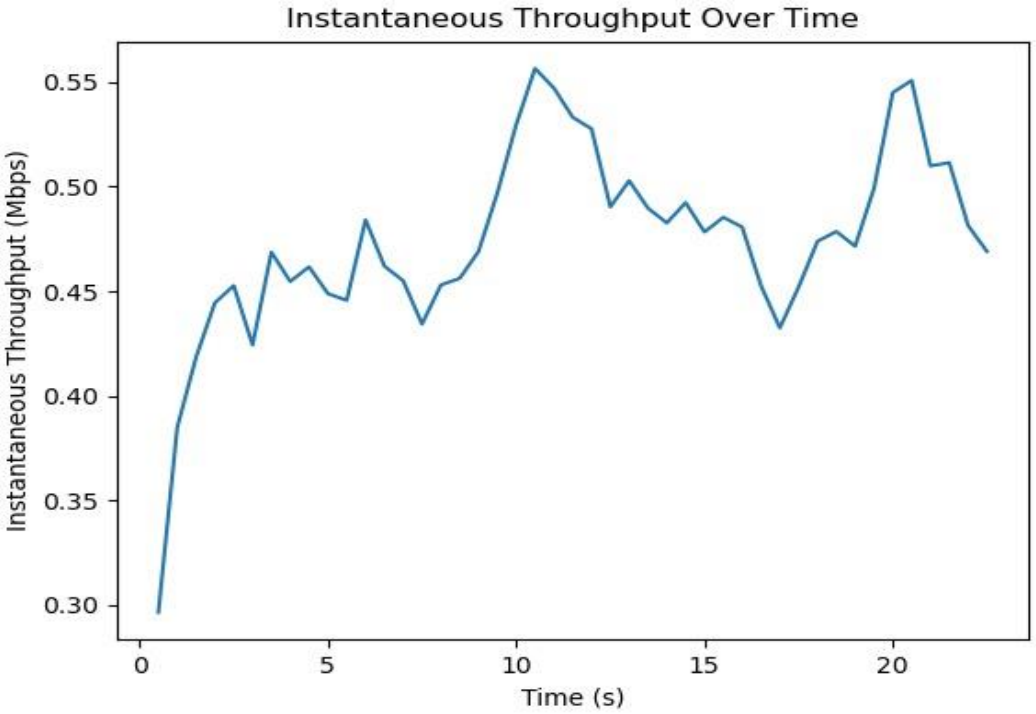


Figure 17: TCP CUBIC, link delay=2ms

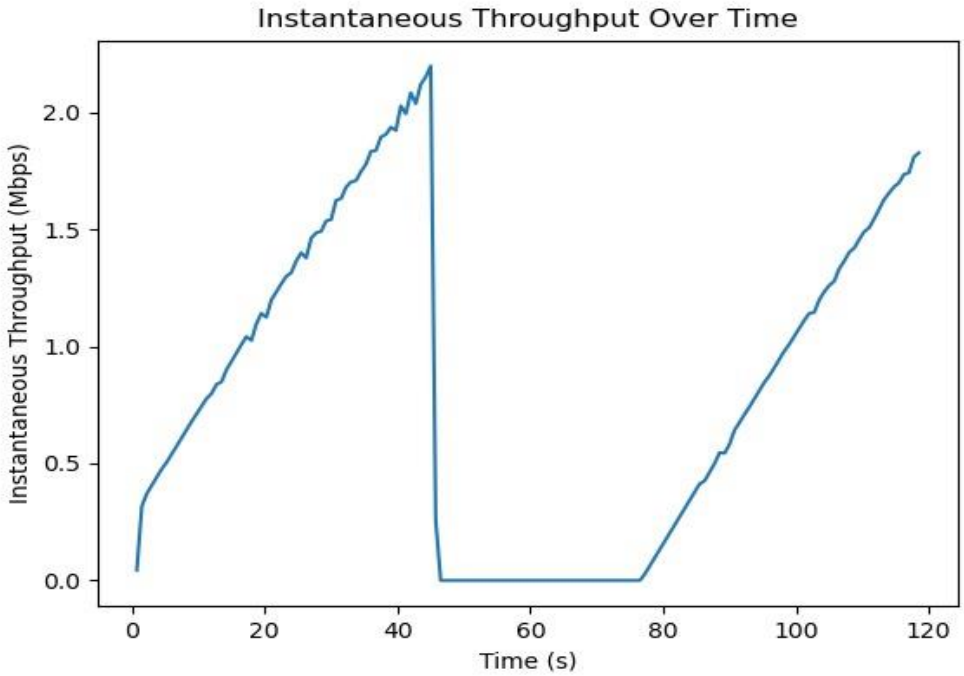


Figure 18: TCP Reno, link capacity=25ms

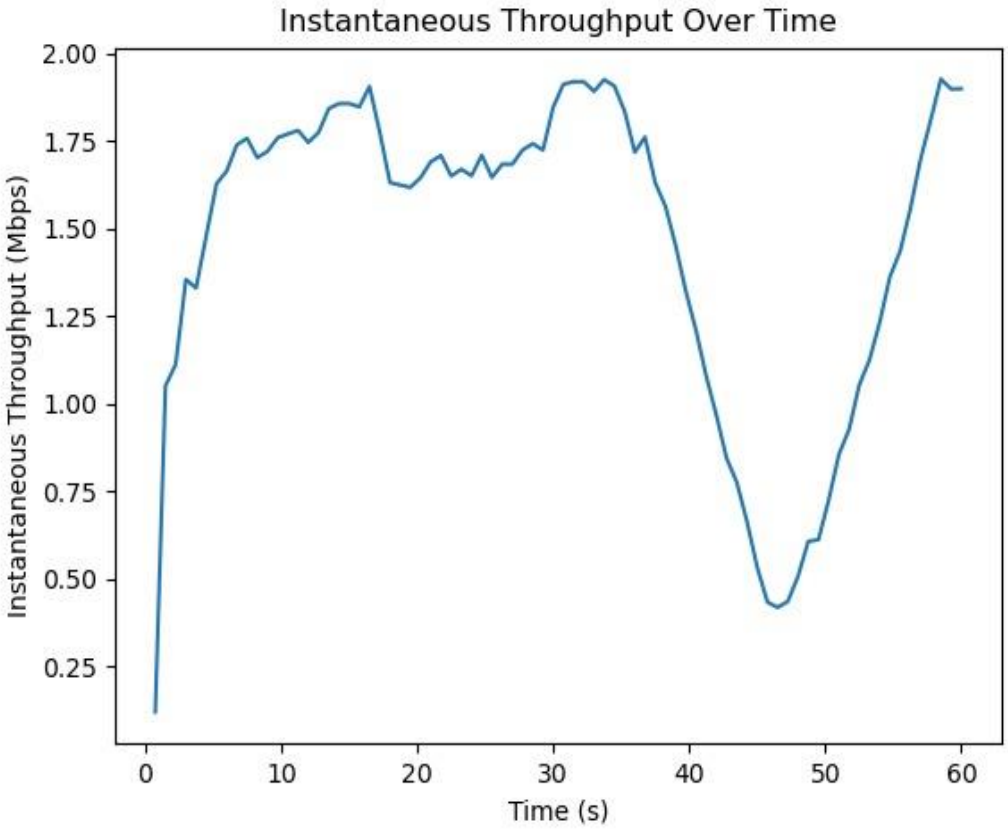


Figure 19:TCP CUBIC, link capacity=25ms

Fairness of Reno vs CUBIC:

1. Fairness Values: Reno vs. CUBIC

- In scenarios where TCP Reno's transmission time (ttc) is observed to be higher than that of TCP CUBIC, it indicates that Reno is less efficient in managing network congestion compared to CUBIC. This inefficiency is primarily due to Reno's linear congestion window growth and its drastic reduction of the window size upon packet loss, leading to longer recovery times.
- TCP CUBIC, with its cubic growth function, adapts more effectively to network conditions, allowing for quicker ramp-up in the congestion window and better utilization of available bandwidth, resulting in lower transmission times.

2. High Fairness Values

- The fairness index, often exceeding 0.9, suggests that both TCP Reno and CUBIC exhibit a high level of fairness in bandwidth allocation among competing flows. High fairness indicates that these protocols can manage congestion well, allowing multiple connections to coexist with minimal bias towards any single flow.
- Fairness can be attributed to the algorithms' mechanisms for detecting and responding to congestion. Although Reno experiences higher ttc, both protocols manage to maintain equitable bandwidth distribution, particularly under controlled network conditions. The high fairness values reinforce the effectiveness of congestion control mechanisms in ensuring stable performance for multiple TCP flows, regardless of individual throughput differences.

```
delay,md5_hash_1,md5_hash_2,ttc1,ttc2,jfi
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,95.80886912345886,58.90941119194031,0.9461815496640419
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,62.0096697807312,48.893715381622314,0.9862063881020746
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,88.29919624328613,77.2899706363678,0.9955991736885189
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,61.94651961326599,48.58869194984436,0.9856062541790587
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,61.89573287963867,48.73389196395874,0.986043184709292
```

Figure 20:Fairness of Reno vs CUBIC, at 2ms

```
delay,md5_hash_1,md5_hash_2,ttc1,ttc2,jfi
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,32.39123225212097,26.07106065750122,0.9884479290876402
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,34.50624442100525,23.216712713241577,0.9631572605690644
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,30.159701824188232,25.450589895248413,0.9928802531355518
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,33.28543400764465,29.511890172958374,0.9964020829378963
acde425282beda81d4ff3f7052634c4d,acde425282beda81d4ff3f7052634c4d,33.303017139434814,22.972774744033813,0.9674024487345955
```

Figure 21: Fairness of Reno vs CUBIC, at 25 ms