Performance Analysis of TCP Variants

Medhavi Mahansaria
Northeastern University, Boston, MA 02115
Email: mahansaria.m@husky.neu.edu
NUID: 001612685

Aswin Gopalan
Northeastern University, Boston, MA 02115
Email: gopalan.a@husky.neu.edu
NUID: 001682521

Abstract- Transmission Control Protocol (TCP) is a core Internet protocol. Along with the Internet Protocol (IP), TCP/IP are the most frequently used protocols in the Internet. The original design of the Transmission Control Protocol (TCP) worked reliably, but was unable to provide acceptable performance in large and congested networks. We compare different TCP variants (Tahoe, Reno, NewReno and Vegas) on a simple network topology, using the NS-2 simulator, and analyze their performance based on throughput, packet drop rate, and latency under congestion and compare the fairness between them. We have also studied the influence of queuing disciplines DropTail and Random Early Drop (RED) on TCP SACK and Reno.

1. Introduction

The most common transport protocol used in internet for data transmission is TCP. TCP's primary purpose is to provide a connection oriented, reliable data transfer service between different applications to be able to provide these services on top of an unreliable communication system. The main purpose of this paper is to perform a complete analysis of the different TCP variants under different conditions to generalize to TCP behavior outside of the simulated environment. There are many variants of TCP, each variant being used for a specific purpose. The five variants we use in this paper are Tahoe, Reno, NewReno Vegas and SACK. Each of these variants were designed to address some specific issues and each have a place of their own under different circumstances. Conducting this experiment will allow us to understand the execution of various TCP variants under different parameters.

The various TCP variants referred to in this paper are described below:

A. TCP Tahoe

Tahoe is the most simplest TCP variant. If three duplicate ACKs are received, Tahoe performs a fast retransmit, sets the slow start threshold to half of the current congestion window, reduces the congestion window to 1 MSS, and resets to slow start state. If an ACK timeouts (RTO timeout), slow start is used, and reduce congestion window to 1 MSS.

B. TCP Reno

If three duplicate ACKs are received, Reno will perform a fast retransmit and skip the slow start phase by instead halving the congestion window (instead of setting it to 1 MSS like Tahoe), setting the slow start threshold equal to the new congestion window, and enter a phase called Fast Recovery. In this state, TCP retransmits the missing packet that was signaled by three duplicate ACKs, and waits for an acknowledgment of the entire transmit window before returning to congestion avoidance. If there is no acknowledgment, TCP Reno experiences a timeout and enters the slow start state. If an ACK timeouts (RTO timeout), slow start is used, and reduce congestion window to 1 MSS

C. TCP NewReno

NewReno was designed to improve TCP Reno performance. It improves retransmission during the fast-recovery phase of TCP Reno. During fast recovery, for every duplicate ACK that is returned to TCP New Reno, a new unsent packet from the end of the congestion window is sent, to keep the transmit window full. For every ACK that makes partial progress in the sequence space, the sender assumes that the ACK points to a new hole, and the next packet beyond the ACKed sequence number is sent. Because the timeout timer is reset whenever there is progress in the transmit buffer, this allows New Reno to fill large holes, or multiple holes

D. TCP Vegas

timeouts were set and round-trip delays were measured for every packet in the transmit buffer. In addition, TCP Vegas uses additive increases in the congestion window. This variant was not widely deployed. In a comparison study of various TCP congestion control algorithms, TCP Vegas appeared to be the smoothest

E. TCP SACK

Tahoe, Reno and NewReno all acknowledge cumulative packets therefore are unable to detect multiple lost

packets per round trip time. SACK's selective acknowledgements algorithm deals effectively with multiple packets lost.

2. Methodology

We have used NS-2 network simulator to conduct all our experiments. NS-2 stands for Network Simulator version 2. NS-2 is a discrete event simulator targeted at networking research. It works at the packet level. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. We have specifically used NS-2 because it has extensive built in support, reliable, open source and universally accepted. The NS-2 simulation to achieve the performance between different TCP variants are performed over the following network topology.

