SIMULATION AND ANALYSIS OF TCP CONGESTION CONTROL VARIANTS

INDEX

INTRODUCTION

CONGESTION CONTROL

Congestion is a situation in Communication Networks in which too many packets are present in a part of the subnet and it subsequently leads to performance degradation. Congestion in a network may occur when the load on the network i.e. the number of packets sent to the network is greater than the capacity of the network.

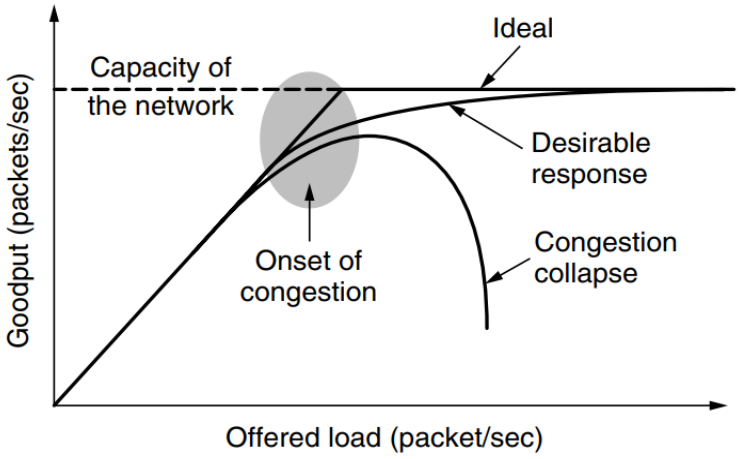

Figure 1: Performance Drop due to Congestion

Congestive collapse or **congestion collapse** is the condition in which congestion prevents or limits useful communication. Congestion collapse generally occurs at choke points in the network, where incoming traffic exceeds outgoing bandwidth.

Congestion Control refers to techniques that can either prevent congestion, before it happens, or remove congestion after it has happened. Congestion control modulates traffic entry into a network in order to avoid congestive collapse resulting from oversubscription. This is typically accomplished by reducing the rate of packets. Congestion control prevents senders from overwhelming the network.

TRANSMISSION CONTROL PROTOCOL

Transmission Control Protocol (TCP) is a standard that defines how to establish and maintain a network communication through which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other. It is a connection-oriented communications protocol that is used for exchanging data between two end devices in a network. It is very powerful in dealing with congestion control and retransmission.

TCP takes messages from a server and divides them into packets, which can then be forwarded by the devices in the network – switches, routers, security gateways – to the destination. TCP numbers each packet and reassembles them prior to handing them off to the server recipient.

TCP is used for organizing data in a way that ensures the secure transmission between the server and client. Therefore, it is used to transmit data from other higher-level protocols such as: Secure Shell (SSH), File Transfer Protocol (FTP), Internet Message Access Protocol (IMAP) and Hypertext Transfer Protocol (HTTP).

TCP CONGESTION CONTROL

TCP Congestion Control techniques prevent congestion or help mitigate the congestion after it occurs. The basis of TCP congestion includes:

- Additive Increase Multiplicative Decrease (AIMD), which involves halving the congestion window for every window containing a packet loss, and increasing the congestion window by roughly one segment per RTT otherwise.
- The second component is the Retransmit Timer, including the exponential back off of the retransmit timer when a retransmitted packet is dropped.
- The third fundamental component is the Slow-Start mechanism for the initial probing for available bandwidth.
- The fourth TCP congestion control mechanism is ACK-Clock, where the arrival of acknowledgements at the sender is used to clock out the transmission of new data.

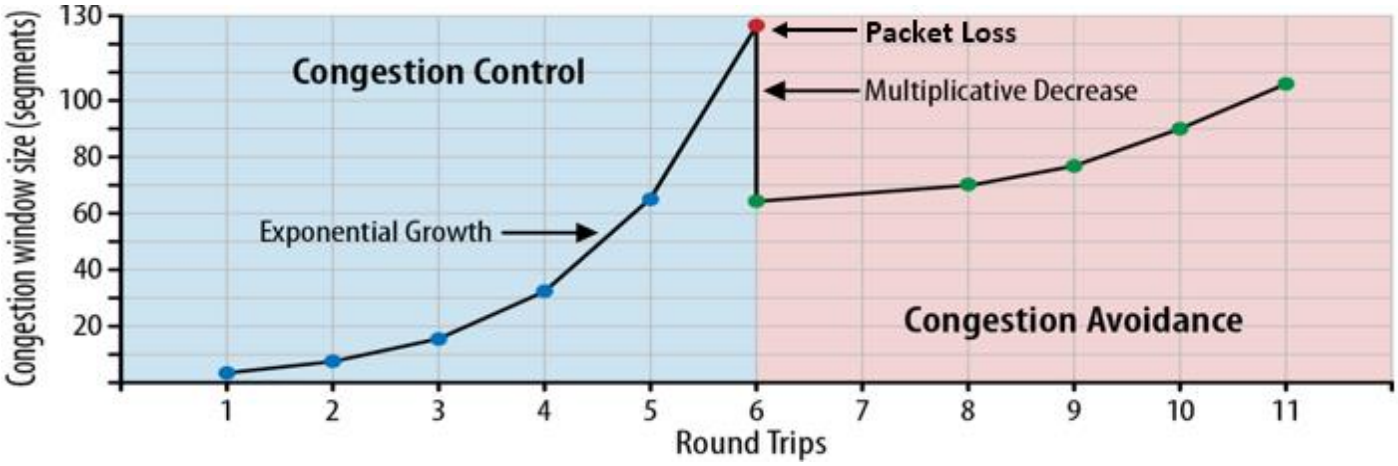

Figure 2: TCP Congestion Control

Some well-known variants of TCP Congestion Control are:

- **TCP Tahoe:** It adds a number of new algorithms and refinements to earlier TCP implementations. The new algorithms include Slow-Start, Congestion Avoidance, and Fast Retransmit.
- **TCP Reno:** It retains the enhancements incorporated into Tahoe TCP but modified the Fast Retransmit operation to include Fast Recovery. It prevents the communication channel from going empty after Fast Retransmit.
- **TCP New Reno:** It is a slight modification over TCP-Reno. It is able to detect multiple packet losses and thus is much more efficient that Reno in the event of multiple packet losses.
- **TCP Vegas:** It adopts a more sophisticated bandwidth estimation scheme. It uses the difference between expected and actual flow rates to estimate the available bandwidth in the network.
- **TCP SACK:** TCP with Selective Acknowledgment (SACK) preserves the properties of TCP Tahoe and Reno and uses retransmit timeouts as a recovery method.

BACKGROUND KNOWLEDGE

ADDITIVE-INCREASE MULTIPLICATIVE-DECREASE

The **Additive-Increase Multiplicative-Decrease (AIMD)** algorithm is a feedback control algorithm best known for its use in TCP congestion control. AIMD combines linear growth of the congestion window when there is no congestion with an exponential reduction when congestion is detected. Multiple flows using AIMD congestion control will eventually converge to an equal usage of a shared link.

The approach taken is to increase the transmission rate (window size), probing for usable bandwidth, until loss occurs. The policy of additive increase may, for instance, increase the congestion window by a fixed amount every round trip time. When congestion is detected, the transmitter decreases the transmission rate by a multiplicative factor. For example, cut the congestion window in half after loss. The result is a saw-tooth behavior that represents the process of bandwidth probing.

Let $\boldsymbol{\omega(t)}$ be the congestion window size indicating the amount of data in flight during time slot **t**, **a** (a>0) be the additive increase parameter, and **b** (0<b<1) be the multiplicative decrease factor.

$$\omega(t+1)= \begin{cases} \omega(t) + a & \text{if congestion is not detected} \\ \omega(t) \text{ x } b & \text{if congestion is detected} \end{cases}$$

In TCP, after slow start, the additive increase parameter **a** is typically one MSS (maximum segment size) per round-trip time, and the multiplicative decrease factor **b** is typically 1/2.

TCP TAHOE

TCP Tahoe involved a few new algorithms in early TCP implementations like **Slow-Start**, **Congestion Avoidance**, and **Fast Retransmit**. Among these the fast retransmit algorithm is of special interest as it has been retained in its basic form in subsequent versions of TCP. In Fast Retransmit, after receiving a small number of duplicate acknowledgments for the same TCP segment (dup ACKs), the data sender infers that the packet has been lost and retransmits the packet without waiting for a retransmission timer to expire.

Generally, it has been seen that the duplicate acknowledgment threshold is fixed at three. Therefore, on receiving three successive duplicate acknowledgments the sender can infer that receiver has not received the packet, and a retransmission is triggered without waiting for a timeout. Independence on the retransmission timeout for taking retransmission decisions of the lost packet and subsequent earlier loss recovery leads to higher channel utilization and connection throughput. The protocol returns to a slow start and sets the slow start threshold to one-half of the congestion window.

TCP RENO

This variant of TCP is similar to the Tahoe TCP, except it also includes **Fast Recovery**. TCP Reno differs from TCP Tahoe at the stage of congestion avoidance. Whenever **3 duplicate ACKs** are received, it will halve the congestion window, perform a fast retransmit, and enter fast recovery phase. If a timeout event occurs, it will enter slow-start, as in same as TCP Tahoe.

Fast Recovery is a temporary mode that aims to maintain the **ACK-clock** running with a congestion window that is the new threshold, or half the value of the congestion window at the time of the fast retransmission. To do this, duplicate acknowledgements are counted (including the three that triggered fast retransmission) until the number of packets in the network has fallen to the new threshold. The upshot of this heuristic is that TCP avoids slow start, except when the connection is first started and when a timeout occurs.

When packet loss is small, Reno performs well. But if there are multiple packet losses then Reno does not give good performance and behaves same as Tahoe. TCP Reno does not perform well when the congestion window is very small. Because when a packet loss occurs, there are not sufficient packets to generate duplicate acknowledgement thrice, needed to enter in the fast retransmit phase. The only solution left is the timeout.

