Performance Analysis of TCP Variants

Dhanush Thotadur Divakara, Vinay Hegde

College of Computer and Information Science, Northeastern University, Boston, MA-02115
[1]dhanush@ccs.neu.edu
[2]hegde5@ccs.neu.edu

***Abstract* – Since the advent of Transmission Control Protocol(TCP) , reliable, ordered delivery of packets between applications on hosts running on an IP network was made possible. The initial design did not perform well in large congested networks. Several TCP variants have been proposed since then TCP Tahoe, Reno, New Reno, Vegas, BIC, CUBIC, SACK to name a few. Each variant handles congestion in ways that maximize throughput and minimize latency and the number of packet drops. The purpose of this paper is to analyse the performance of these different TCP variants. This is done by performing experiments and analysing the behaviour of the TCP variants under various load conditions and queuing algorithms by running them in a simulated environment.**

I. INTRODUCTION:

The TCP variants implements one or more of these algorithms to handle congestion 1. Slow Start 2. Congestion Avoidance 3. Fast Retransmit 4. Fast Recovery. We try to contemplate the results of the experiments based on our knowledge of how these variants implement these algorithms. We used the Network Simulator 2 (NS-2) to perform the experiments and analyse TCP Performance under Congestion, Fairness between the Variants and the influence of Queuing.

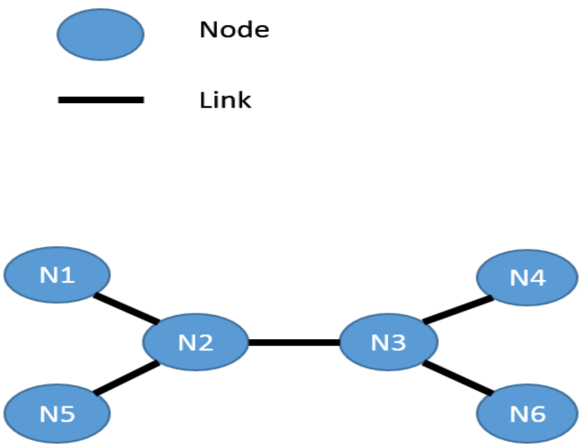

Fig 1.0

II. METHODOLOGY

To study the behavior of the TCP variants, we set up a network topology. The Network topology is setup as shown in the fig 1.0 which has 6 nodes and 5 duplex links connecting them. The network topology is setup by writing a tcl script and then setting the protocols that the nodes and links implement, start and end time of the flow of packets. We then run the simulation and generate a tracefile which contains indicators for the no of packets dropped, packets received at a node, start and end times, source address and destination address, packet id, flow id , packet size etc.. The tracefile is then parsed by writing a python script which performs all the necessary calculations (throughput, packet drop and latency) and then writes them into separate files from which the values are extracted and plotted using MS-Excel.

III. EXPERIMENT ONE

Performance of TCP variants under congestion: The experiment setup has an FTP application over a TCP agent starting at Node N1 which flows through Node N2 , Node N3 and finally sinks at Node N4. The Bandwidth of links is set as 10Mbps and we use a droptail queue with size as 10. Further in this experiment we introduce a Constant Bit Rate flow over UDP from Node N2 to Node N3.By increasing the value of cbr in increments of 0.1 we introduce congestion in the link connecting N2 and N3. Each flow is identified by their flow id and we observe the TCP flow . We start the CBR flow after 0.1 seconds and FTP after 1.0 seconds and stop the CBR flow after 9.5 seconds and FTP after 8.5 seconds. We perform these simulations for four variants namely Tahoe, Reno, New Reno and Vegas.

We run the simulation for each CBR value until the link has some decent congestion and reaches its bottleneck capacity and forces a packet drop. We run close to a 100 simulations and analyze the throughput, latency and packet drop for all the four variants by recording the values and plotting them on a graph. The purpose of this experiment is to analyze the graphs and conclude which TCP variant(s) achieves the highest throughput, which has the lowest latency, fewest number of packet drops and finally sum if there exists a single best variant or if it depends on the circumstances.

IV. EXPERIMENT TWO

Fairness Between the Variants: The experiment setup has two TCP flows with source nodes as N1 and N5 and sinks at N4 and N6 respectively. The source nodes run an FTP application over a particular TCP agent. The bandwidth of the links is set as 10 Mbps and we use a droptail queue with size as 10. Like in the previous experiment we introduce a Constant Bit Rate flow over UDP from Node N2 to Node N3. By increasing the CBR in increments of 0.1 we introduce congestion in the link connecting N2 and N3.We start CBR after 0.1 seconds and then start both the TCP flows after 1.0 seconds and stop the CBR flow after 9.5 seconds and similarly FTP over TCP after 8.5 seconds. We perform the experiments for different combinations of TCP flows at Nodes N1 and N5 namely, Reno/Reno, NewReno/Reno, Vegas/Vegas, NewReno/Vegas.

We run the simulation for each CBR value until the bottleneck capacity is reached and forces a packet drop. After running almost 100 simulations, we analyze the throughput, latency and packet drop for all the four combinations by recording the values and plotting them on a graph. The purpose of this experiment is to analyze the graphs and conclude if the different combinations of variants are fair to each other, if there is any such combination where one TCP variant is not fair to the other and in such a case the reason for the unfairness.

