Performance Analysis of TCP Variants

Sarvesh Kapre
kapre.s@husky.neu.edu
Northeastern University, Boston, MA, USA.

Manu Saxena
saxena.m@husky.neu.edu
Northeastern University, Boston, MA, USA.

Abstract

This paper analyzes the performance of various TCP variants, Tahoe, Reno, NewReno, Vegas and SACK using simulations, under different conditions by varying parameters such as; CBR flow, time, queuing algorithms, DropTail and RED (Random Early Drop). The results obtained are used to analyze the throughput, drop rate, latency and fairness between these variants. In general, we will discuss our findings from the three experiments we performed. In Experiment 1, we compared the throughput, drop rate and latency for Tahoe, Reno, New Reno and Vegas. By varying CBR flow in our simulation implementation we have showed, why Vegas is the best amongst the rest. In the second experiment we have compared the fairness between popular TCP variants like, Reno-Reno, Vegas-Vegas, NewReno-Reno and NewReno-Vegas. In experiment 3, we have tested the influence of queuing algorithms, RED and DropTail on TCP Variant Reno and SACK.

1. Introduction

TCP is one of the most popular and necessary protocol used in day to day lives. Scalability of networks gave rise to complexity and the congestion due to ever increasing traffic. In order to tackle the problem due to congestion and increase the productivity of protocol, many TCP variants have evolved since the original version. Thus it is crucial to take into consideration the different variants, the reasons for their evolvement and the quality of service they provide. This paper also elaborates on the findings from our experiments, where we tested the properties of the TCP protocols to its limit and ultimately determined the performance of each TCP variant. Furthermore, we have also discussed the properties of each protocol and the reason for their uniqueness as compared to each other. Properties of few of these TCP variants is given below:

1.1 Tahoe

Tahoe uses a very basic approach when injecting packets. It listens for acknowledgments of the transferred packet and only then injects a packet in the network. Features of Tahoe are –

a. Slow Start algorithm
b. Congestion avoidance
c. Fast Retransmit

To avoid overwhelming the network at the first instance, due to lack of information about the bandwidth, Tahoe implements slow start algorithm. The congestion window (CWD) size is set to 1 and for every acknowledgment received the CWD size increases by 1. The number of packets transferred are increased exponentially until it detects a packet loss due to congestion. Tahoe detect packet loss by implementing coarse grained timeouts. When a congestion is detected the CWD size is set to one and the sending rate is decreased. Tahoe uses "Additive increase and multiplicative decrease" to avoid congestion. Thus, when a packet loss occurs, it is considered as a sign of congestion and the threshold of CWD is set to half and its size is set to one. Problems with Tahoe is that, it takes longer to detect packet loss as it follows coarse grained timeouts and empties the pipeline resulting in loss of bandwidth [1] [2].

1.2 Reno

Reno retains all the features of Tahoe and adds one feature – Fast Recovery. It does not wait for complete timeout in order to detect packet loss. Instead, 3 duplicate ACKs are considered as a sign of packet loss and it immediately retransmits the packet. It avoids emptying he pipeline thus saving the bandwidth. Problem with Reno occurs, when multiple packets are lost in the same window. Its performance in this case is almost same as Tahoe as the second packet loss will only be detected after the ACK for first retransmitted segment is received after one round trip time(RTT) [1][3].

1.3 NewReno

NewReno retains all the properties of Reno expect, it modifies its Fast Recovery algorithm and add some intelligence over it, in order to detect multiple packet loss. Unlike Reno, NewReno does not exit Fast Recovery unless it receives acknowledgement for all the packets that were present when it entered Fast Recovery. This enables the detection of multiple packets loss. Problem with NewReno is, it requires one RTT to detect each packet loss [2].

