Performance Analysis of TCP Variants

Jiawei Cui*, Joey Huang* and Surbhi Gupta*

Abstract

In this paper we have compared simulated performance of TCP variants Tahoe, Reno, NewReno and Vegas to observe how they differ from each other when placed under similar conditions. Separate hosts connected on the internet may use different operating systems with different TCP variants. Thus, it is important to determine the level of performance provided by each TCP variant in different network environments. In order to compare TCP variants we have placed these variants in similar conditions in different networks using NS2 simulations. We also place the variants in the same network to check if there is fairness between them in terms of bandwidth. We also observe the influence of two different queueing algorithms on the performance of TCP variants.

1 Introduction

This paper compares the performance of various TCP variants under various conditions. Firstly, we compare the performance of TCP variants Tahoe, Reno, NewReno and Vegas under congestion. Secondly, we analyze fairness between different variants of TCP in a network. In other words, we check if some variants of TCP are using more bandwidth than others. Lastly, we analyze the influence of different queueing algorithms on the performance of different variants of TCP. On the internet millions of hosts communicate using different TCP variants. All these hosts have different requirements in terms of performance depending on the conditions they deal with. Thus, analysis of different TCP variants may give us insights into how using a different TCP variant may help achieve the desired level of performance.

The results of our experiments show that Vegas has the highest throughput, least latency and least packet drop rate in the presence of congestion in the network. Vegas adopts the best strategy when dealing with these conditions. NewReno performs better than Reno because it adopts an improved version of Fast Retransmit strategy. Tahoe does not implement Fast retransmit and Fast recovery, thus it performs badly in comparison to other variants. We expect all TCP variants to be fair with each other, but in practice, most variants are not. The only exception is Reno, which was fair when paired with itself.

When we analyze the influence of two different queuing algorithms on the performance of TCP, the results show that both SACK and Reno provide better throughput under DropTail but provide better latency under RED algorithm.

2 Methodology

The results in this paper are based on simulation. We use the NS2 Network Simulator to set up the topology required for the experiments and observe the behavior of the different TCP variants. NS2 supports TCP variants including Tahoe, Reno, NewReno and Vegas etc., and provides the tools to simulate conditions similar to those experienced in a real-world network. NS2 also supports various transport layer protocols like UDP and TCP, and also applications like FTP and CBR. NS2 produces trace files which record every packet that passes through the network.

Although NS2 is a powerful tool used for network simulation, it lacks in providing a way to analyze the results produced. We use Perl scripts to analyze the results produced by the simulation. We calculate throughput, latency and packet drop rate under various conditions.

We also take advantage of Bash scripts to automate the process of changing various simulation parameters to set up different environments under which TCP performance is analyzed.

We performed three experiments to compare performance of TCP variants:

Experiment 1

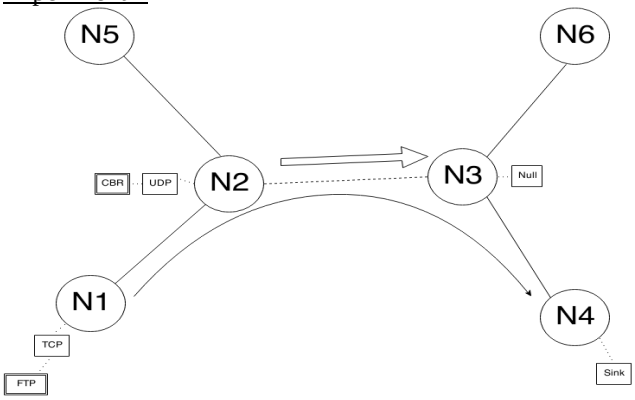

Fig 2.1

In the first experiment we measure TCP performance under congestion. Fig 2.1 shows the topology used for this experiment. The nodes in the topology are linked by duplex links of bandwidth of 10Mbps, delay of 10ms and Drop Tail queue. We set up CBR (Constant Bit Rate) as the traffic source at node N2 with N3 as the null. CBR sends traffic at a constant rate without taking congestion control into account. We vary the rate of CBR flow to observe TCP behavior under various load conditions. The rates at which we observe TCP performance are 1 through 10 Mbps. We set up FTP over TCP

at N1 as the traffic source and a single TCP stream from N1 to the TCP sink at N4. We start CBR flow 10 seconds after we start TCP flow so that once TCP flow stabilizes we are able to observe how CBR flow makes an impact on the TCP connection. We perform this experiment with four TCP variants: Tahoe, Reno, NewReno and Vegas. In order to compare the performance of various TCP variants we calculate three properties: throughput, latency and packet drop rate.