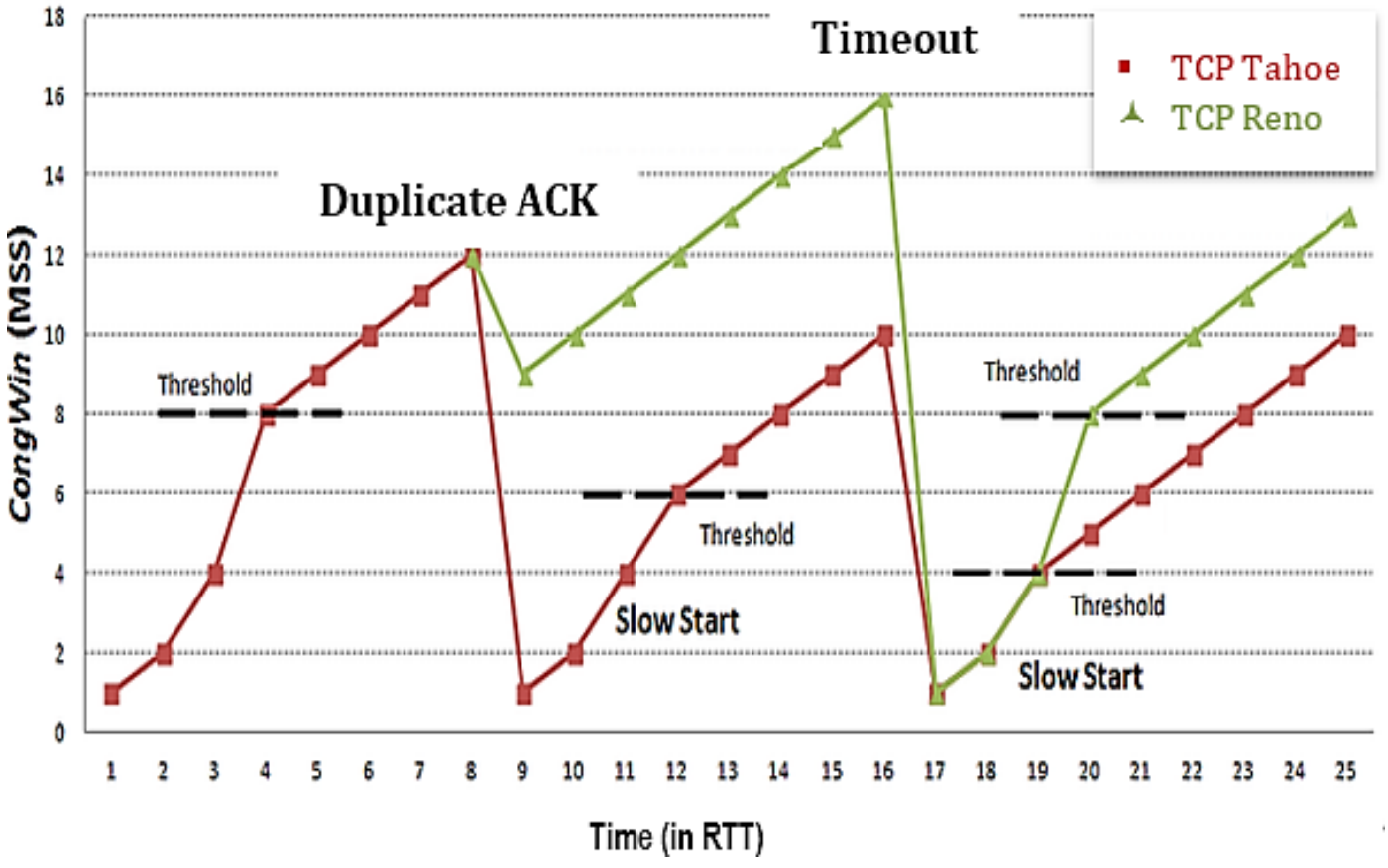

Figure 3: TCP Tahoe vs TCP Reno

TCP NEW RENO

TCP New-Reno includes a small change to the Reno algorithm at the sender that eliminates Reno's wait for a retransmit timer when multiple packets are lost from a window. The change concerns the sender's behavior during Fast Recovery when a **partial ACK** is received that acknowledges some, but not all of the packets that were outstanding at the start of that Fast Recovery period.

In New-Reno, partial ACKs do not take TCP out of Fast Recovery. Instead, partial ACKs received during Fast Recovery are treated as an indication that the packet immediately following the acknowledged Packet in the sequence space has been lost, and should be retransmitted.

Thus, when multiple packets are lost from a single window of data, New-Reno can recover without a retransmission timeout by retransmitting one lost packet per round-trip time until all of the lost packets from that window have been retransmitted.

New-Reno remains in Fast Recovery until all of the data outstanding (when Fast Recovery was initiated) has been acknowledged. In order to exit the Fast Recovery, the source must receive an ACK for the highest sequence number sent before entering Fast Recovery. Thus, unlike TCP Reno, New “partial ACK’s” do not take TCP New Reno out of Fast Recovery.

TCP VEGAS

TCP Vegas is a modification of Reno. It builds on the fact that a proactive measure to encounter congestion is much more efficient than reactive ones. Vegas implements **congestion avoidance** rather than first detecting the congestion and then taking steps to decrease congestion on the channel.

TCP Vegas calculates a base **Round Trip Time (RTT)** and compares it with RTT of packet with recently received ACK. It increases its sending window if the compared RTT is much smaller than the base RTT and if RTT is greater than the base RTT, it decreases its sending window.

THROUGHPUT, PACKET DROP RATE AND LATENCY

- **Throughput** is the rate at which the data is delivered over a TCP connection. It is calculated as total bits received by receiver node divided by the total transfer time.
- **Packet Drop Rate** is an indicator that shows how many packets are lost during the transmission process. It is calculated as the number of dropped packets over total number of packets.
- **Latency** is calculated as summation of RTT’s over total number of packets sent.

FAIRNESS

Congestion control mechanisms for new network transmission protocols must interact well with TCP. **TCP fairness** requires that a new protocol receive no larger share of the network than a comparable TCP flow. If new protocols acquire unfair capacity, they tend to cause problems such as congestion collapse.

DROPTAIL

DropTail is a simple queue mechanism that is used by the routers when packets are needed to be dropped. In this mechanism each packet is treated identically and when queue is filled to its maximum capacity the **newly incoming packets are dropped** until queue has sufficient space to accept incoming traffic. DropTail distributes buffer space unfairly among traffic flows.

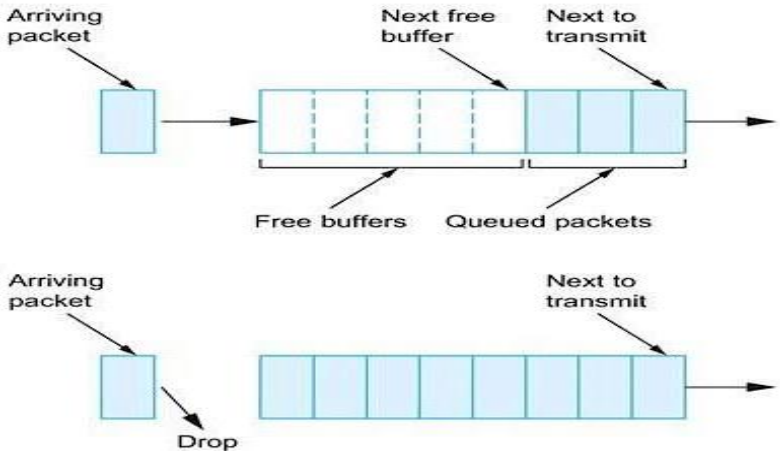

Figure 4: DropTail Queuing Algorithm

RANDOM EARLY DETECTION

Random Early Detection (RED) is a congestion avoidance queuing mechanism. It is active queue management mechanism. It operates on the average queue size and drop packets on the basis of probability. If the buffer is empty, all incoming packets are acknowledged. As the queue size increases, the **packet discarding probability** also increases. When buffer is full all incoming packets are dropped.

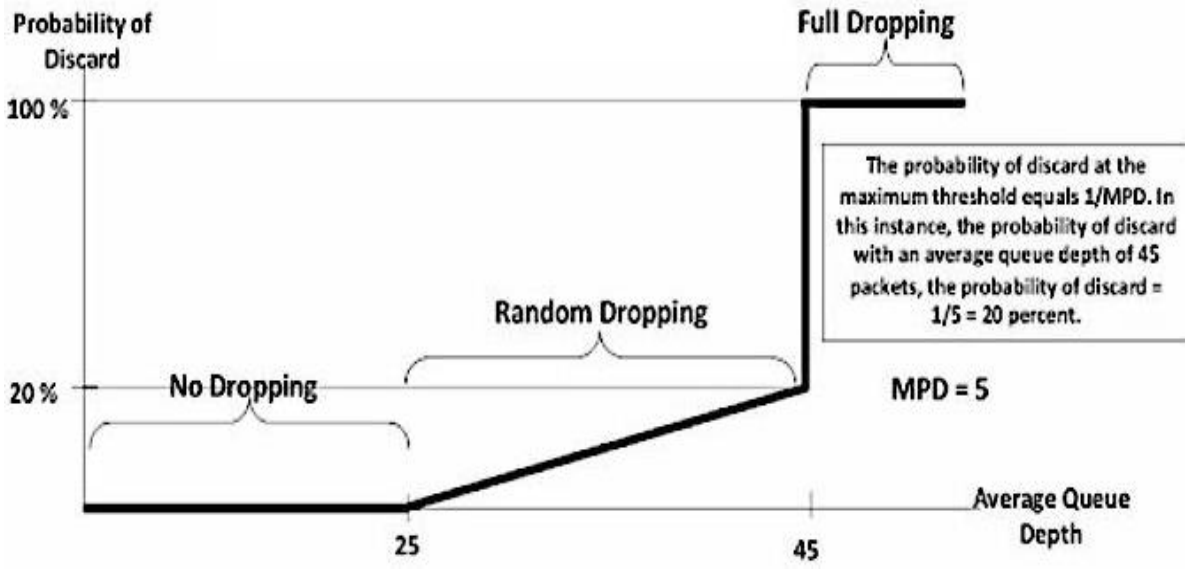

Figure 5: RED Queuing Algorithm

SIMULATION AND ANALYSIS

PROBLEM STATEMENT

Congestion Control refers to techniques that can either prevent congestion, before it happens, or remove congestion after it has happened. Various variants of Transmission Control Protocol (TCP) are available which achieve Congestion Control through their different algorithms. In this project, we have considered five variants of TCP namely:

- TCP Tahoe
- TCP Reno
- TCP New Reno
- TCP Vegas
- TCP SACK

The objective of this project is to provide a comprehensive analysis of TCP Congestion Control techniques including:

- Simulate TCP Congestion in a Network using Cisco Packet Tracer by setting up an appropriate topology involving client and server.
- Analyze and compare the performance of TCP variants by means of performance metrics such as Throughput, Packet Drop Rate and Latency under different load conditions by linearly increasing Constant Bit Rate (CBR) from 1 Mbps to 10 Mbps between two nodes.
- Analyze and compare the Fairness of TCP variants among each other in a network by means of Throughput under different load conditions by linearly increasing Constant Bit Rate (CBR) from 1 Mbps to 10 Mbps between two nodes. Pair of TCP variants analyzed are Reno-Reno, New Reno-Reno, Vegas-Vegas and New Reno-Vegas.
- Analyze and compare the influence of the queuing disciplines namely: DropTail and Random Early Drop (RED) to TCP Reno with and without Selective Acknowledgement (SACK) by means of Average Bandwidth and Average Latency.