1.4 Vegas

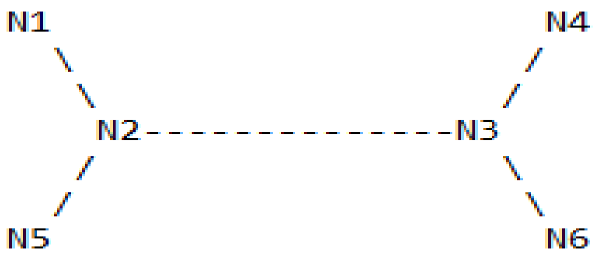
Unlike the above variants, that uses a reactive approach to tackle congestion, Vegas uses a proactive approach. It tries to prevent the congestion by detecting it quite early, which helps it to reduce the flow of packets from overwhelming the network. It does so by checking timeouts on regular intervals. Vegas can be considered as a modification of Reno and Tahoe, it supports their feature as well as adds its intelligence in detecting packet loss [2].

1.5 SACK

TCP with Selective Acknowledgment is modification of Reno and Tahoe. It overcomes the drawback of Reno by detecting and sending multiple packets in one RTT. Unlike cumulative acknowledgments in Reno, it uses selective acknowledgments to detect packet loss. It retains the slow start and fast retransmit features of Reno [2].

2. Methodology

To conduct our experiment, we have used the following network topology.



The bandwidth of each duplex link is set to 10 Mbps and the queue size is kept 10. We use the default packet size of 1000. For every experiment, different parameters are changed and this topology is run in ns-2 simulator. TCL script is written to perform different experiments and a trace file is generated whose results are analyzed using a python script. For the values obtained, we have manually plotted the graphs in excel to understand the performance of various TCP variants under different load condition.

2.1 Experiment 1

In this experiment, we shall analyze the performance measures such as throughput, drop rate and latency for the TCP variants Tahoe, Reno, NewReno and Vegas. TCP variant has its source at N1 and sink at N4. Its performance is analyzed in presence of CBR flow whose source is at N2 and a sink at N3. CBR flow can be considered as an UDP flow which does not perform flow control and congestion control. The queuing algorithm used is DropTail (works on FIFO - First in First out, rule). The following parameters are used to test the performance of TCP variants.

- Incremented the CBR between 1 to 10 Mbps.
- Packet size of 1000 is kept constant in this case.
- CBR flow is started before the TCP stream and vice versa, to validate the change in results.

We then plot, throughput, packet drop rate, and latency (three of them on Y axis) as a function of CBR bandwidth (on X axis), we will note the properties associated with each TCP variant that could have affected the results and conclude which TCP variant is better with respect to another, by performing T-tests (standard deviation).

2.2 Experiment 2

In this experiment, we shall compare the fairness between TCP variants. We conduct this experiment by using one CBR, and two TCP flows. The CBR source is at N2 and a sink at N3 and two TCP streams from N1 to N4 and N5 to N6, respectively. The following parameters are used to test the performance of TCP variants.

- Incremented the CBR between 1 to 10 Mbps.
- Packet size of 1000 is kept constant in this case.
- CBR flow is started before both the TCP streams and vice versa, to validate the change in results.

We then plot average throughput, packet loss rate, and latency on Y axis vs CBR bandwidth on X axis and also analyzed how these protocols are implemented and how the different choices in different TCP variants can impact fairness. We conclude the fairness between different TCP variants by performing T-tests.

- Reno/Reno
- NewReno/Reno
- Vegas/Vegas
- NewReno/Vegas

2.3 Experiment 3

In this experiment, we shall add one TCP flow between N1-N4 and one CBR/UDP between N5-N6 and then test the influence of each queuing discipline (DropTail and Random Early Drop-RED) for each TCP stream variant (TCP Reno and SACK) and CBR flow. The following parameters are used to test the performance of TCP variants.

- The size of queue is set to 10.
- Packet size of 1000 is kept constant in this case.
- CBR flow is started before the TCP stream and vice versa, to validate change in the results

Then we plot TCP and CBR flow's average throughput, packet loss rate, and latency on y axis vs time on X axis. We analyze how queuing disciplines influence the bandwidth provided to each flow, by performing T-tests.