The formulae we use to compute throughput, latency and drop rate are:

Throughput (Mbps):

$$= \frac{P_c * P_s * 8\frac{bits}{byte}}{\frac{1000000bits}{Mbits} * (t_L - t_F)}$$

P_c : packet count P_s : packet size (bytes) t_L : timestamp that the last TCP packet was received by the receiver (seconds)
t_F : timestamp that the first TCP packet was received by the receiver (seconds)

Latency (seconds):

$$L = t_R - t_S$$

t_R : timestamp that a certain packet was received (seconds)
t_S : timestamp that the packet with the matching packet ID was sent (seconds)

Drop rate (%):

$$D = \frac{P_D}{P_S}$$

P_D : number of TCP packets dropped
P_S : total number of TCP packets sent by the source

Experiment 2

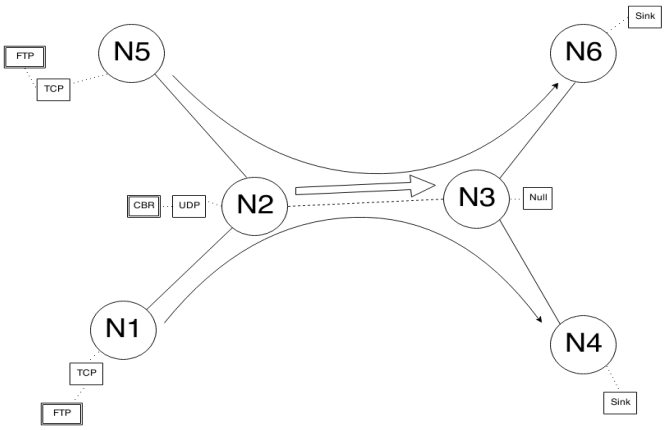

Fig 2.2

Fig 2.2 shows the topology used for this experiment. We compare fairness between different pairs of TCP variants. For this experiment we use the same topology as the one used in first experiment, but with two TCP streams instead of one. We set up CBR at N2 with a null at N3. We set up two TCP streams from nodes N1 to N4 and N5 to N6 respectively. In order to compare the fairness between various variants we set up TCP variants in pairs one at each TCP streams. Then we vary the CBR flow as in first experiment and calculate throughput, latency and packet drop rate for each of the variant.

Experiment 3

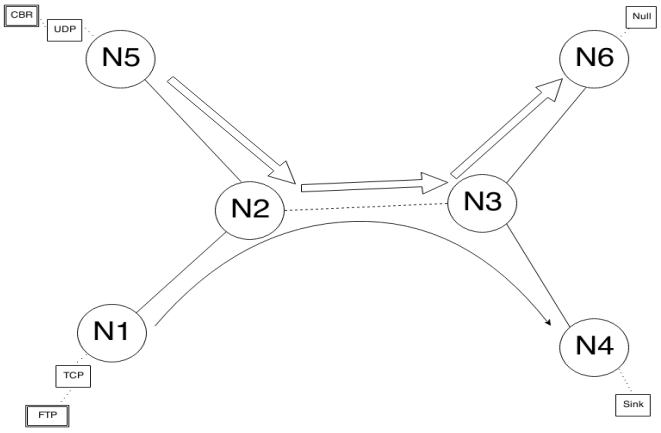

Fig 2.3

Fig 2.3 shows the topology used for this experiment. We observe the influence of two different queueing algorithms on two TCP variants: Reno and SACK. For this experiment we use the same topology as in the first experiment. However, we set up a TCP stream from N1 to N4 and CBR over UDP at N5 with the null at N6. First we start the TCP flow, and once it achieves a steady state then we start CBR flow. In this experiment we do not vary the rate of CBR flow as in the first and second experiments. We observe the behavior of Reno and SACK once with DropTail and once with RED (Random Early Drop) queueing algorithm.