The Topology used for analysis is shown in Figure 6. There are 6 Nodes in the network. The link between Node 2 and Node 3 is used to control the CBR in the simulations.

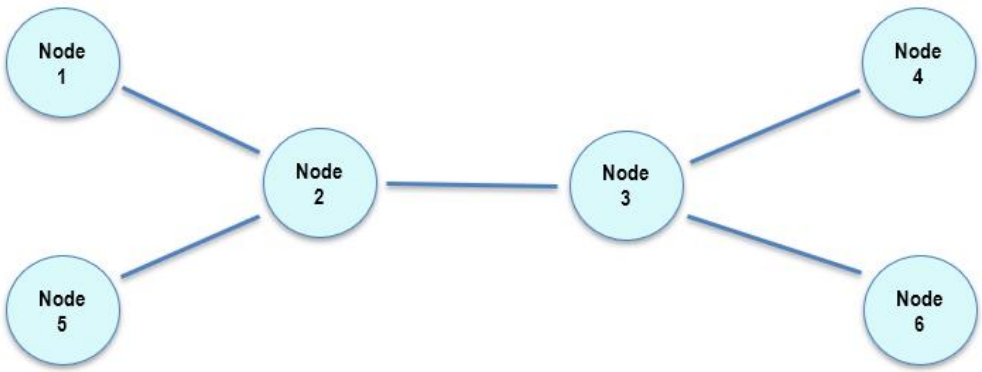

Figure 6: Network Topology for Analysis of TCP variants

SIMULATION OF TCP CONGESTION

In this section, we have simulated TCP Congestion using **Cisco Packet Tracer**. Figure 7 shows the Network topology considered for simulation.

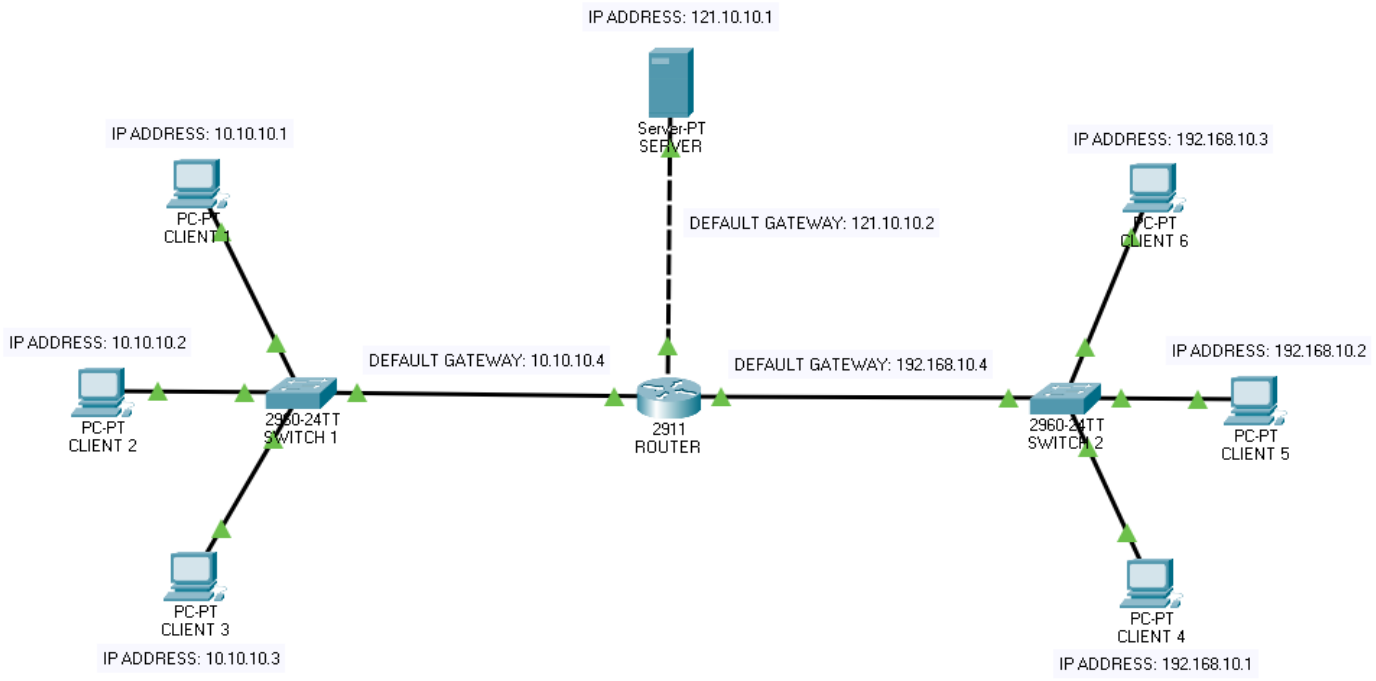

Figure 7: Network Topology for TCP Congestion Simulation

It consists of **6 Client** devices and **1 Server**. The Server is connected to **Central Router** through FastEthernet (100 Mbps) Port. Three Clients are connected with each **Switch** using FastEthernet Ports. The Switches are connected to Central Router through GigabitEthernet (1000 Mbps) Ports. The IP Addresses of all the devices are listed alongside them. The Default Gateways are indicated along the connections.

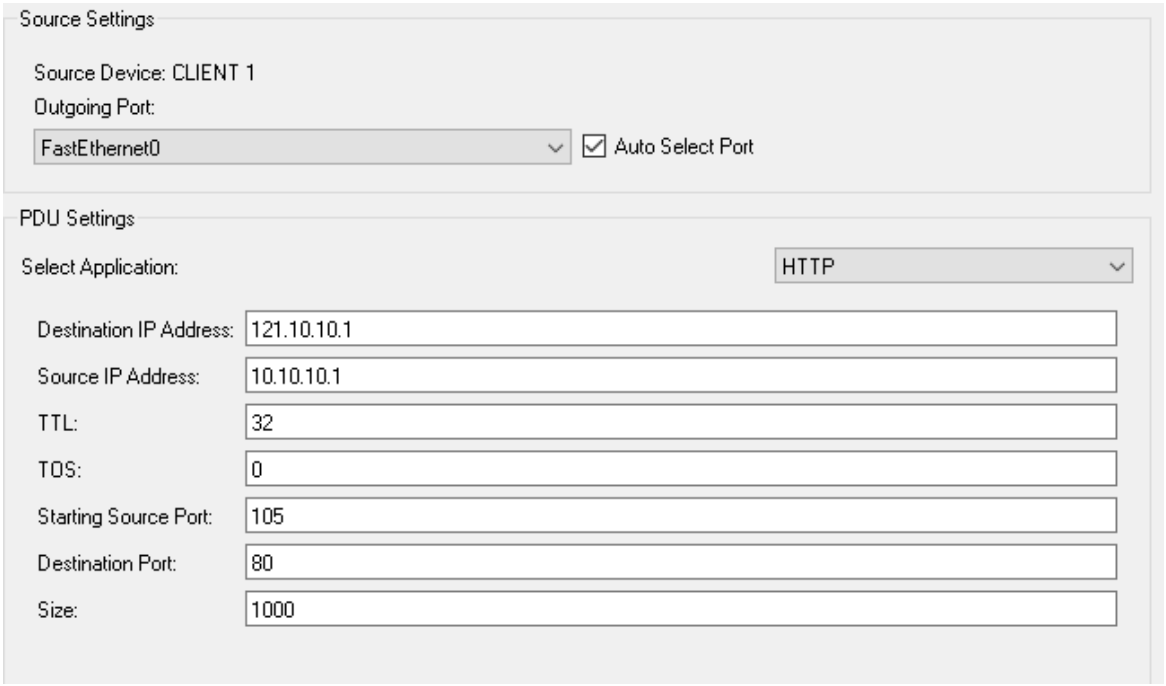

Figure 8: HTTP Traffic from Client 1 to the Server

After setting up the network, **HTTP traffic** is generated from each Client to the Server. It is handled by TCP as the packets leaves the clients and reach the corresponding Switches. Figure 8 shows the HTTP Traffic generated from Client 1 to the server.

Time(sec)	Last Device	At Device	Type
0.001	CLIENT 1	SWITCH 1	TCP
0.001	CLIENT 2	SWITCH 1	TCP
0.001	CLIENT 3	SWITCH 1	TCP
0.001	CLIENT 6	SWITCH 2	TCP
0.001	CLIENT 5	SWITCH 2	TCP
0.001	CLIENT 4	SWITCH 2	TCP

Figure 9: Packet Transfer from Clients to Switches

The Packets are queued at Switches before leaving for Central Router. Figure 10 shows the Packets from Clients queued at Switch 1 and 2.

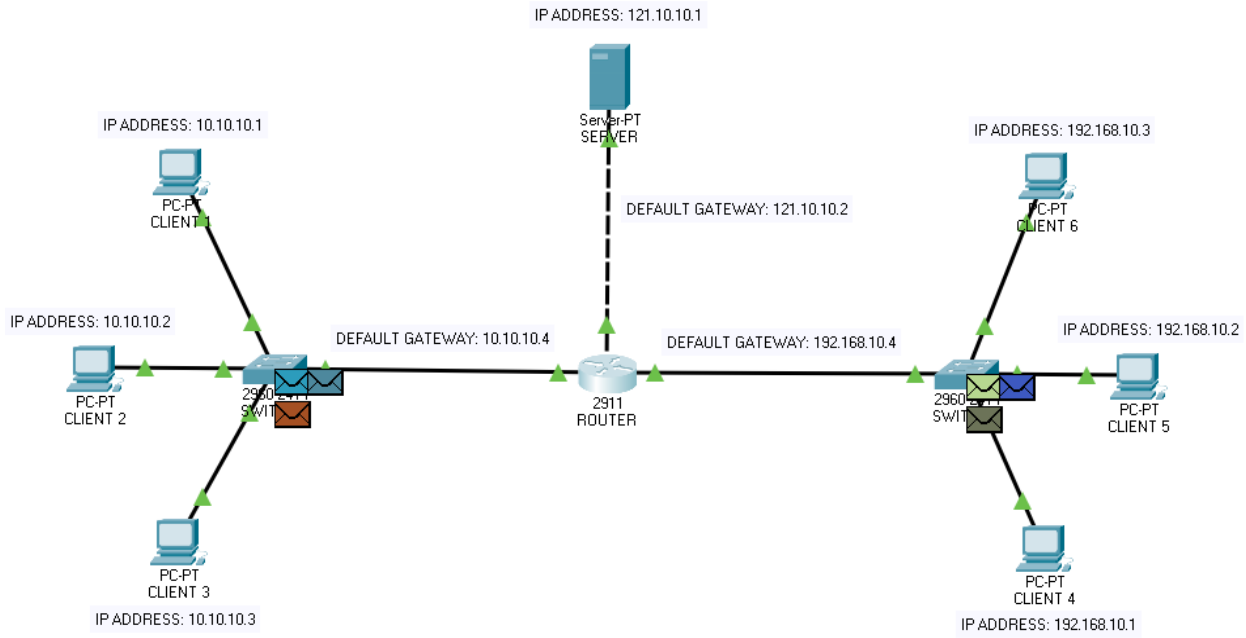

Figure 10: All Packets queued at Switches 1 and 2

A large amount of HTTP Traffic leads to Packet drops at Central Router as well as Server. Figure 11 shows the **Packet Drop at Central Router** due to Congestion in Network.