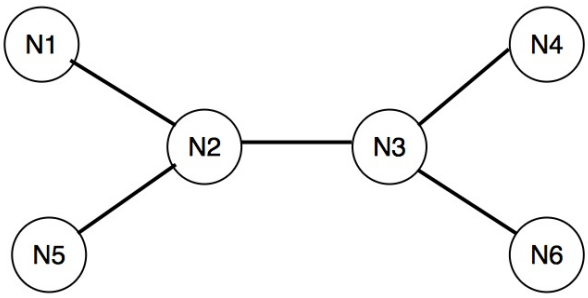

Figure 1: Network Topology

N1, N2, N3, N4, N5 and N6 represent nodes over which we design our experiments. Each of nodes are connected by a full-duplex link which has a bandwidth of 10 Mbps with a delay of 5ms. We have kept the packet size as 1040 and the TCP window size as 20 for all experiments.

In this paper, we have conducted three different experiments to understand clearly the performance of TCP variants including Tahoe, Reno, NewReno, Vegas and SACK. This network topology was executed under different conditions in NS-2 to generate trace files. These trace files were parsed to analyze the behavior of TCP variants under the given conditions.

Experiment 1

We analyze the performance of TCP variants (Tahoe, Reno, NewReno, and Vegas) like throughput, latency and packet drop rate under the influence of various load conditions based on the topology shown in Figure 1. For this experiment, a TCP flow is set from N1 to N4 and a varying Constant Bit Rate (CBR) flow is set from N2 to N3. We have analyzed our experiments based on the following conditions : 1) Introduce a TCP flow first, let it stabilize, introduce CBR flow wait for TCP to stabilize 2) Introduce CBR first, TCP later and then keep varying CBR from 1 Mbps to 10 Mbps 3) Start both TCP and CBR flows simultaneously and increase CBR from 1 Mbps to 10 Mbps. The performance of TCP variants is analyzed by plotting the graphs for each performance attribute with respect to CBR flow rate.

Experiment 2

Out on the Internet, there are many different operating systems, each of which may use a different variant of TCP. In this experiment we compare the fairness, in the same network with congestion, by inspecting the throughput, latency and packet drop rate on the following TCP variants pairs: Reno/Reno, NewReno/Reno, Vegas/Vegas and NewReno/Vegas. For this experiment, a TCP flow is set from N1 to N4, another TCP flow is set from N5 to N6 and a varying Constant Bit Rate (CBR) flow is set from N2 to N3. The fairness between different variants is analyzed by plotting the graphs for each flow with respect to CBR flow rate.

Experiment 3

Queuing disciplines like DropTail and Random Early Drop (RED) are algorithms that control how packets in a queue are treated. In this experiment we study the influence of the queuing disciplines, DropTail and RED, used by nodes on the overall throughput and latency of flows for the TCP variants TCP Reno and TCP SACK.

3. Experiment 1: TCP Performance under congestion

The network topology is as given in Figure 1. A TCP flow is established from N1 to N4 for the TCP variants Tahoe, Reno, New Reno and Vegas. A CBR flow is introduced from N2 to N3, which is the only varying parameter of this experiment as it increases from 1Mbps to 10Mbps gradually by 0.1Mbps. The simulation is carried out for each variant against the increasing CBR. With the increase in CBR flow, the performance of TCP decreases because TCP is a reliable protocol and it will only send the next window of packets only if the acknowledgment for previous packets are received. Whereas CBR does not rely on any acknowledgments and it keeps on sending the packets to the network and create network congestion. Hence, the throughput of any network should decrease with an increase in CBR flow. This can be verified in experiment 1.

A. Throughput

We calculate throughput for the experiment by parsing the trace files from NS-2. The file has information about various events for various flows. Through custom scripts, we parse all TCP events based on the flow and calculate throughput by finding out the total amount of bytes sizes

sent during the time of simulation under different CBR flow rates. The simulation result is as shown in Figure 2.