3 Results

Experiment 1:

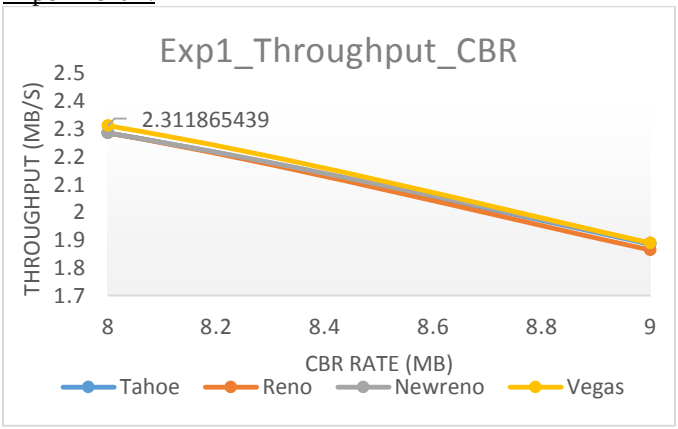

Fig 3.1

Fig 3.1 shows the results of the experiment we carry on to compare the throughput of the TCP variants under congestion.
The graph is plotted between throughput and CBR rate.

The results show that Vegas has the highest throughput under congestion when compared to other TCP variants. Throughput of Vegas decreases with the increasing CBR rate but was higher than that of Tahoe, Reno and NewReno. At 8Mbps CBR, Vegas has the highest throughput compared to other variants. The graph reflects the expected behavior of Vegas. Vegas does not depend on loss of packets to detect congestion as TCP Reno and NewReno does. Rather, Vegas calculates the RTT and sets a timeout for every segment. When it receives a duplicate ACK it checks if the timer has expired, and if so, it retransmits the packet. The throughput of Vegas depends on the accuracy of the calculation of the initial RTT which is BASE RTT. The BASE RTT should not be very small otherwise the throughput decreases. BASE RTT should not be very large otherwise the network will be overwhelmed. Thus, Vegas is expected to have higher throughput that the other TCP variants.

Other TCP variants seem to have almost the same throughput over the range of CBR flows, with Reno performing slightly better than Tahoe and NewReno performing better than Reno. This result can be attributed to the fact that Reno includes all the features of Tahoe, in addition to two new features: Fast retransmit and Fast recovery.

Fast retransmit is the technique of retransmission of lost packets in which Reno does not wait for packets to be dropped before retransmitting. Instead it waits for three duplicate ACKs and then retransmits the packet.

Fast recovery is the technique in which the size of congestion window (cwnd) is not decreased to 1. Instead it is reduced to half of threshold after retransmit. Thus, TCP Reno does not go to the slow start phase again.

NewReno further improves the technique of fast retransmit. It does not even wait for three duplicate ACKs. It retransmits the packet after a single duplicate ACK is received.

Latency Analysis

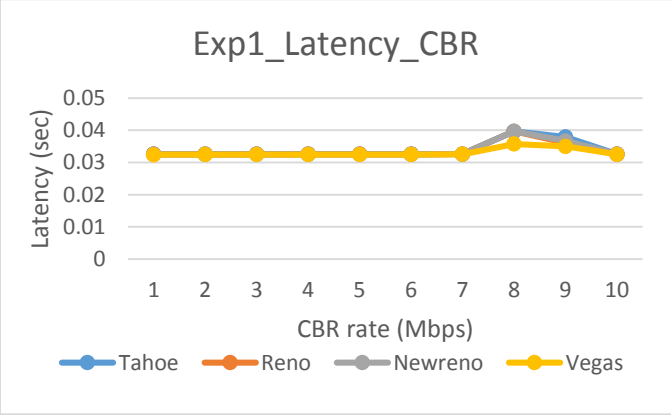

Fig 3.2

Fig 3.2 shows the result of the experiment we carry on to compare the latency, or end-to-end delay, of the TCP variants. The graph shows that when CBR rate is low, all the variants have the same latency. As the CBR rate approaches 10 Mbps, all the variants show different behavior. Vegas has the least latency. Reno and NewReno have almost same latency. Tahoe has the highest latency.