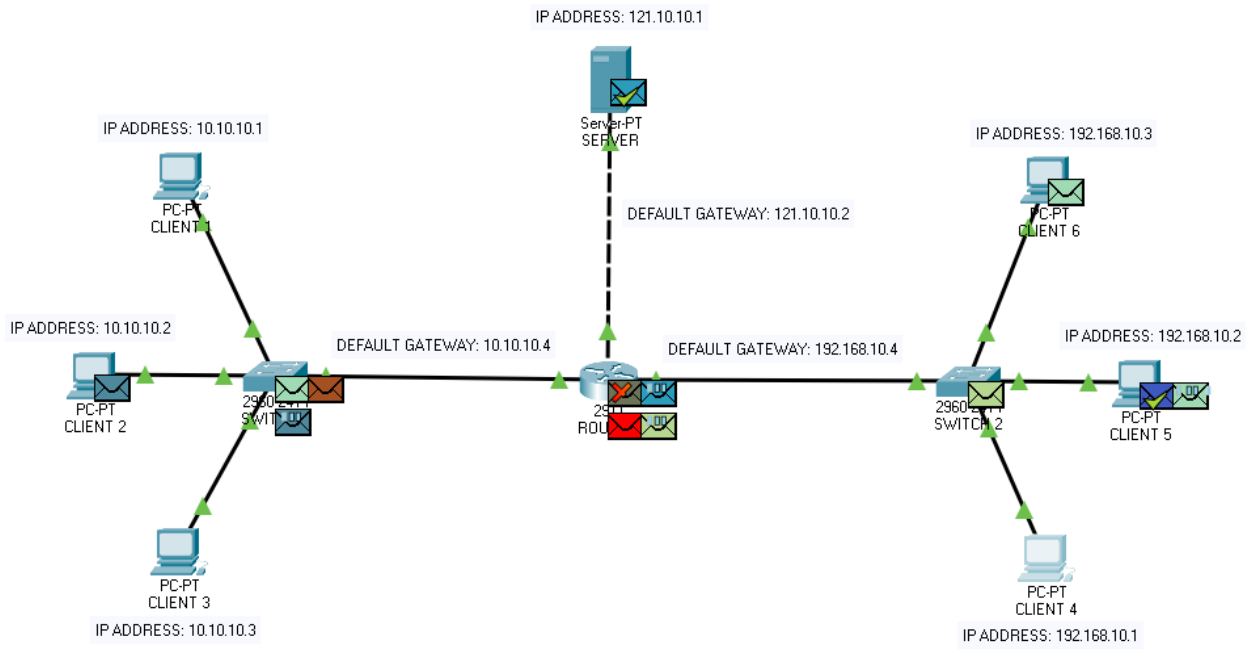

Figure 11: Packet Drop at Central Router due to Congestion

Packet transmission time from Switches to Router is very less due to use of Gigabit connections. This leads to large Packet Queuing at Central Router and thus floods the Server with Packets leading to Packet Loss. Figure 12 shows **Packet loss at Server** due to Congestion.

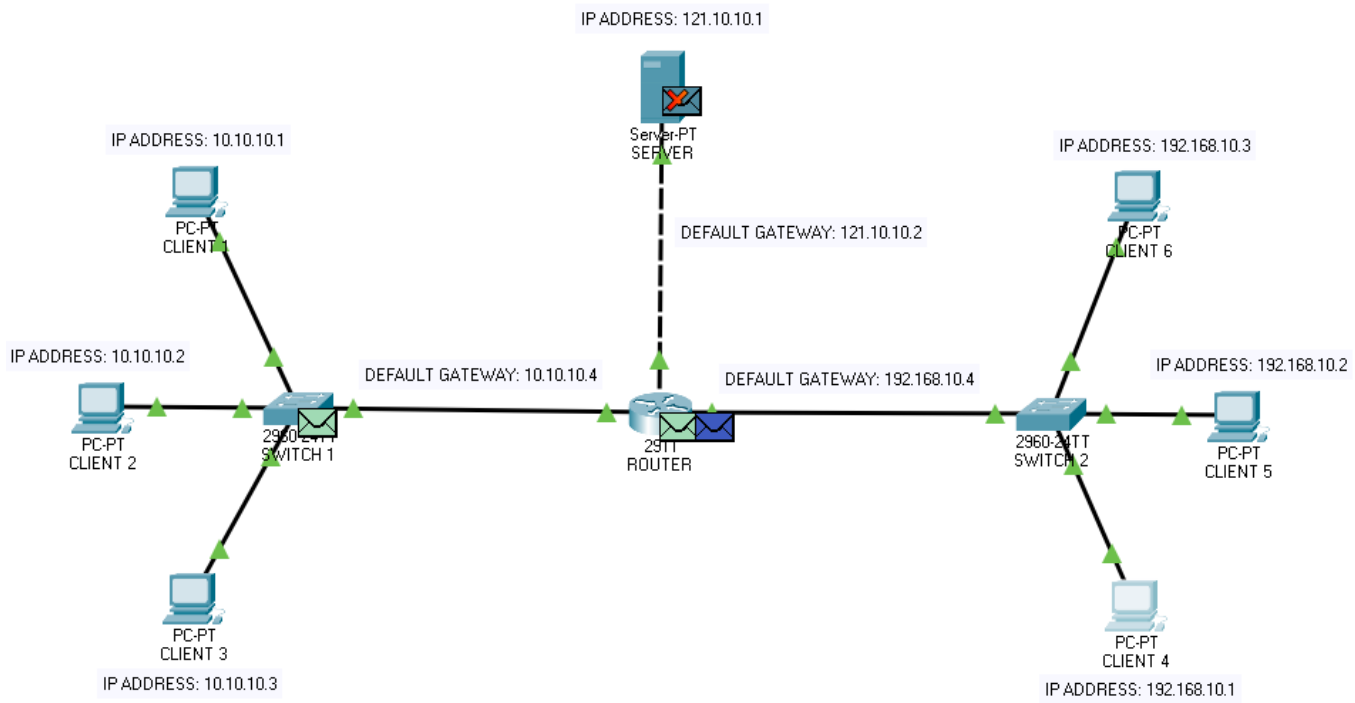

Figure 12: Packet Drop at Server due to Congestion

Figure 13 shows the information of Packet dropped at the server. Source Port can be seen as 105 which belongs to Client 1 while Destination Port is 80 belonging to the Server itself.

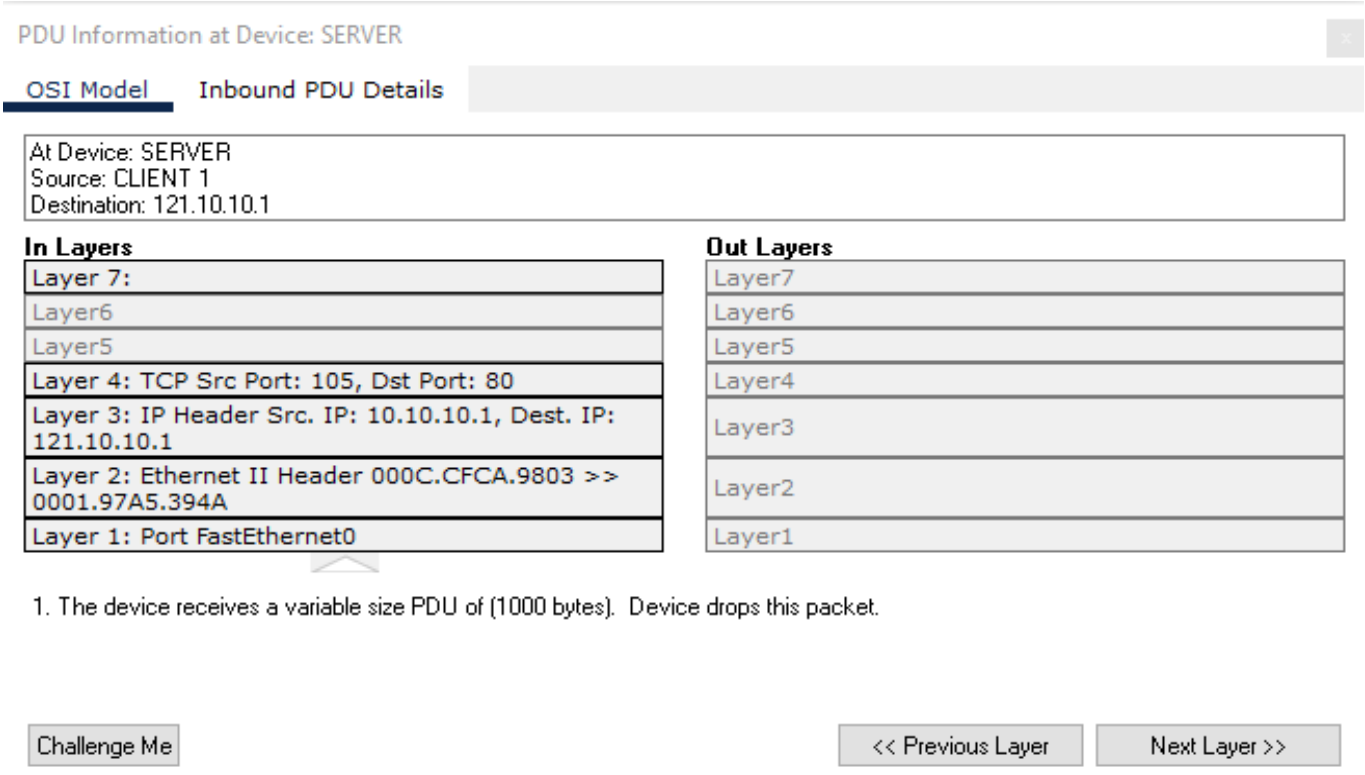

Figure 13: Information of the Dropped Packet at Server

PERFORMANCE COMPARISON OF TCP VARIANTS

In this section, we have analyzed and compared the performance of TCP variants by means of **Throughput, Packet Drop Rate** and **Latency** under different load conditions by linearly increasing **Constant Bit Rate (CBR)** from 1 Mbps to 10 Mbps between two nodes at a granularity of 0.01 Mbps. Figure 14 shows the **Network Topology** and **Flow Setup** used for analysis. The CBR flow from Node 2 to Node 3 is the only varying quantity. The performance of each TCP variant is measured for flow from Node 1 to 4.