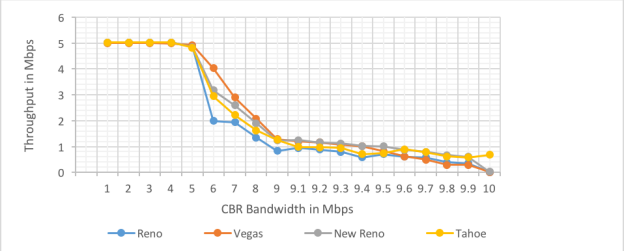

Figure 2: Throughput vs CBR Bandwidth

For CBR < 6Mbps, the throughput of of all the TCP variants is almost the same. However, as the CBR flow increase we see a varying decrease in the throughput for different TCP variants. This is because once the CBR flow increases, the network starts getting congested and each variant applies there congestion avoidance mechanisms as explained. Vegas performs better than the other three variants, Reno having the lowest throughput among all. Since Vegas detects congestion at an early stage, its performance is significantly better compared to others.

B. Latency

The average latency of each variant is calculated based on the actual RTT of the packets. We parsed the trace file generated to keep track of the send and ACK received time of a particular packet based on the sequence number. Figure 3 shows the average latency of the different TCP variants.

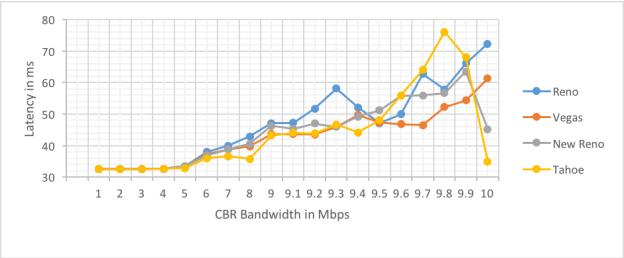

Figure 3: Latency vs CBR Bandwidth

Till CBR ≤ 6Mbps, all TCP variants have almost the same latency. Tahoe has a sharp increase in the latency after this point because Tahoe uses Go Back N. This causes the retransmission rate to increase due to the packet drops which causes high queuing delays. Since Vegas does a pre-detection, it queues packets when the received RTT is greater than the base RTT. Hence, Vegas has the least and Tahoe has the highest latency.

C. Packet Drop Rate

The packet drop rate is calculated by the difference between the number of packets that was sent and the number of acknowledgement received of the sent packets. The packet drops for different TCP variants against the different varying CBR is shown in Figure 4.

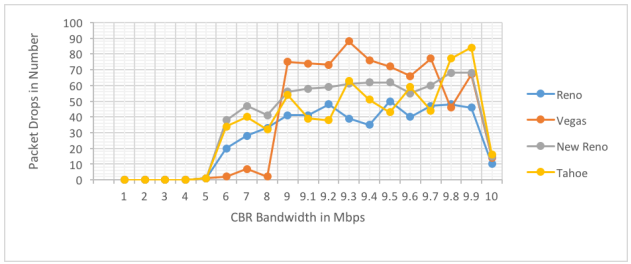

Figure 4: Packet Drop Rate vs CBR Bandwidth

For CBR ≤ 5Mbps, we see little to no packet drops in the network. As the CBR increases, there is a sharp increase in the packet drops for Vegas when compared with others. This happens because once the extended queue is filled, the arrived packets are dropped, and the window reduces drastically to half its current value. The concept holds good for the other three as well and they almost show the same characteristics.

Based on the above observation, Vegas performed better than the other three variants. Though this is true for this experiment, but we cannot assertively say that Vegas is the best among all the others as this was an analysis done on a simple network. We may need to expand and make our topology a little more complex and perform the same experiment again.

4. Experiment 2: Fairness between TCP Variants

We have a similar topology like Experiment 1 for this experiment as well apart from one addition where we add another TCP flow from node N5 to N6. We calculate the throughput, latency and packet drop for each pair of TCP flows in the network. Experiment similar to Experiment 1 was conducted and the results were obtained. For the rest of experiment 2 we will refer to the TCP flow from N1 to N4 as TCPFlow1, the TCP flow from N5 to N6 as TCPFlow2 and the CBR flow from N2 to N3 as CBR flow.