As mentioned above, Vegas calculates RTT for every segment and does not depend on the dropping packets to detect congestion. If the calculated RTT decreases below a certain value, then it increases the size of the window. As the calculated RTT increases for the segments, Vegas becomes aware of the fact that congestion might occur in the network. It gets prepared and adjusts the window size accordingly well before the occurrence of congestion. However, Reno and NewReno depend on the dropping of packets to detect the congestion. Thus, good congestion handling technique is the reason why Vegas has least latency as the CBR rate approaches 10Mbps. Tahoe has the highest latency because it does not implement fast retransmit and fast recovery techniques as Reno and NewReno do.

Packet Drop Rate Analysis

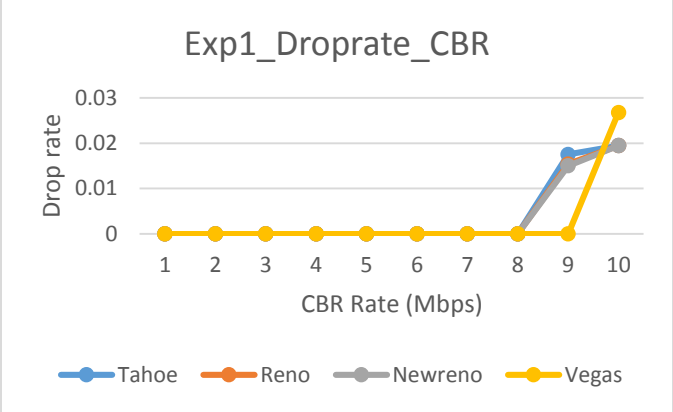

Fig 3.3

Fig3.3 shows the result of the experiment we carry on to compare the packet drop rate of the TCP variants. The graph shows that initially the packet drop rate for all the variants is 0, as no packets were dropped at the low CBR. When CBR increases to 8Mbps, packets start dropping from the queue whose size we have taken equal to 10.

The graph shows that Vegas has the least packet drop rate when the CBR rate is between 8 to 9.5 mbps. This is due to the fact that Vegas does not wait for the packets to drop to realize that congestion might occur in the network. It instead depends on the value of the calculated RTT to detect congestion. It reduces the size of the congestion window as it detects congestion and thus the packets are not dropped.

On the other hand TCP Tahoe, Reno and NewReno wait for the packets to be dropped to detect congestion. Thus, they detect congestion much after Vegas does and thus their

packet drop rate is much higher than that of Vegas at high CBR rates.

According to the graph Reno and NewReno have almost the same packet drop rate. Tahoe has the highest packet drop rate because as mentioned above it does not implement fast retransmit technique.

Experiment 2:

Throughput Analysis

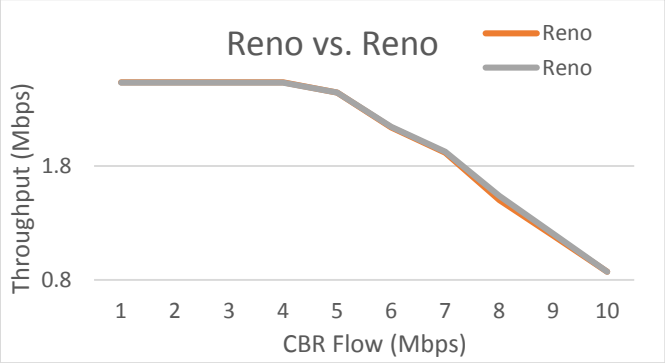

Fig 3.4

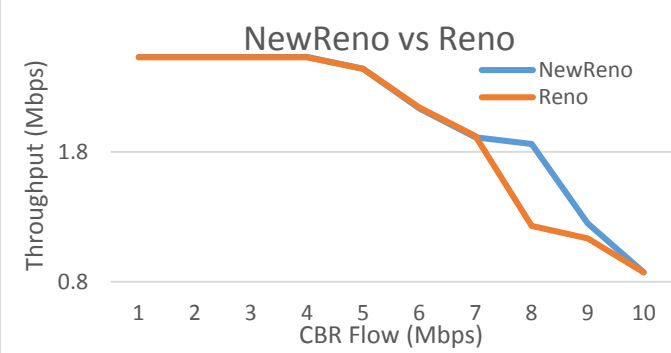

Fig 3.5

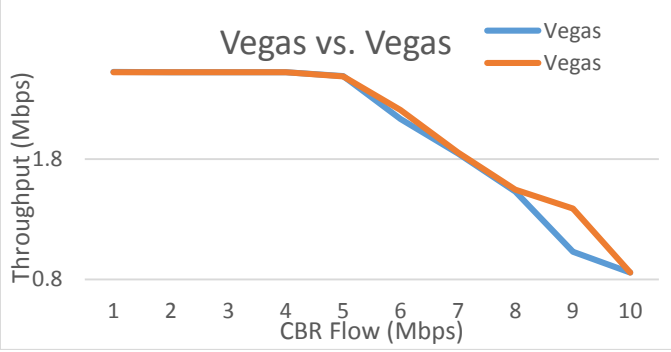

Fig 3.6

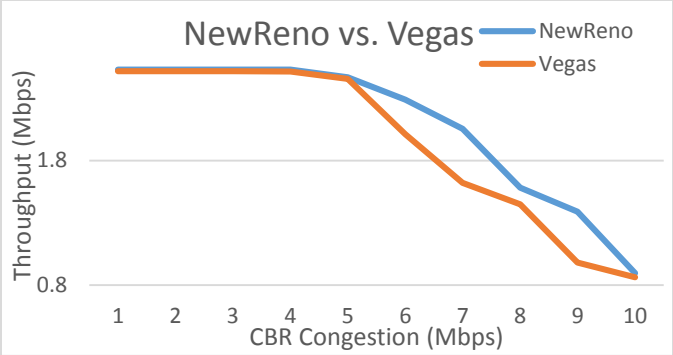

Fig 3.7

In this experiment we set up two TCP streams: one between nodes N1 and N4 and the other between nodes N5 and N6. The aim is to analyze the fairness between various TCP variants. We can say that two TCP variants are fair with each other if they both consume the equal amount of bandwidth. Although various TCP variants should be fair with each other but in practice they are not. We set up different TCP variants at the two streams to observe fairness between them.

Firstly, we set up TCP Reno at both the streams and then repeat the experiment with the pairs: NewReno-Reno, Vegas-Vegas and NewReno-Vegas. Figures 3.4, 3.5, 3.6, 3.7 show the throughput observed in all the four cases.

The first graph shows the throughput when Reno was set up at both the streams. Both the streams with Reno have the same throughput. Thus we can say that Reno is fair with Reno.

The second graph shows the throughput when NewReno was set up at one stream and Reno was set up at the other stream. The graph shows that initially the throughput for both the variants was same but as the CBR rate increases beyond 7 mbps, throughput of NewReno becomes high in comparison to Reno. Thus, it can be inferred that New Reno consumes more bandwidth as compared to Reno.

This is due to fact that as the packets are lost due to congestion in the network Reno adopts the strategy of Fast Retransmit, in which it retransmits the packet after receiving three duplicate ACKs. Reno does not wait for packet drops to retransmit the packet. However, NewReno adopts an improved version of the Fast Retransmit strategy. New Reno retransmits the packet for every ACK received by it. Thus, fast retransmit is faster in NewReno than it is Reno. Thus, in the condition when congestion is about to occur NewReno has higher throughput than Reno and there is no fairness between NewReno and Reno when congestion occurs.