3. Results for Experiment 1

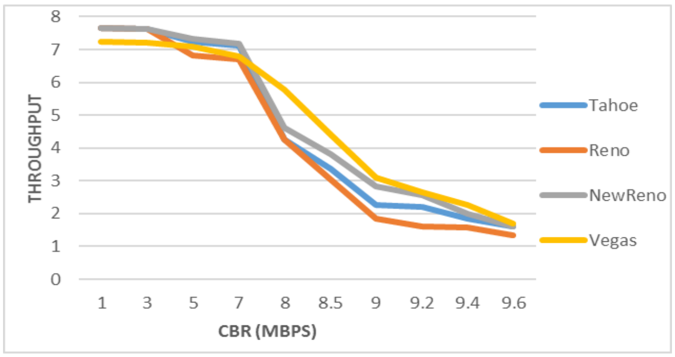


Fig. 3.1 – Throughput for TCP variants over CBR

In the above graph, the throughput of Vegas is less as compared to other in the initial phase. This is because, Vegas follows a proactive rather than a reactive approach. It detects congestion by comparing the sending rate and the expected rate. When the sending rate comes closer to expected rate, it senses that congestion is about to reach and hence lower the throughput even before the congestion, thus saving the bandwidth. Once congestion occurs at around CBR 7, we can see the performance of other variants dropping suddenly as compared to Vegas. Tahoe reduces the congestion window size to 1 when it detects congestion and hence its throughput is decreased. As there are multiple packet drops the throughput of Reno is decreased as well and its performance is affected the same way as that of Tahoe due to its inability to handle and detect multiple packet drops. NewReno has a better throughput as it does not exit fast recovery until it receives all the acknowledgment for all the packets and hence can handle and detect multiple packet drop efficiently as compared to Reno and Tahoe. Thus Vegas has the best throughput as compared to others.

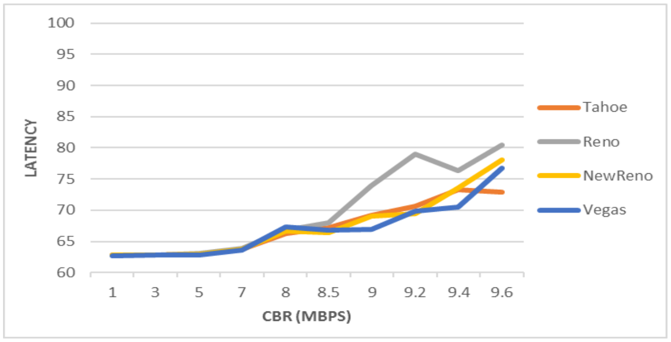


Fig. 3.2 – Latency for TCP variants over CBR

TCP Variants follows a slow start approach to avoid overwhelming the network at the first instance due to lack of knowledge of the bandwidth. As a result of less packets in the network the latency is very low initially. As Vegas senses the nearing of congestion it reduces the packets and hence the latency of Vegas can be seen greater just before the congestion begins at around CBR 8 Mbps. Eventually, due to its proactive approach its latency never peaks suddenly. Tahoe detects packet loss using coarse grained timeouts and hence detects it very late meanwhile continuing to send packet at the same rate. TCP Reno suffers from multiple packet loss in the same window and this can be seen by its exponential rise in latency immediately after congestion. The problem with NewReno is it detects packet loss only after 1 RTT and hence the latency is high due to delay. The average latency for Vegas is less as compared to others.

T-Test Calculation

In order to determine the stability of one variant over other, T-Test was performed for all combinations of variants. T-Test values of all combinations was calculated. Hence we can conclude that whenever the T-values of TCP Vegas are compared with other TCP variants, it is always greater than zero, thus making it more stable than any other variant. Thus the throughput for Vegas is more stable as compared to any other variant under same conditions. We calculated T- Test values using the Excel T-test command containing 2 sample equal variants and one-tailed distribution. When CBR starts before TCP the T- values for Vegas is as given below –

1. T Test value for Vegas/Tahoe – 0.392
2. T Test value for Vegas/Reno – 0.307
3. T Test value for Vegas/NewReno – 0.464

Vegas gives lowest value for variance and standard deviation and best value of mean for throughput amongst all variants.