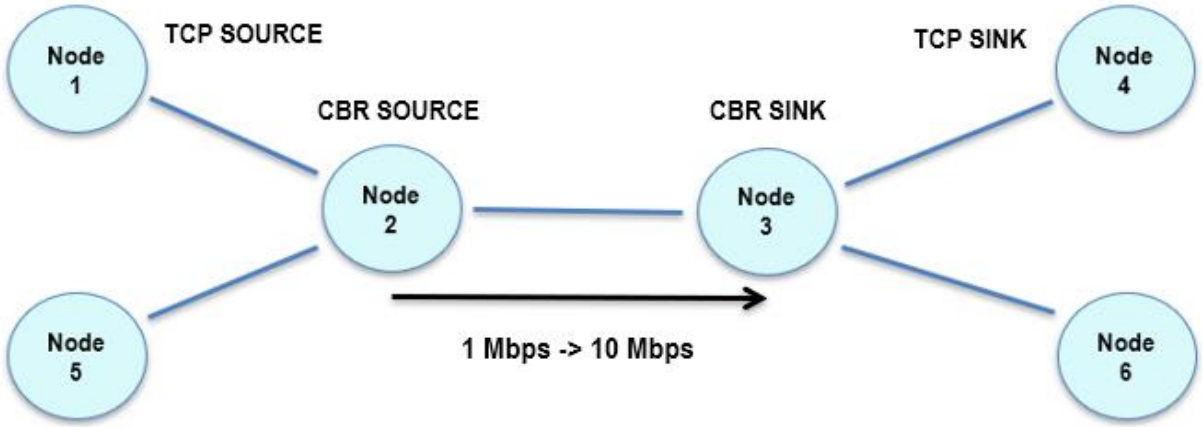

Figure 14: Network Topology and Flow Setup

- **THROUGHPUT**

Figure 15 shows the Throughput comparison of TCP variants. When CBR between Node 2 and 3 is less than 7.5Mbps, the throughput for all TCP variants is almost similar.

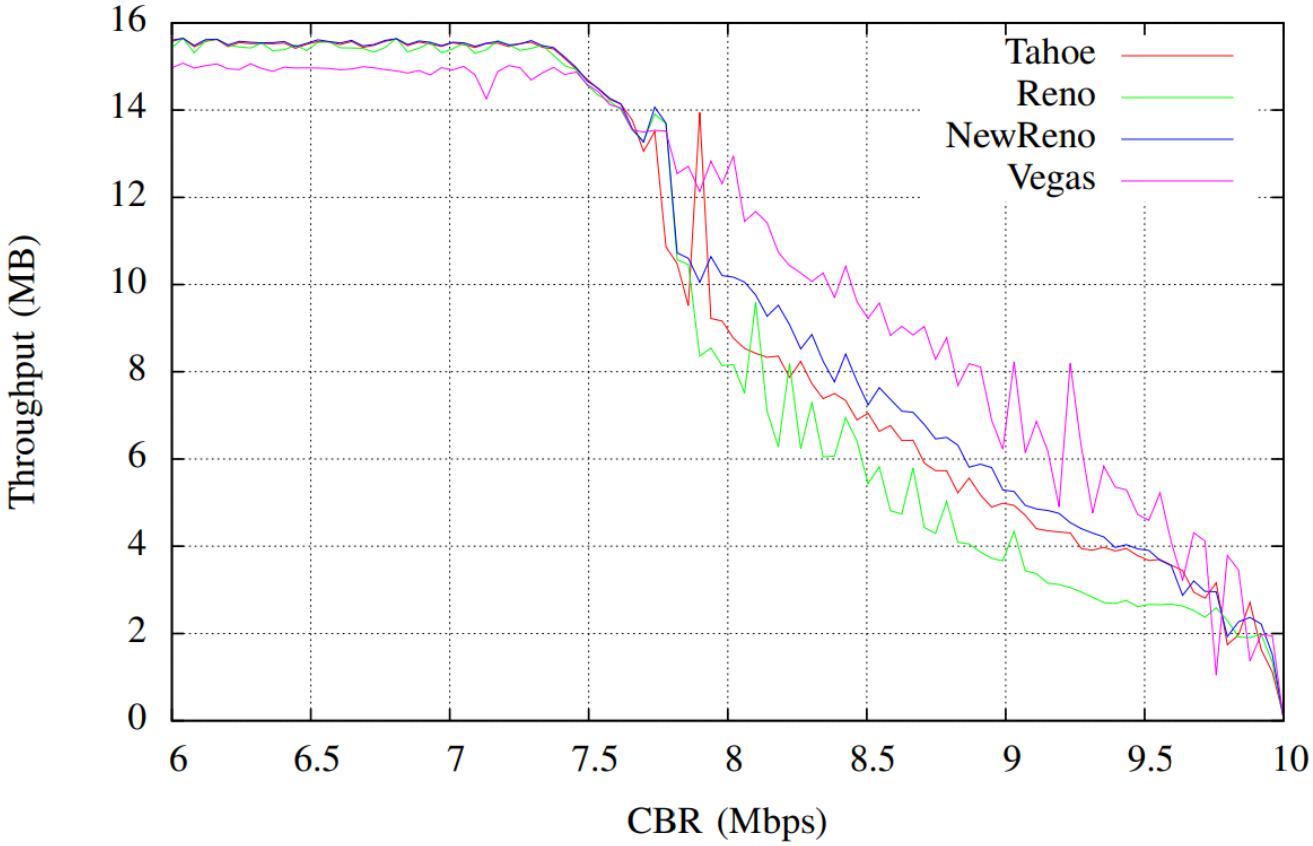

Figure 15: Throughput of TCP variants

However, when the CBR flow increases and the network start to congest, TCP Vegas performs better and have the highest average throughput as compared to other three, while TCP Reno has the lowest throughput. This is mainly because TCP Vegas detects congestion at an early stage based on the increasing RTT values of the packets unlike other variants such as TCP Reno or TCP New Reno. Hence, Vegas transmit less data when the network is not congested since it conducts more congestion detection. Thus, the good throughput of Vegas, as the network becomes congested, is the result of its pre-detection of the network congestion.

- **PACKET DROP RATE**

Figure 16 shows the Packet Drop Rate comparison of TCP variants. Initially, when the CBR is less than 7.5Mbps, the packet drop rate is very less.

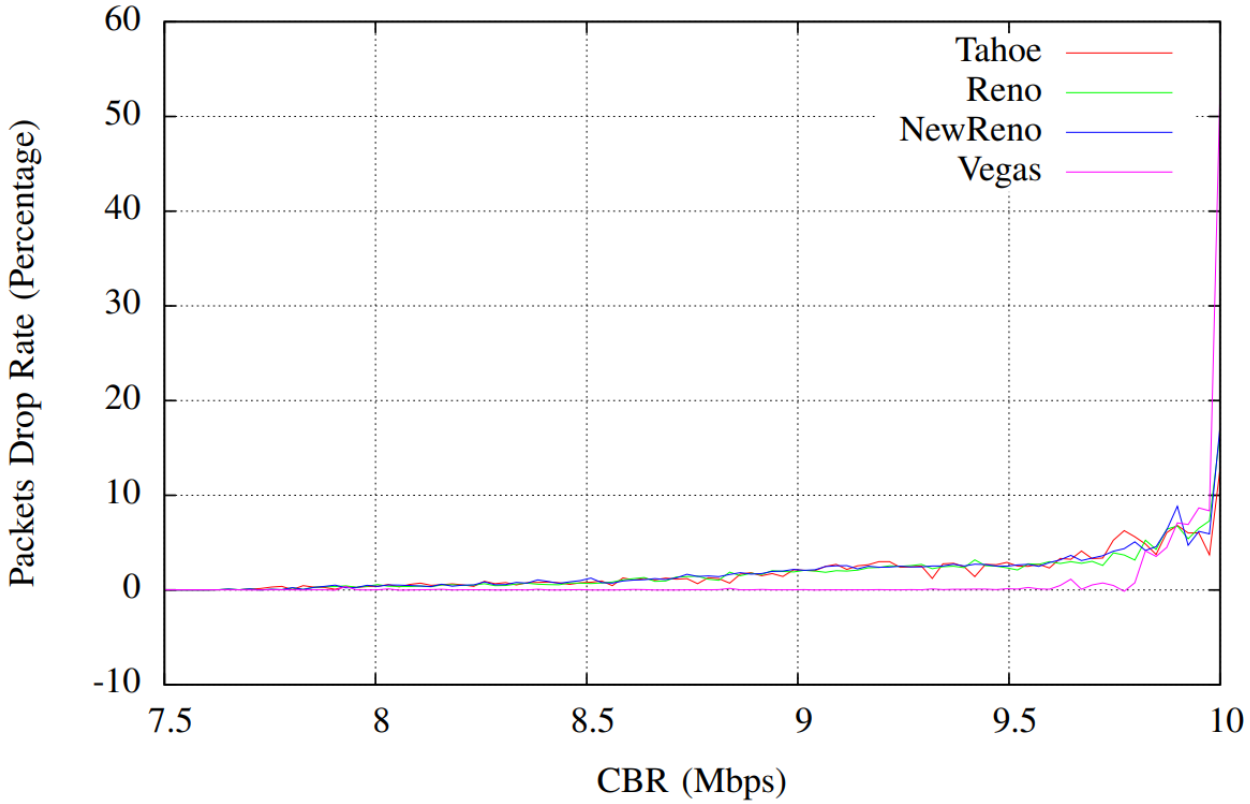

Figure 16: Packet Drop Rate of TCP variants

When CBR flow gets dominant but has not occupied all the bandwidth in the network, TCP Vegas performs better by dropping less packets while the other 3 has similar higher drop rate. TCP Vegas runs steadier than others with the increase of CBR. When CBR increases to the bandwidth limitation, Vegas has a 50% drop rate compared with the other 3 because when the packet drop rate remained constant during the initial stages, the window increased, and doubled the input from the previous transfer. However, once the queue got filled, the arrived packets that are large in number are dropped, and the window reduces drastically to half of its current value. However, this cannot totally deny that Vegas is better than Tahoe, Reno and New Reno on packets drop rate in congestion, because the CBR flow has occupied the entire bandwidth.

- **LATENCY**

Figure 17 shows the Average Latency comparison of TCP variants. TCP Tahoe, Reno, New Reno and Vegas, all exhibit the same latency till CBR is less than 8.5 Mbps, but thereafter the average latency for each TCP variant varies.