The third graph shows the throughput when Vegas was set up at both the streams. The graph shows that initially both the streams have the same throughput, but as congestion is about to occur one stream occupies more bandwidth than the other.

This is due to the fact that Vegas detects congestion by comparing the sending rate with the expected rate. If the sending rate is higher than the expected rate then it reduces the transmission of packets. However, if the sending rate is less

than the expected rate then that means bandwidth is available to be used and thus it increases the transmission of packets. Thus, at the stage when CBR increases beyond 8Mbps, one of the Vegas streams reduces transmission due to detection of congestion and the other Vegas at the other stream increases transmission of packets due to the bandwidth made available to it by the first Vegas.

Thus one of the Vegas at the first stream has higher throughput than the other Vegas at the second stream.

The fourth graph shows the throughput when NewReno was set up at one of the stream and Vegas was set up at the other stream.

This shows that both the variants have same throughput before CBR reaches 5.2 Mbps. After CBR increases beyond 5.2 Mbps NewReno has much higher throughput than Vegas.

This is due to the fact that Vegas detects congestion much before NewReno does as it does not wait for the packets to be dropped to detect congestion. Thus as Vegas detects that congestion might occur in the network, it reduces the transmission of packets and reduces the size of the congestion window. But at the time when Vegas detects congestion, NewReno is unaware of the possibility of the occurrence of congestion as packets are not dropped till that time.

Thus, Vegas consumes lesser bandwidth than NewReno does and there is no fairness between these variants.

Latency Analysis

Fig 1.8 shows the latency observed in all the four cases.

The first graph shows the latency when Reno was set up at both the streams. Latency for both the TCP streams is same. Thus we can again say that Reno is fair with Reno. That is, both of them use the same amount of bandwidth.

The second graph shows the latency when NewReno was set up at one stream and Reno was set up at the other stream. The graph shows that both NewReno and Reno have same latency up to 7 Mbps CBR but as the CBR increases beyond 7 Mbps the latency of NewReno is higher than the latency of Reno.

The third graph shows the latency when Vegas is set up at both the streams. The latency is same for both the streams up to 8 Mbps CBR. As the CBR increases beyond 8 Mbps one stream shows more than the other.

The fourth graph shows the latency when NewReno was set up at one stream and Vegas was set up at the other stream. The results show that Vegas and NewReno has same latency up to 6 Mbps of CBR. But as the CBR increases beyond 6 Mbps latency of Vegas decreases as compared to NewReno.

Experiment 3:

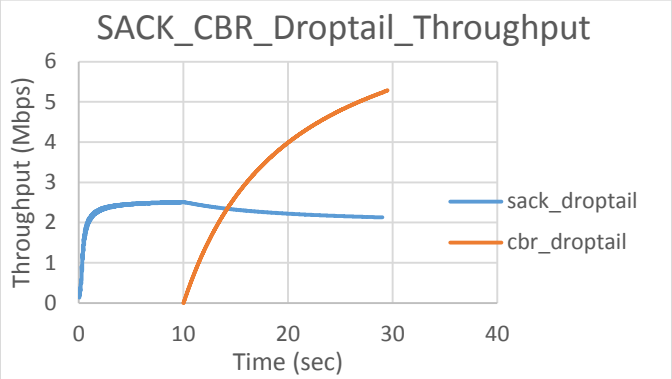

Fig 3.8

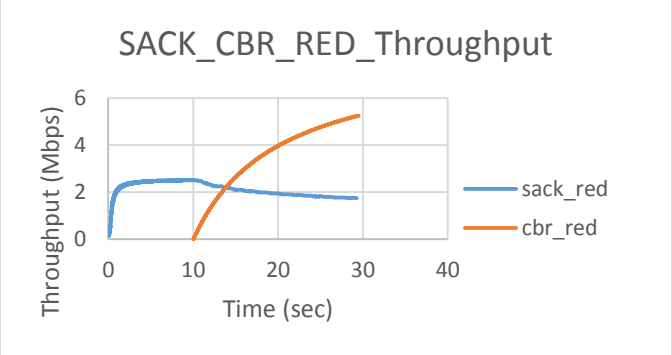

Fig 3.9

In this experiment we analyze the impact of using two different queuing algorithms: Droptail and RED (Random Early Detection) on the performance of TCP variants: Reno and SACK. For this experiment we start TCP flow at 0 seconds and after it stabilized then we start CBR flow at 10 seconds.