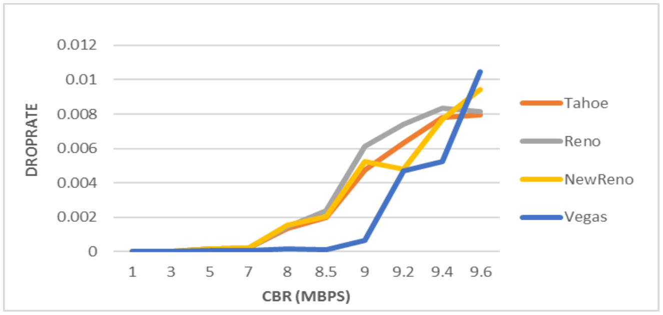


Fig. 3.3 – Drop rate for TCP variants over CBR

In the above graph we can see that the average drop rate is the least for Vegas as compared to others. As Vegas slows down the input rate before congestion and due to queue size being only 10, there is an exponential rise in the drop rate at the end of graph. Tahoe detects packet losses by coarse grained timeouts and hence it may take some time before the packet loss is actually detected. TCP Reno does not wait for complete RTT (Round Trip Time) to detect packet loss. Once it gets 3 duplicate ACK's it moves to Fast Retransmit. The congestion window size of Reno is reduced twice in 1 window. For multiple packet loss Reno and Tahoe performance is almost similar.

Thus from the above three graphs it is evident that TCP Vegas performs better than other TCP variants in all the aspects such as throughput, drop rate and latency.

4. Results for Experiment 2

In this experiment, we analyzed the fairness of the TCP variants to one another. To do so we plotted graphs for throughput, latency and drop-rate of each TCP flow w.r.t CBR flow. When it was not evident from the graph to conclude. we checked for average deviations w.r.t median on each parameter by performing T tests.