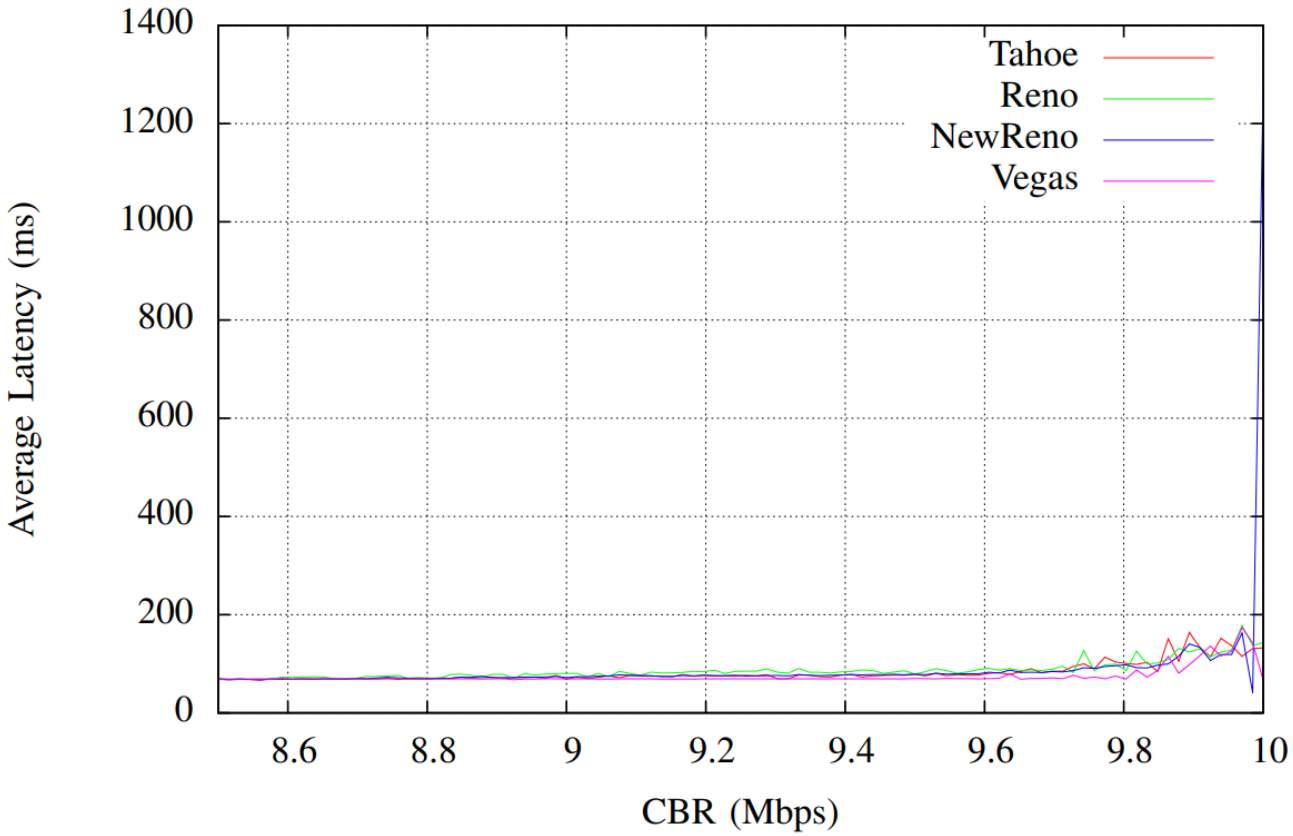

Figure 17: Average Latency of TCP variants

TCP Vegas detects congestion at the initial stage, and it queues the packets when the RTT received is greater than the base RTT. Hence, Vegas has the lowest average latency. The other three variants have comparatively higher latency than TCP Vegas. The other variants continue to increase their window size until a packet loss is detected after which they enter into congestion avoidance mechanism. They react to the congestion only after the congestion has started and packet drops tend to increase, which thereby causes high latency in packet transmission. Only TCP New Reno has a latency blast when CBR reaches to the bandwidth limitation.

- **RESULTS**

The analysis concludes that **Vegas performs the best in this scenario**. It has evidently better throughput performance in a congested network, and trivial superiority on losing fewer packets, controlling latency and running steady. As CBR flow is increased, the performance of TCP decreases because TCP being a reliable protocol, will only send the next window of packets only when the acknowledgment for previous packets are received. **Hence, the throughput of any TCP variant should decrease with the increase in CBR flow, while the TCP packets drop rate and transmission latency should decrease.**

FAIRNESS BETWEEN TCP VARIANTS

In this section, we have analyzed and compared the **Fairness of TCP variants** among each other in a network by means of **Throughput** by linearly increasing Constant Bit Rate (CBR) from 1 Mbps to 10 Mbps between two nodes. Figure 18 shows the Network Topology and Flow Setup used for analysis. One TCP variant is used along Node 1 to Node 4 and other is used along Node 5 to Node 6. **Pair of TCP variants analyzed are Reno-Reno, New Reno-Reno, Vegas-Vegas and New Reno-Vegas**.

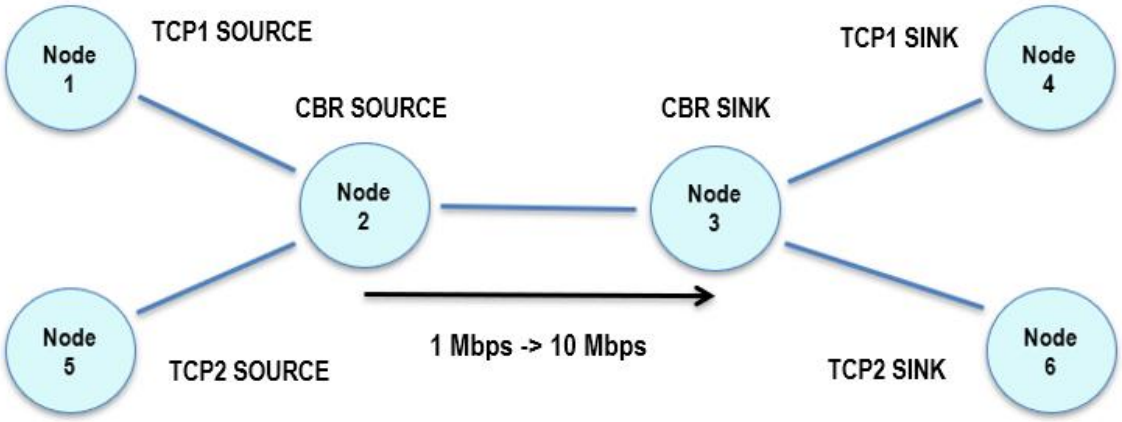

Figure 18: Network Topology and Flow Setup

- **RENO-RENO**

In Figure 19, there is certain oscillation in the throughput, the two Reno lines follow similar tracks and keep close with the increase of CBR. The fairness refers that the two TCP agents could alternately occupy more bandwidth. Hence, Reno-Reno TCP pair is fair.

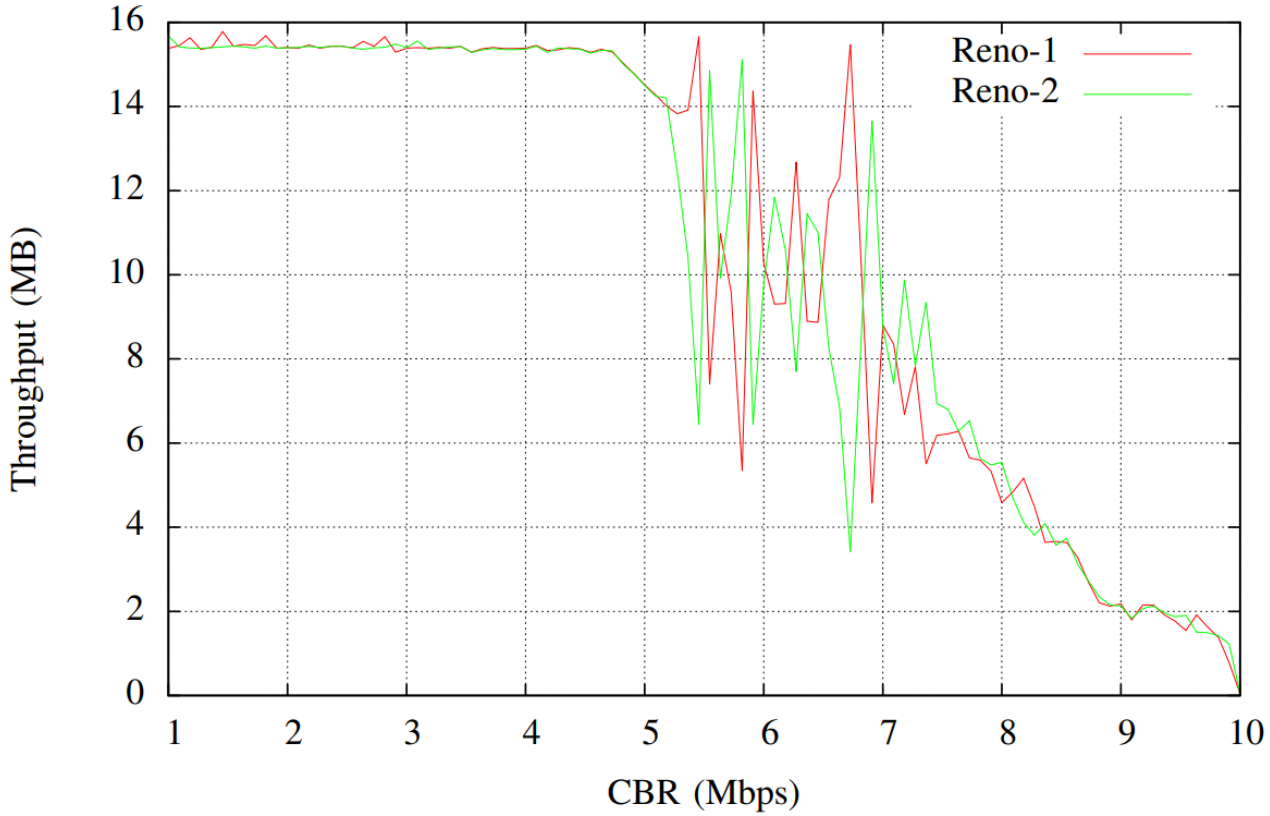

Figure 19: Fairness of Throughput on Reno-Reno

- **NEW RENO-RENO**

In Figure 20, Reno TCP has lower throughput than New Reno when CBR gets larger than 5Mbps. So New Reno is not that fair to Reno TCP, but it does not suppress Reno's performance too much.