A. Reno/Reno

Both the TCPFlow1 and TCPFlow2 are assigned to TCP Reno. The results are shown in Figure 5 for Throughput, Figure 6 for Latency and Figure 7 for Packet Drop rate against varying values of CBR bandwidth.

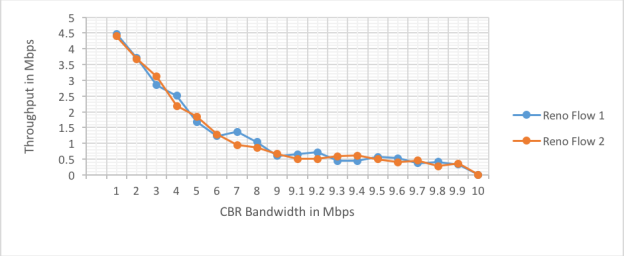

Figure 5: Reno/Reno - Throughput vs CBR Bandwidth

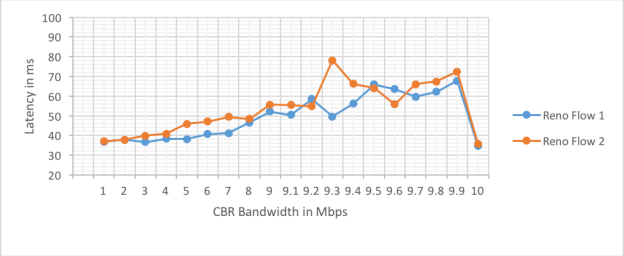

Figure 6: Reno/Reno - Latency vs CBR Bandwidth

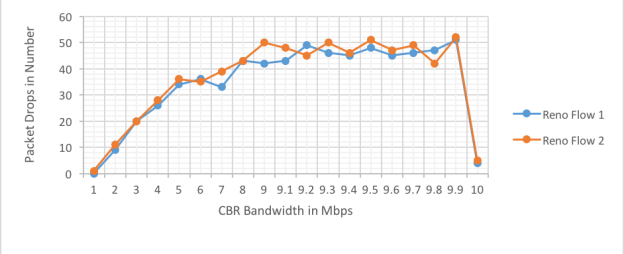

Figure 7: Reno/Reno - Packet Drop Rate vs CBR Bandwidth

Even though there is a slight wavering in throughput and latency, both the flows remain close and on similar tracks. This wavering is justified as both the variants are trying to synchronise their flows by alternatively utilizing the bandwidth. We conclude that two TCP Reno flows are fair to one another in the same system.

B. NewReno/Reno

TCPFlow1 is assigned TCP New Reno and TCPFlow2 is assigned TCP Reno. The results are shown in Figure 8 for Throughput, Figure 9 for Latency and Figure 10 for Packet Drop rate against varying values of CBR bandwidth.