Fig. 3.9 and 3.8 show the throughput of SACK and CBR under different queuing algorithms.

The graphs show that there is no effect of queuing algorithms on CBR throughput. But throughput TCP SACK has changed under different queuing algorithms.

The graphs show that up to 10 seconds the throughput of SACK remained unaffected. But as the CBR flow started at the tenth second TCP throughput decreased below 2 Mbps under RED algorithm while under Droptail algorithm TCP throughput always remained above 2 Mbps.

CBR flow quickly increased under both the queuing algorithms but SACK flow first increased and then decreased after 10 seconds when CBR flow was started.

The results were the same when set up Reno instead of SACK.

Thus, it can be inferred that SACK and Reno provide better throughput under Droptail algorithm than under RED algorithm.

These behaviors can be attributed to the properties of the Droptail and RED queueing algorithms. Droptail is the normal behavior of router queues on the internet, and drops all traffic that overflows the queue. RED (Random Early Drop)

drops packets from flows before it reaches its maximum capacity, and therefore allows network links to slow at a steadier rate than allowed by Droptail.

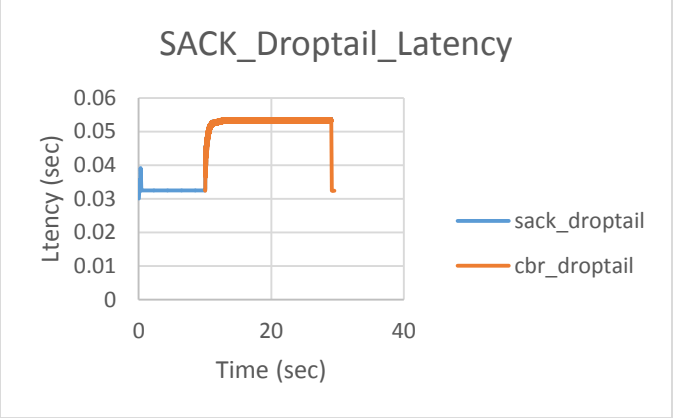

Fig 3.10

Fig 3.11

Fig 3.10 and Fig 3.11 show the latency of SACK and CBR under different queuing algorithms. Under Droptail algorithm, latency of SACK jumped above 0.05 after CBR flow was started at tenth second. On the other hand, under RED algorithm latency of SACK remained below 0.04 after tenth second.

Thus, it can be inferred that SACK and Reno provide better latency under RED algorithm than under Droptail algorithm.

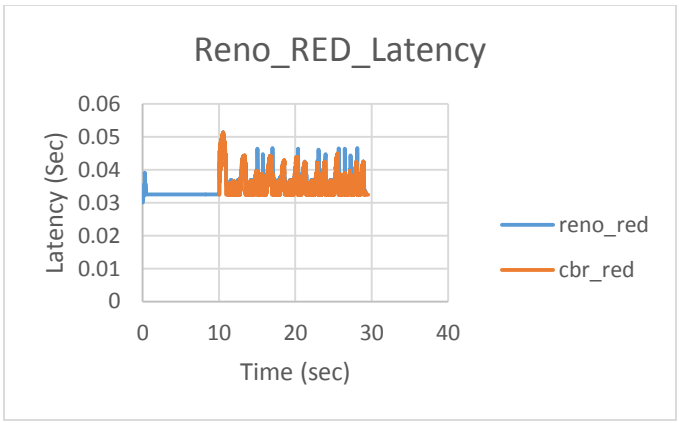

Fig 3.12

Reno has the same latency as SACK under Droptail algorithm which is 0.05. Fig 3.12 shows the latency of Reno under RED algorithm. Latency of Reno increased up to 0.048 which lower than that under Droptail algorithm.