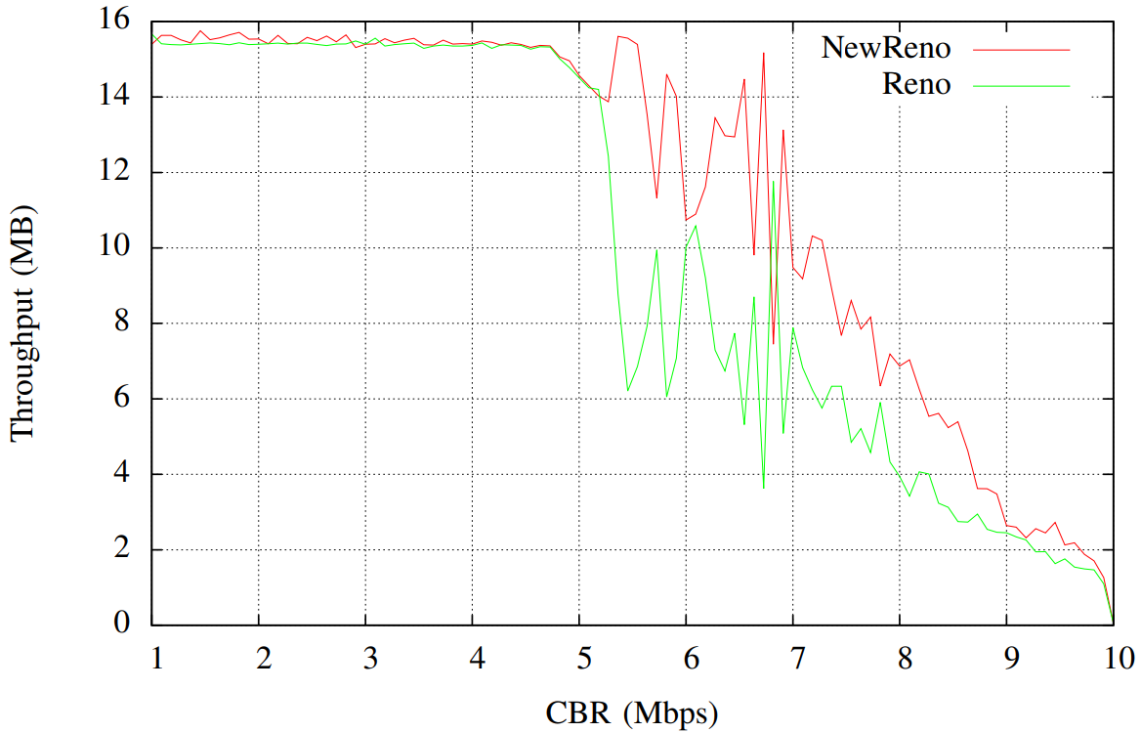

Figure 20: Fairness of Throughput on New Reno-Reno

- **VEGAS-VEGAS**

In Figure 21, there are certain oscillations in the throughput, the two Vegas lines follow similar tracks and keep close with the increase of CBR. The oscillation is acceptable, since only one TCP can have higher bandwidth. Hence, Vegas-Vegas TCP pair is fair.

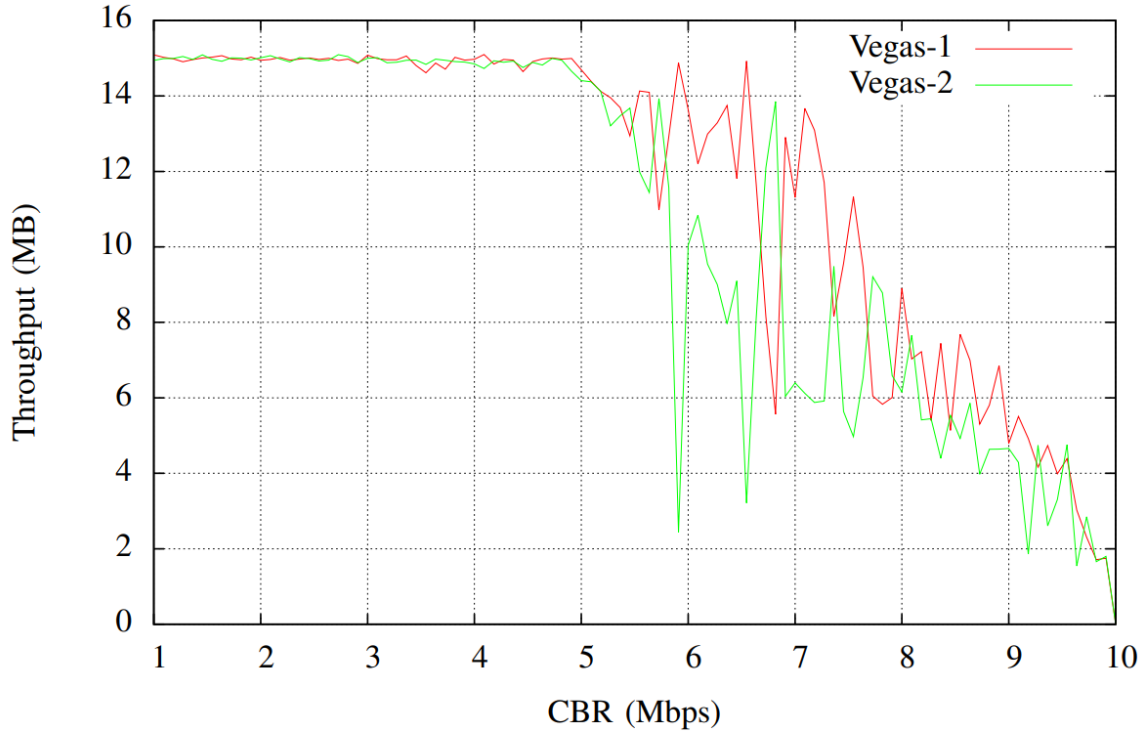

Figure 21: Fairness of Throughput on Vegas-Vegas

- **NEW RENO-VEGAS**

With the increase of CBR, New Reno has higher throughput than Vegas TCP and this gap grows larger. It runs steadier and loses fewer packets than Vegas.

New Reno is unfair to Vegas. This is mainly because Vegas detects congestion early and reduces its sending rate before New Reno, so that it gives greater bandwidth to the co-existing New Reno TCP flow, which make New Reno dominant in the network.

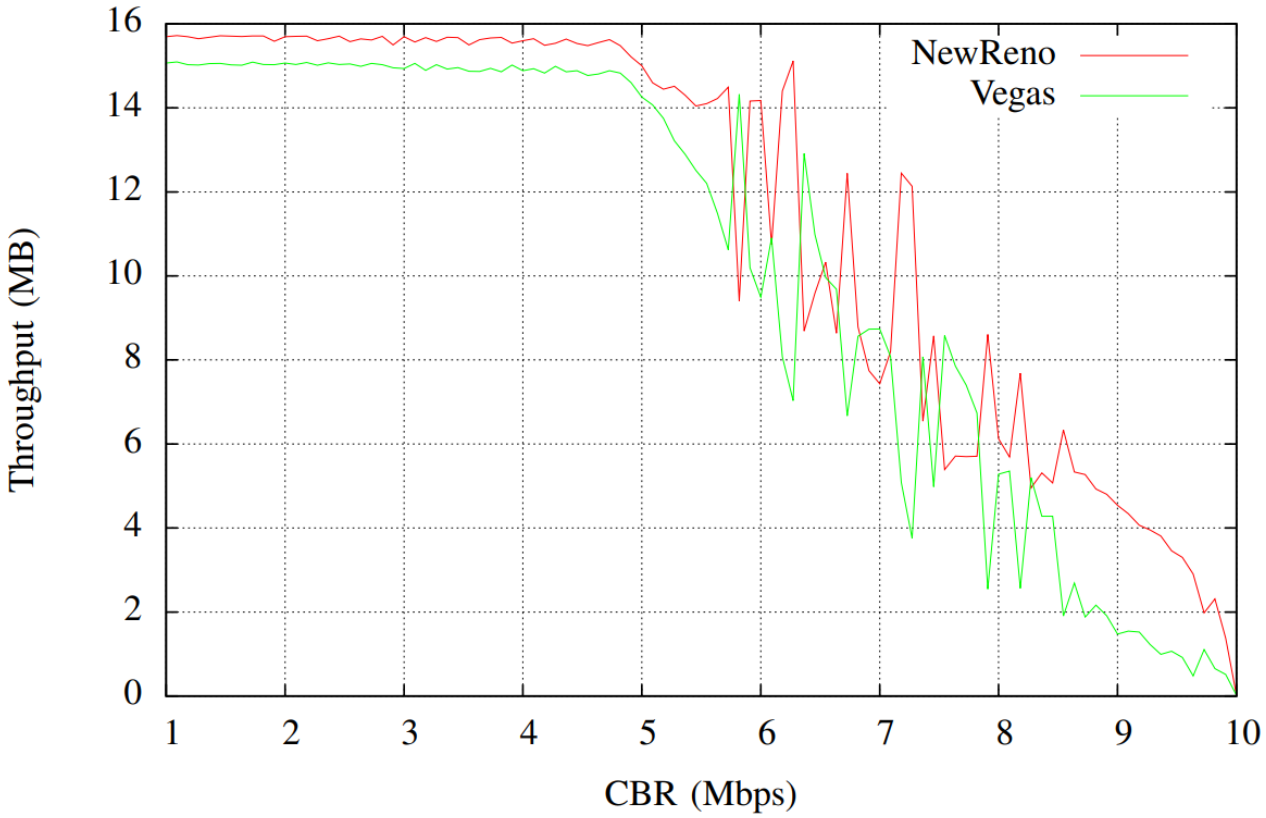

Figure 22: Fairness of Throughput on New Reno-Vegas

- **RESULTS**

The analysis concludes that same **TCP variant is usually fair to each other on throughput, while different variants cannot be completely fair to the co-existing variants** because different variants have different implementation details such as how to do congestion control or avoidance and retransmission strategy.

Thus, the differences between TCP variants implementation algorithms might make one variant get suppressed by another. For example, the early congestion detection feature of Vegas makes it suppressed by New Reno, and once New Reno becomes dominant, Vegas will always assume the network is becoming congested and reduce its sending rate, hence never gets dominant back again.

Hence, Reno-Reno and Vegas-Vegas are fair to each other in same network, but on the other hand, New Reno-Reno and New Reno-Vegas are not fair to each other.

INFLUENCE OF QUEUING ALGORITHMS

In this section, we have analyzed and compared the **influence of the queuing disciplines** namely: **DropTail** and **Random Early Drop (RED)** on TCP Reno with and without **Selective Acknowledgement (SACK)** by means of Average Latency and Average Bandwidth. The total duration is taken as 200 seconds. TCP flow starts initially. After 50 seconds, CBR flow starts. CBR flow and TCP flow are stopped after 150 and 200 seconds resp. The bandwidth of each link is 10Mbps, TCP window size is 200 and the CBR flow rate is 3Mbps.