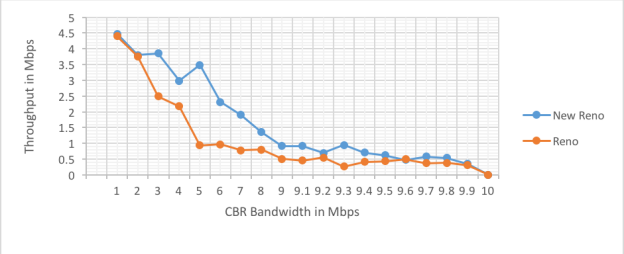

Figure 8: New Reno/Reno - Throughput vs CBR Bandwidth

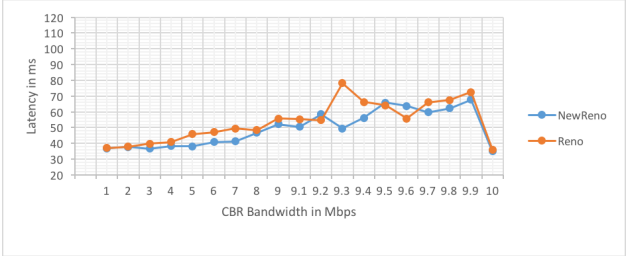

Figure 9: New Reno/Reno - Latency vs CBR Bandwidth

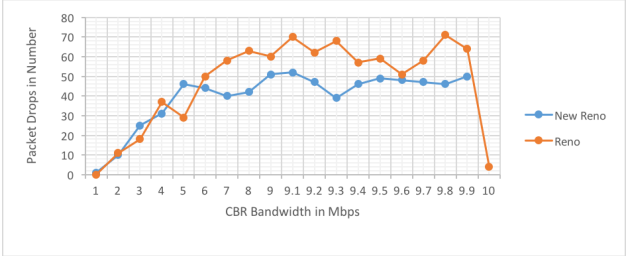

Figure 10: New Reno/Reno - Packet Drop Rate vs CBR Bandwidth

TCP Reno has lower throughput than NewReno when CBR≥ 2Mbps. Reno drops more packets and has a higher increment in latency when CBR comes to close bandwidth limitation. This is because it handles a single packet loss scenario and exits and enters the Fast Recovery stage per loss. Whereas NewReno handle multiple packet drops but it does not comes out of fast recovery unless all previous packets are acknowledged. Therefore, NewReno is not fair to TCP Reno; also it does not overcome Reno's performance completely.

C. New Reno/Vegas

TCPFlow1 is assigned TCP New Reno and TCPFlow2 is assigned TCP Vegas. The results are shown in Figure 11 for Throughput, Figure 12 for Latency and Figure 13 for Packet Drop rate against varying values of CBR bandwidth.

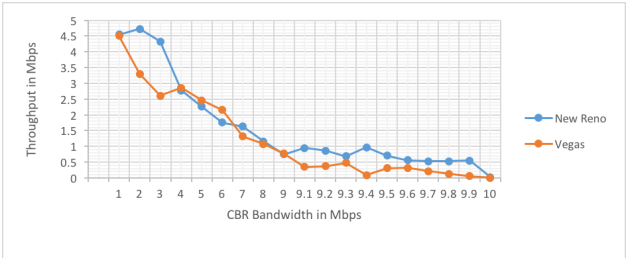

Figure 11: New Reno/Vegas - Throughput vs CBR Bandwidth

As the CBR flow increases, the throughput of TCP NewReno becomes higher than the throughput of TCP Vegas. Also, it runs more steadily and loses less number of packets than Vegas. For latency, they remain close for a

particular period, but when CBR reaches 9.4Mbps, it suddenly goes to a higher latency than NewReno.

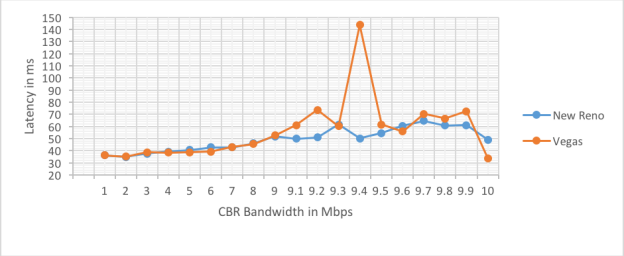

Figure 12: New Reno/Vegas - Latency vs CBR Bandwidth

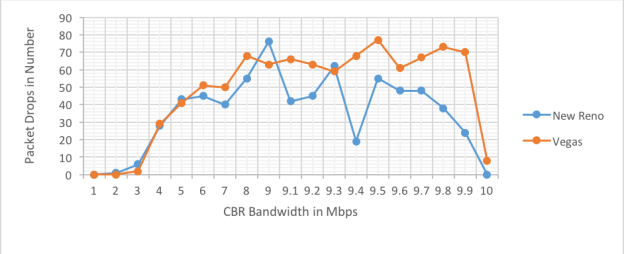

Figure 13: New Reno/Vegas - Packet Drop Rate vs CBR Bandwidth

So we confirm that NewReno is unfair to Vegas since Vegas recognizes congestion at an early stage and it reduces its sending rate which allows it to provide more bandwidth to NewReno and hence, NewReno becomes dominant.

D. Vegas/Vegas

Both the TCP Flow 1 and TCP Flow2 are assigned to TCP Vegas. The results are shown in Figure 14 for Throughput, Figure 15 for Latency and Figure 16 for Packet Drop rate against varying values of CBR bandwidth.