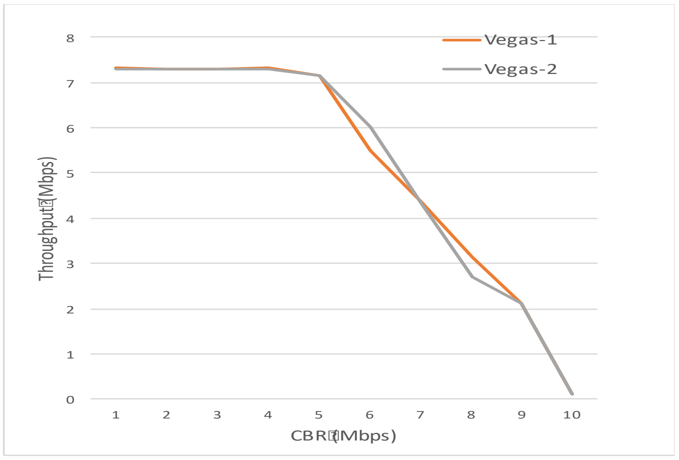


Fig. 4.1.1 Throughput vs. CBR between two TCP Vegas

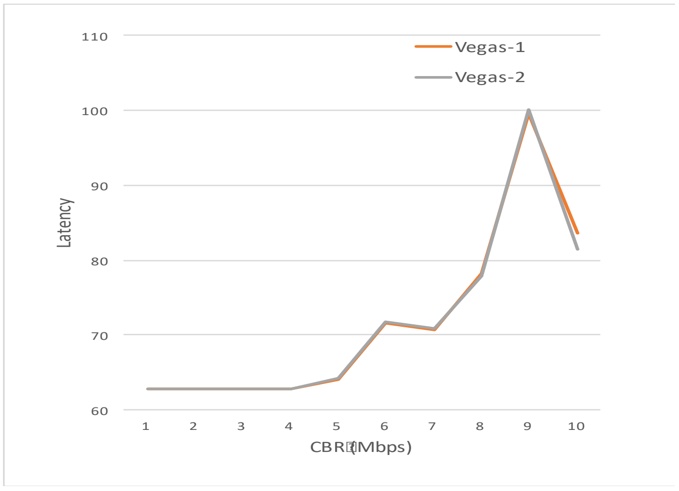


Fig. 4.1.2 Latency vs. CBR between two TCP Vegas

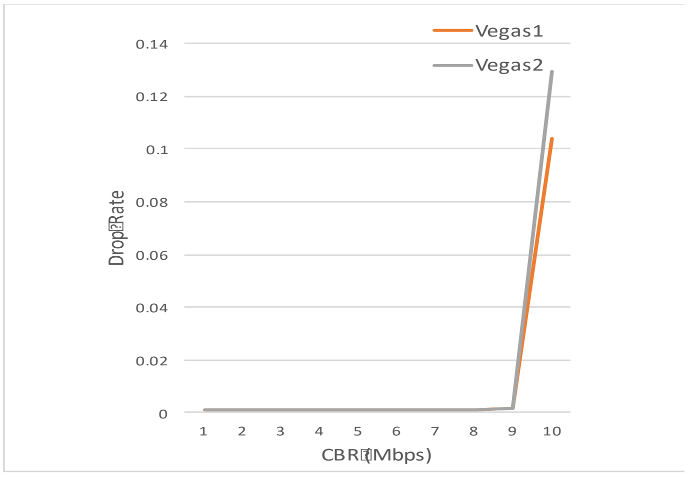


Fig. 4.1.3 Drop Rate vs. CBR between two TCP Vegas

Fig. 4.1.1-3 shows that both Vegas flows are fair to each other. Throughputs of both flows tend to remain same until a congestion is detected. During congestion, one of the flows withdraws to give more bandwidth to other. Hence we see alternate drop and rise in throughput for both flows.

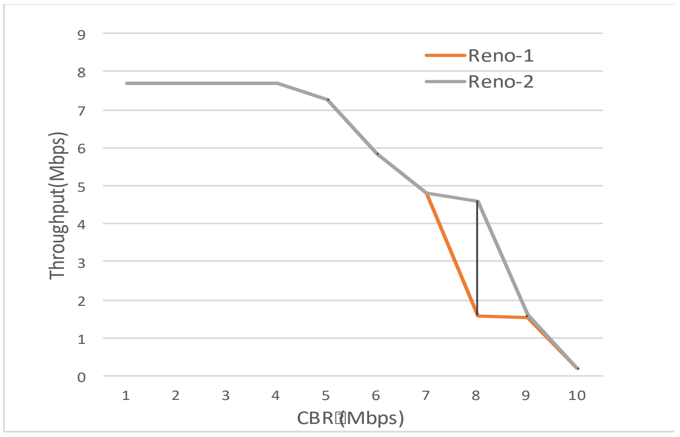


Fig. 4.2.1 Throughput vs. CBR between two TCP Reno

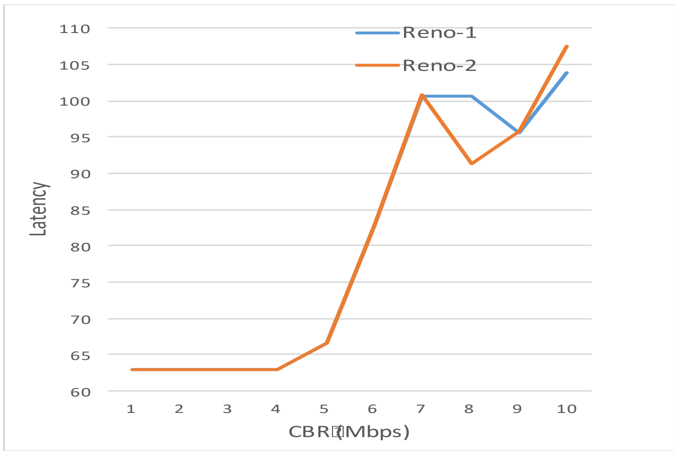


Fig. 4.2.2 Latency vs. CBR between two TCP Reno

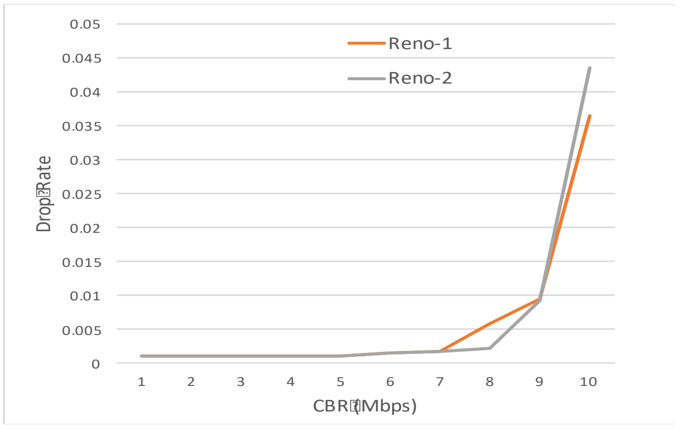


Fig. 4.2.3 Drop Rate vs. CBR between two TCP Reno

Following similar trend as Vegas, we see Reno flows too remain fair to each other as in Fig. 4.2.1-3.