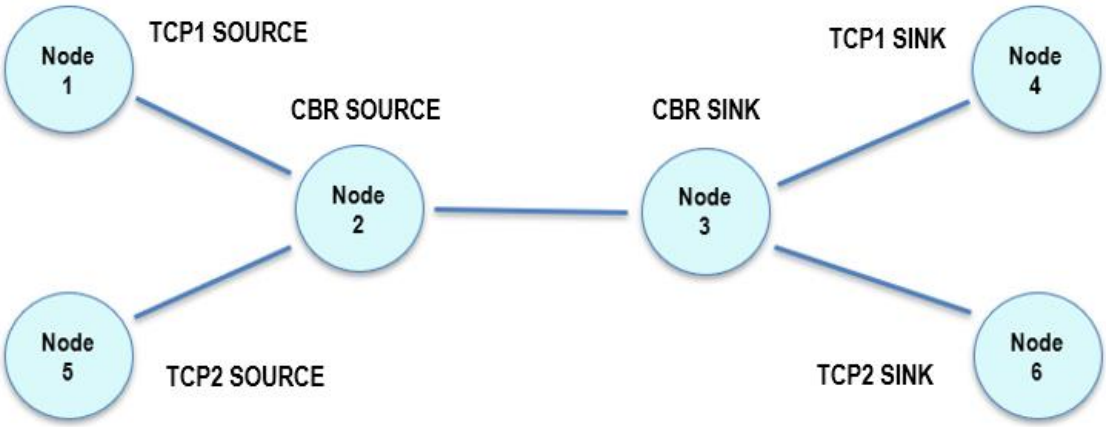

Figure 23: Network Topology and Flow Setup

- **AVERAGE LATENCY**

For Reno, it has a smaller latency with DropTail than with RED, except when CBR flow starts. For SACK, it has a larger latency with DropTail than with RED. However, Reno and SACK both with DropTail show similar nature. Thus, for Reno, DropTail has a better performance on latency than RED and for SACK, RED has a better performance on latency.

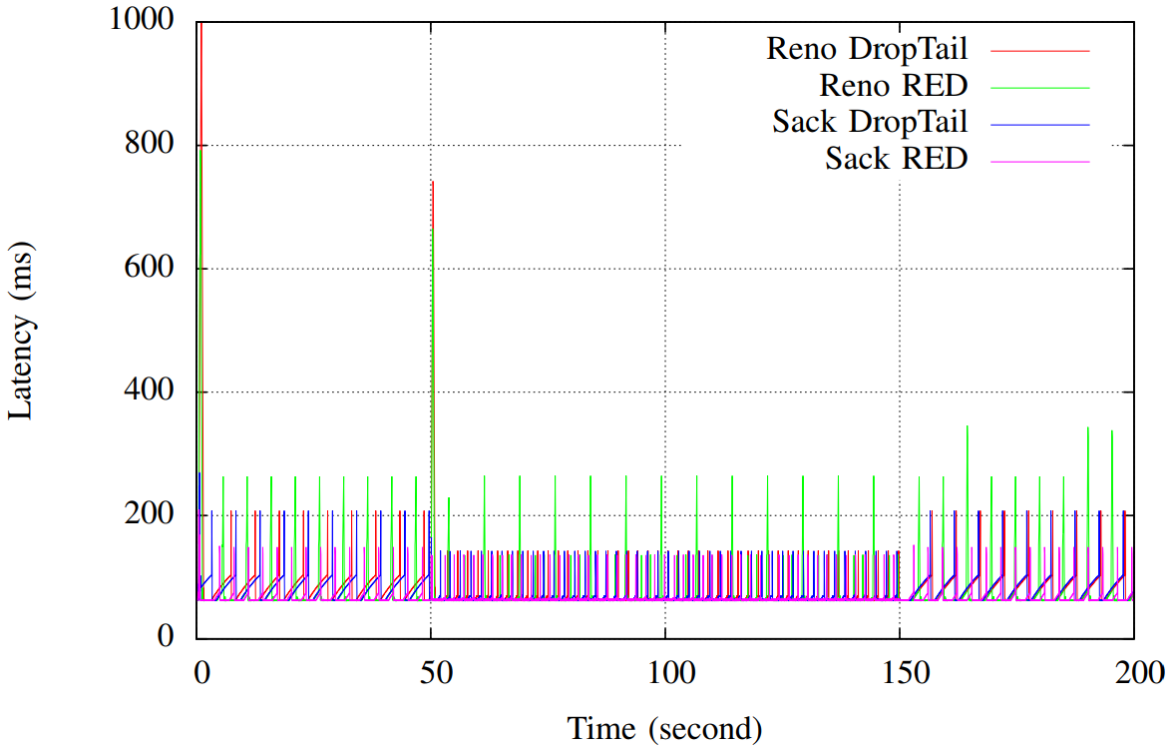

Figure 24: Bandwidth on Delta Time

- **AVERAGE BANDWIDTH**

When the CBR flow has not started, regardless of the TCP invariant, the TCP flows with DropTail always has a higher bandwidth than those with RED queue type. After CBR flow starts at about 50s, the DropTail TCP flows still have higher bandwidth than the RED ones. Thus, for Reno and SACK, DropTail queue type has better bandwidth performance than RED, so that queuing discipline does not provide fair bandwidth to each flow.

Reno with RED queue type has the worst performance in bandwidth, because RED will drop some packets even before the queue is full. Once a packet is dropped, Reno enters fast recover and changes its congestion window size to the half of the current threshold size. That is why even without CBR flow, Reno with RED still has a low bandwidth. On the other hand, the bandwidth of Reno with RED does not experience a sharp decrease when the CBR flow starts. Reno with DropTail experiences a sharp decrease when CBR flow starts, because it is more likely to drop packets continuously until getting a packet timeout, and then enters fast recovery phase, using slow start to retransmit the dropped packet again.

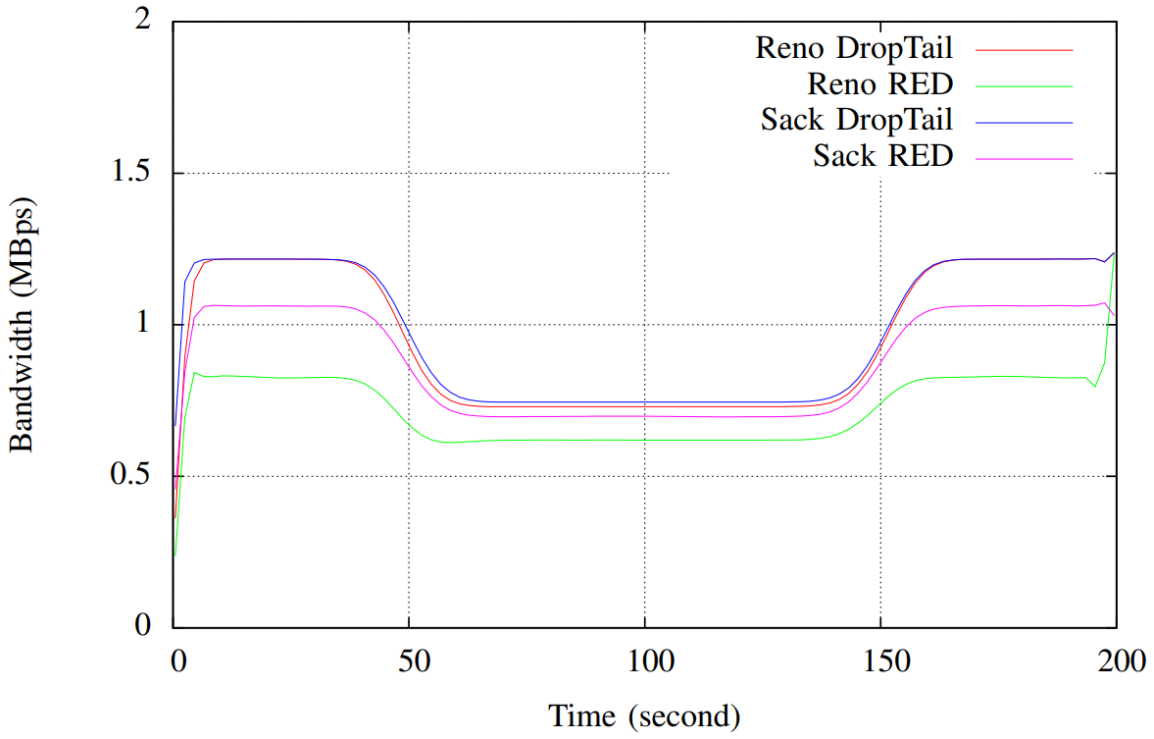

Figure 25: Bandwidth on Delta Time

- **RESULTS**

The analysis concludes that **RED works well with SACK**. As for latency, SACK with RED has a better performance than SACK with DropTail, because RED drops packet randomly while SACK retransmits packets selectively. Although SACK with DropTail takes relatively larger bandwidth than SACK with RED, SACK with RED still provide more stable and reliable transmission especially in congestion.

CONCLUSION

In this project, we have simulated TCP Congestion in a network and analyzed some of the TCP variants on the basis of performance, fairness and influences on them due to queuing algorithms.

TCP variants considered for simulation and analysis include:

- TCP Tahoe
- TCP Reno
- TCP New Reno
- TCP Vegas
- TCP SACK

In the Simulation and Analysis section:

- Firstly, we have simulated TCP Congestion in a Network using Cisco Packet Tracer Simulator by setting up an appropriate topology involving client and server. After setting up the network, HTTP traffic, which is handled by TCP, is generated from each Client to the Server and Packet Loss at Server has been finally shown.
- Secondly, Performance Comparison of TCP variants has been done which is based on throughput, packet drop rate and latency. The results conclude that TCP Vegas performs comparatively better than TCP Tahoe, Reno and New Reno.
- Thirdly, analysis of Fairness between TCP variants has been done which is based on throughput. We found that same TCP variants are usually fair to each other in the same network, while different variants might suppress the co-existing ones.
- Lastly, we analyzed and compared the influence of the queuing disciplines namely: DropTail and Random Early Drop (RED) on TCP Reno with and without Selective Acknowledgement (SACK) by means of Average Bandwidth and Average Latency. Results show that RED works well with SACK TCP, which provides low-latency and more stable transmission.

LEARNING

In this project, we studied and simulated different TCP variants for TCP Congestion Control. By understanding these algorithms, we came to know how these variants could be used to solve real life network congestion problems. This project helped us to have a better **understanding of TCP Congestion Control** and become familiar with different types of variants available to deal with various types of network congestion situations. We performed **TCP Congestion Simulations** using **Cisco Packet Tracer Simulator**. During the course of this project, we used **Network Simulator** for analytical purposes. Above all, we learnt a lot about team work and time management.

REFERENCES

- https://en.wikipedia.org/wiki/TCP_congestion_control
- https://www.tetcos.com/downloads/TCP_Congestion_Control_Comparison.pdf
- https://en.wikipedia.org/wiki/Network_congestion
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- https://accedian.com/blog/measuring-network-performance-latency-throughput-packet-loss/
- https://en.wikipedia.org/wiki/Packet_Tracer
- https://en.wikipedia.org/wiki/Network_simulation