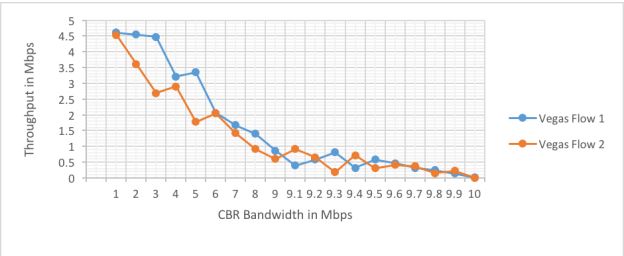

Figure 14: Vegas/Vegas - Throughput vs CBR Bandwidth

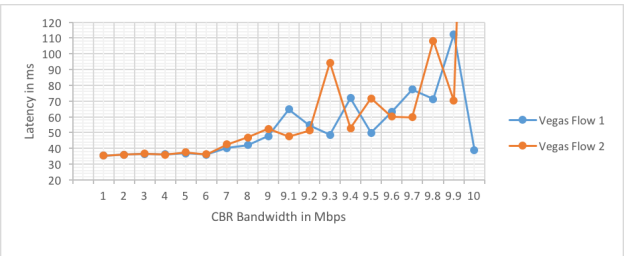

Figure 15: Vegas/Vegas - Latency vs CBR Bandwidth

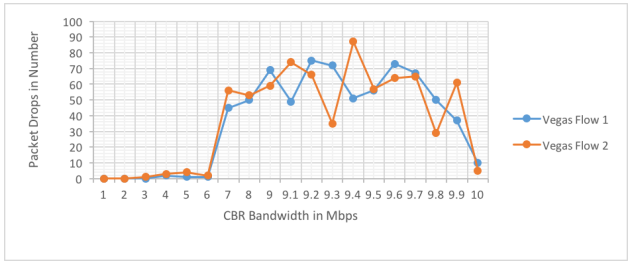

Figure 16: Vegas/Vegas - Packet Drop Rate vs CBR Bandwidth

The observations of two simultaneous TCP Vegas flows is similar to that of two simultaneous TCP Reno flows as we had seen above. We confirmed that wavering is acceptable as both the flows are trying to synchronize their flows and hence it is fair for two Vegas TCP running on the same system.

From the experiment we conducted above, we come to a conclusion that if the same TCP variant is introduced as the second flow, it is usually fair to one another. The same cannot be said about different variants of TCP flowing on the same network. Different TCP variants running on the same network are not entirely reasonable with one another. The reason for this is that each variant has its own execution points, transmission rate, congestion avoidance window or retransmission strategy, which is different for different variants. For instance, Vegas will always consider the network to be congested and it will diminish its sending rate, thus never gets predominant back again.

5. Experiment 3: Influence of Queuing

The network topology for this experiment is the same as above, but the flows change. A TCP flow is present from N1 to N4 and a CBR flow is present from N5 to N6.

Firstly, we plotted a throughput vs time graph to see, when TCP flow was stabilizing. The Flow stabilized around the 11-12 second mark. Since CBR rate was supposed to be a constant, we set a constant value of 10 Mbps. We introduced a CBR flow at 12s and stopped it at 30s to understand the behaviour of TCP under its influence. The bandwidth of every link was set to 10Mbps with a delay of 10ms, TCP window was set to 150, TCP Maximum Congestion Window was set to 200. All other values were default values. We recreate four sets of TCP variations and queue types: Reno with DropTail, Reno with RED, SACK with DropTail, and SACK with RED. The assessment procedure depends on the average bandwidth and average latency of TCP stream.

A. Average Throughput

The graph below shows the average bandwidth of TCP Reno and SACK, both using RED and DropTail queuing algorithm each against time.