New Reno however does not come out fair to Reno when network sharing and throughput is concerned. Unlike Reno, New Reno can send new packets which are at the end of the congestion window during fast recovery phase. This way New Reno keeps the transmission window full even after multiple packet drops during congestion and maintains high throughput, as observed in Fig. 4.3.1-3

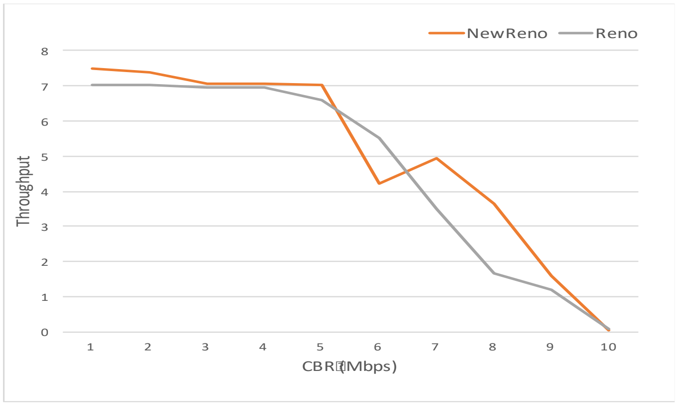


Fig. 4.3.1 Throughput vs. CBR between TCP New Reno and Reno

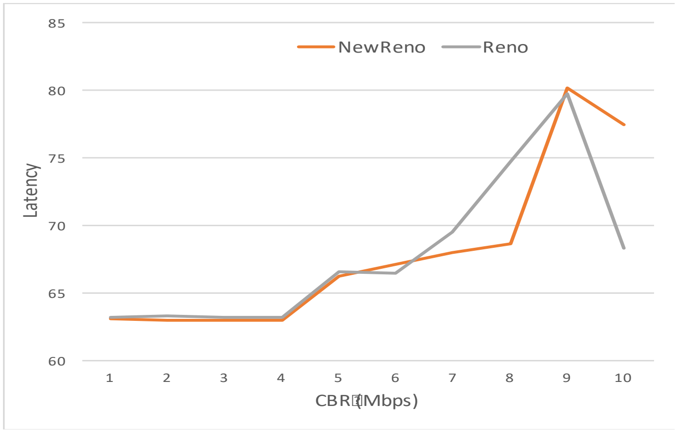


Fig. 4.3.2 Latency vs. CBR between TCP New Reno and Reno

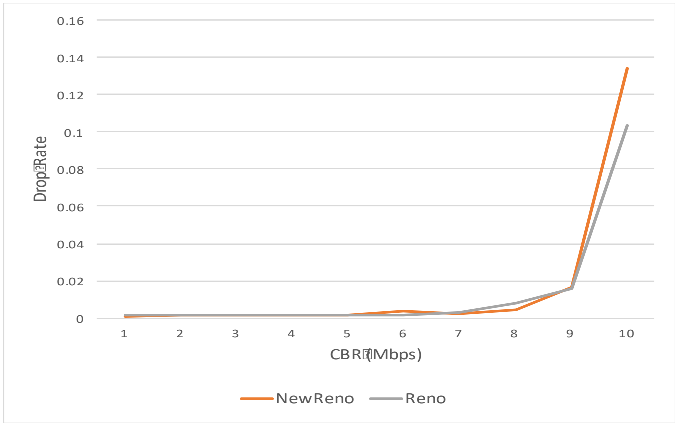


Fig. 4.3.3 Drop Rate vs. CBR between TCP New Reno and Reno

Fig. 4.4.1-3 shows that New Reno is unfair to Vegas as well. Vegas follows a congestion detection mechanism to prevent packet drops. As soon as it detects congestion due to increased traffic by New Reno, it reduces the number of packets it sends in a window. As a result, gives more bandwidth to New Reno. But when CBR traffic increases, packets drop is greater for New Reno than to Vegas.