Thus, it can be inferred that Reno also performs better under RED algorithm than under Droptail algorithm.

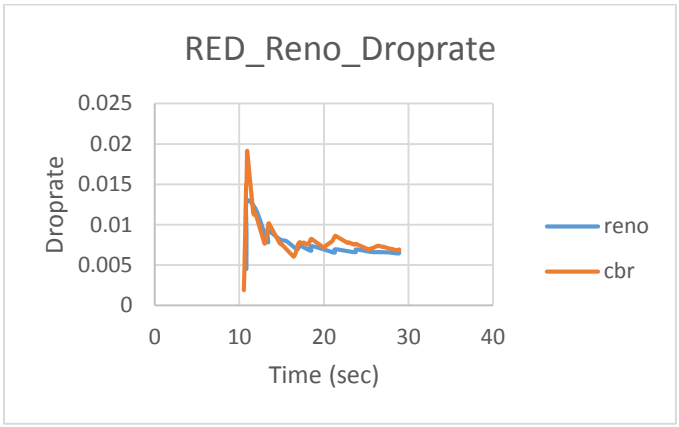

Fig 3.13

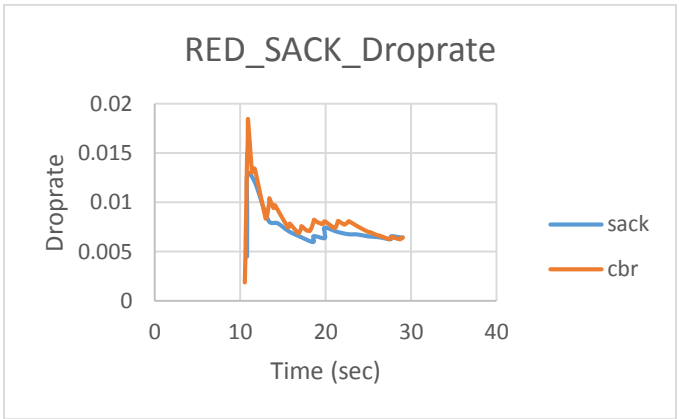

Fig 3.14

Fig 3.13 and Fig 3.14 show that after tenth second packet drop rate of Reno and SACK increased up to 0.013 and then dropped to 0.007. But in case when Droptail was implemented as the queuing algorithm there were no dropped packets at all.

4 Conclusion

Comparison of various TCP variants enable us to predict the level of performance we will get when different TCP variants communicate with each other in the real world.

While comparing the performance of various TCP variants, we concluded that the overall performance of TCP Vegas is better than the other variants. Vegas adopts a good strategy to deal with congestion as it can detect congestion before it actually occurs, which is not the case with TCP Tahoe, Reno and NewReno. We also concluded that although all the TCP variants should be fair with each other, this is usually not the case. There is one exception to this, Reno shows fairness with Reno. Thus, if fairness is the concern then Reno can be preferred over other variants.

There are many more TCP variants whose performance can be analyzed like BIC, CUBIC, Hybla and Compound.

5 References

1. "Random Early Detection (RED)". Accessed October 26, 2014. http://lartc.org/howto/lartc.adv-qdisc.red.html

2. Larry Peterson and Limin Wang. Understanding TCP Vegas: Theory and Practice. February 2000. Accessed October 26, 2014. ftp://ftp.cs.princeton.edu/techreports/2000/616.pdf

3. Samios C B and Vernon M K, Modeling the Throughput of TCP Vegas (ACM SIGMETRICS Performance Evaluation Review, 2003). Accessed October 26, 2014. http://pages.cs.wisc.edu/~vernon/papers/sword.03sigm.pdf

4. Joel Sing and Ben Soh, TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency (Network Computing and Applications, Fourth IEEE International Symposium on. IEEE, 2005). Accessed October 26, 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1565940

5. U. Hengartner1, J. Bolliger and Th. Gross, TCP Vegas Revisited (Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. IEEE, 2000). Accessed October 26, 2014. http://www.cs.cmu.edu/~uhengart/infocom00.pdf