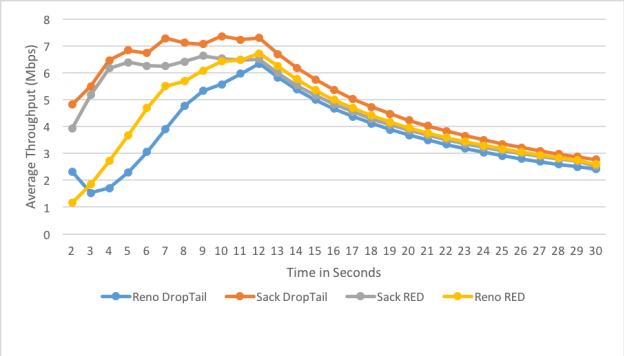

Figure 17: Throughput vs Time

The graphs contradict our understanding that RED algorithms perform better than Droptail algorithms. Initially we get high throughputs till CBR is introduced. Once CBR flow is introduced, the throughput for TCP flows gets reduced and stabilizes after a time period.We don't clearly see a winner from the graphs, although one of the DropTail algorithms perform better here. The RED algorithms perform consistently and give high throughput values. RED algorithms monitor average queue sizes and perform early drops if necessary, whereas DropTail algorithms simply drop packets if its buffer gets full. So, it is expected that RED behave better than DropTail in network scenarios when congestion is likely. This is also a reason why RED is fairer than DropTail algorithms as it is not biased against bursty traffic, whereas DropTail simply floods the buffer even with bursty traffic.

B. Average Latency

The graph below shows the average latency of TCP Reno and SACK, both using RED and DropTail queuing algorithm each against time.

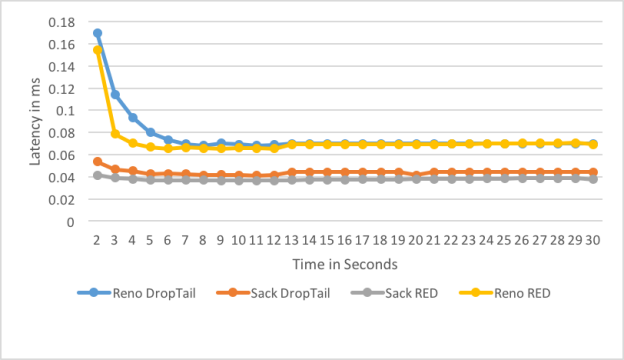

Figure 18: Latency vs Time

As we observe from the graph in Figure 18, the SACK with RED and SACK with DropTail have the lowest latency. The Reno algorithms show a significant increase in latency initially and then stabilize. The combination of RED and SACK give low delay and appear to be a good choice. Also DropTail and SACK perform good. It seems both algorithms work well with SACK.

6. Conclusion

In this paper we discussed about the various ways of analyzing a network using the NS-2 simulator. We mapped different performance parameters like throughput, latency and packet drop rate against the varying parameters for each experiment.

Following are the observations:

1) We can understand from the results of experiment 1 that the TCP Variant, VEGAS, performs better in a congested networking giving higher values of throughput, low values of delay and high values of packet drops.
2) From Experiment 2, we can conclude that same variants are fairer to each other and other combinations are not fairer. E.g Reno/Reno is fair to each other whereas NewReno/Vegas are not fair to each other
3) A TCP SACK variant while implementing the RED and DropTail queuing algorithms has the best throughput and the least latency compared to others. Also RED algorithms are fairer than DropTail algorithms.

These results are solely based on the experiments conducted above on the network topology mentioned under certain conditions. Each TCP variant has its own way of handling congestion and have their own merits and demerits. Choice of a variant depends solely on the situation in hand. Some variants work well in some scenarios, some in others. For a concrete analysis we will have to perform more experiments under more complex networks.

References

[1] Simulation-based Comparisons of Tahoe, Reno, and SACK TCP by Kevin Fall and Sally Floyd

[2] An Overview of Performance Comparison of Different TCP Variants in IP and MPLS Networks *http://link.springer.com/chapter/10.1007%2F978-3-642-22185-9_11#page-1*

[3] Wikipedia, TCP congestion-avoidance algorithm, *https://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm*

[4]*http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.258.6711&rep=rep1&type=pdf*

[5] *https://en.wikipedia.org/wiki/Random_early_detection*