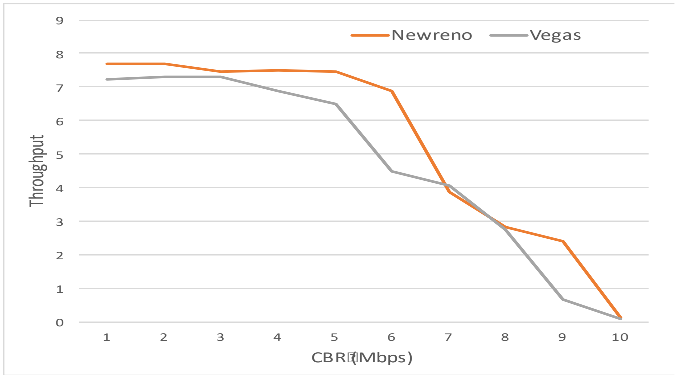


Fig. 4.4.1 Throughput vs. CBR between TCP New Reno and TCP Vegas

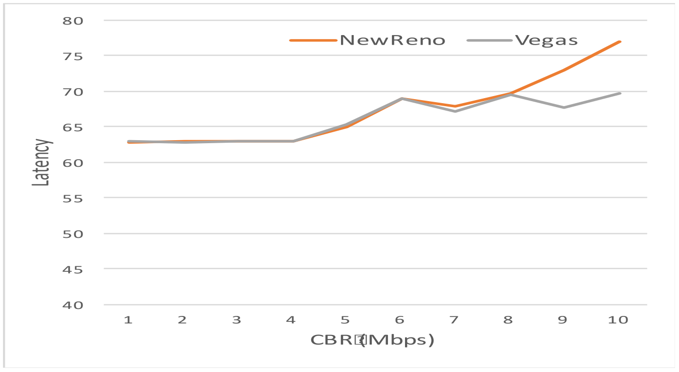


Fig. 4.4.2 Latency vs. CBR between TCP New Reno and TCP Vegas

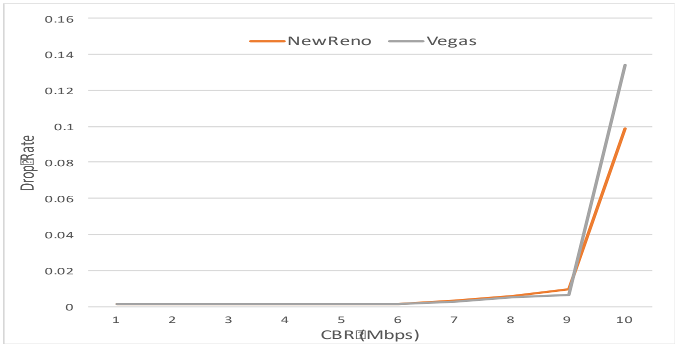
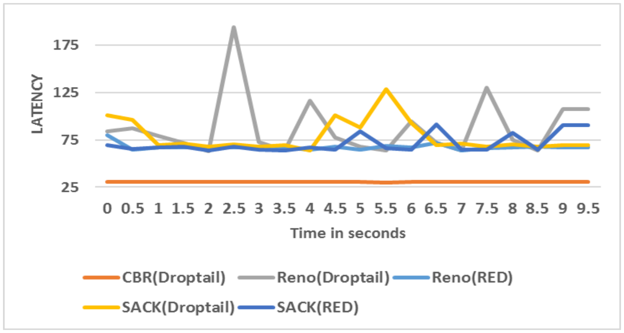


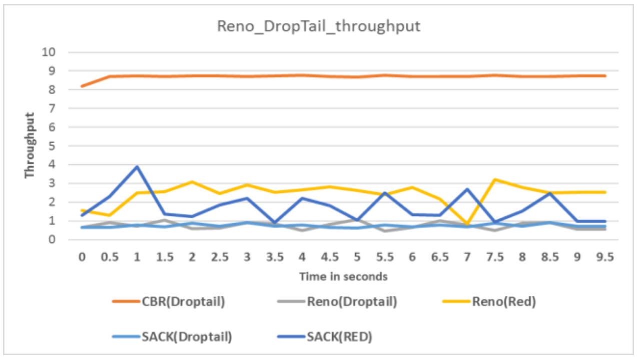
Fig. 4.4.3 Drop Rate vs. CBR between TCP New Reno and TCP Vegas

5. Results for Experiment 3

In this experiment, we shall compare the two queuing algorithms, DropTail and RED. DropTail also called Tail Drop, is a Passive Queuing algorithm which follows FIFO approach to add or drop packets from the router queue. It does not perform packet differentiation neither assigns priorities to any packets. Random Early Drop (RED), adds intelligence in adding or dropping packets. It assigns minimum and maximum threshold values and before accepting a packet it the queue size. If it lies between maximum and minimum threshold, then the packet is dropped or marked with average. If the average queue size is above maximum threshold, all new coming packets are dropped [4].



3.1 Latency for DropTail and RED w.r.t TCP SACK and RENO and CBR over Time



3.2 Graph of Throughput for DropTail and RED w.r.t TCP SACK and RENO and CBR over Time

CBR flow can be considered as UDP, is connectionless and does not care about dropped and out of sequence packets. It is only meant to work faster than TCP where reliability is optional but fast performance is mandatory. Hence, latency of CBR is the lowest as compared to other TCP variants whose latency depends on usage of queuing algorithm. As it is meant to be the quickest, CBR has higher throughput as well.

Looking at the graph we can state that the latency for DropTail is higher than RED. As RED uses some intelligence as compared to DropTail this was bound to happen. For the same TCP variant Reno and SACK, queue algorithm RED is much efficient in handling and dropping packets. SACK is an extension of Reno. SACK retains the slow start and fast retransmit properties of Reno but overcomes the problem of identifying multiple dropped packets and their retransmission in one RTT. Due to this additional improvement over Reno it performs better than it. Reno performs better with single lost packet. Hence with usage of RED with Reno, it performs much better. Average throughput for Reno(RED) is almost equal to that of SACK(RED) as a result of single packet loss rather than multiple packet loss. Results from the graph also states that SACK(DropTail) leads better throughout than Reno(DropTail) which has the least throughput. Hence we can also see that there is a drastic fall in the throughput with Reno(DropTail) due to multiple packets loss rather than Reno(RED). The difference in throughput is relatively less with the usage of SACK with RED and DropTail. From the above discussion and the results obtained in the graph, it is evident that SACK(RED) is the best amongst the rest, followed by Reno(RED), SACK(DropTail) and Reno(DropTail) respectively. Hence the combination of TCP SACK and RED works well together whereas TCP Reno and DropTail does not work well and must be avoided. This also leads us to the conclusion that better queuing algorithm must also be supported by better TCP variant in order to achieve better result and better performance under different load conditions.

6. Conclusion

Thus we have analyzed the performance of different TCP variants and queuing algorithm by performing different sets of experiments. From the above experiments we can conclude that –

1. In terms of different properties of TCP variants like average throughput, lowest average latency, fewest drops, the overall best TCP variant is Vegas.
2. TCP variants Reno-Reno and Tahoe-Tahoe are fairer to each other.
3. RED is a better queuing algorithm than DropTail and efficiently achieves low latency and high throughput.

In the real world, with the ever increasing traffic TCP variant Vegas should be preferred as it has a clear edge over other variants. Queuing algorithm DropTail must be avoided and RED must be used to avoid packet loss. Implementing selective acknowledgment is TCP is a good strategy, but currently is not provided by the receiver. Efforts must be applied in implementing SACK in TCP so that it opens up opportunities for further development.

7. References

[1] TCP Tahoe, Reno, NewReno, and SACK–a Brief Comparison http://www.roman10.net/2011/11/10/tcp-tahoe-reno-newreno-and-sacka-brief-comparison/

[2] A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas -http://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf

[3] TCP Reno and Congestion Management http://intronetworks.cs.luc.edu/current/html/reno.html

[4] Comparison and Analysis of Drop Tail and RED Queuing http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F83ABA3ABAEAB6A0BC4AC18E52E57827?doi=10.1.1.233.2255&rep=rep1&type=pdf