V. EXPERIMENT THREE

Influence of Queuing: Here we analyze how using different Queuing mechanisms influences the performance of the TCP variants. For this we use Droptail queuing mechanism and the Random Early Drop (RED) queuing mechanism. We start an FTP application over TCP at node N1 and its sink at node N4. A CBR flow over UDP is initiated at node N5 and sinks at node N6. Here we plot the performance of TCP and CBR flow over time. The variants we use for this experiments are TCP Reno and TCP Sack.

We start the ftp application after 0.1 seconds and then start CBR flow after 10 seconds. Both FTP and CBR flow stop after 25 seconds. The bandwidth of the lnks is set as 10 Mb and the value of the CBR here is set as 8 Mb. The queue size for RED and droptail queue is set as 5. By plotting the graphs for TCP Reno and TCP Sack we infer if each queuing discipline provides fair bandwidth to each flow. We later analyze how the TCP flow reacts to the creation of the CBR flow and finally conclude if using Random Early Drop a good idea while dealing with Sack.

VI. ANALYSIS OF EXPERIMENT ONE

The throughput values for all the variants are plotted for different CBR values starting from 1 and is increased in increments of 0.1 until the value of the CBR equals the link bandwidth.

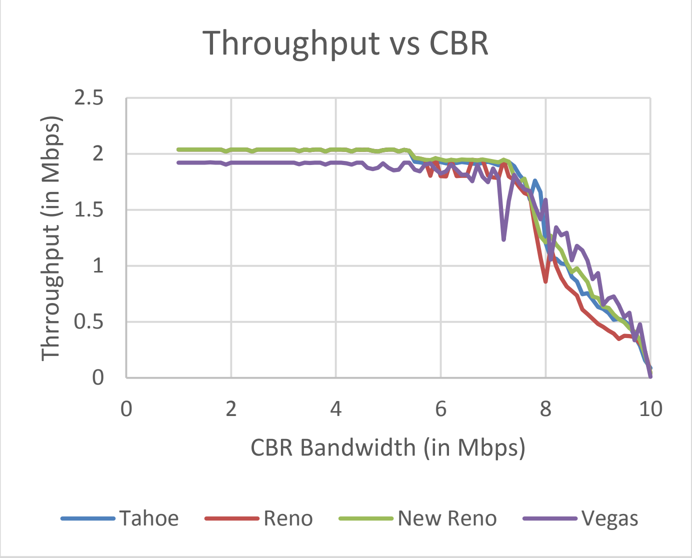

Fig 1.1

We can infer from the graph that congestion starts when there is a fluctuation in the throughput values. The see-saw pattern can be observed for CBR values of 5 and higher i.e. by increasing the CBR flow, we are introducing congestion in the link between N2 and N3. The see-saw pattern has asymmetric dips and highs in values of throughput. The dips are due to reduction in the size of the congestion window which is basically how the variants handle congestion in general. After the onset of congestion the variants differ in the way they handle congestion. Reno waits for 3 DUP ACKS and then retransmits the packets reducing the congestion window to half the original size while New Reno differs from Reno by retransmitting after receiving the first DUP ACK. Thus New Reno is better in handling multiple packet drops. This is quite evident from the graph where these two variants are essentially the same during the slow start phase, but after the first packet drop New Reno performs better than Reno.

The average throughput for Reno during congestion is 1.17 Mbps while New Reno has a better average of 1.30 Mbps. Both Reno and New Reno probe trying to maximize utilization until they reach congestion phase. They use packet drop as a signal for congestion. Vegas on the other hand uses an estimation of propagation delay as a signal for congestion. During peak congestion Vegas performs better than the rest of the variants which is evident from Fig 1.1. The throughput for Vegas during the initial phase (when there is no congestion) is slightly below the rest of the variants. This may be due to the fact that Vegas estimates the Base RTT value as the RTT value for the first packet, and doesn’t increase the congestion window until every other RTT , while Reno and New Reno probe trying maximize utilization until the first packet is dropped.

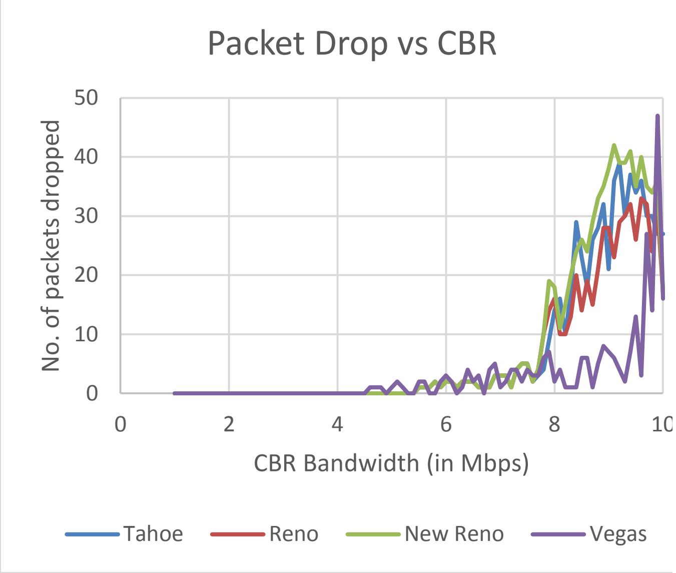

Fig 1.2

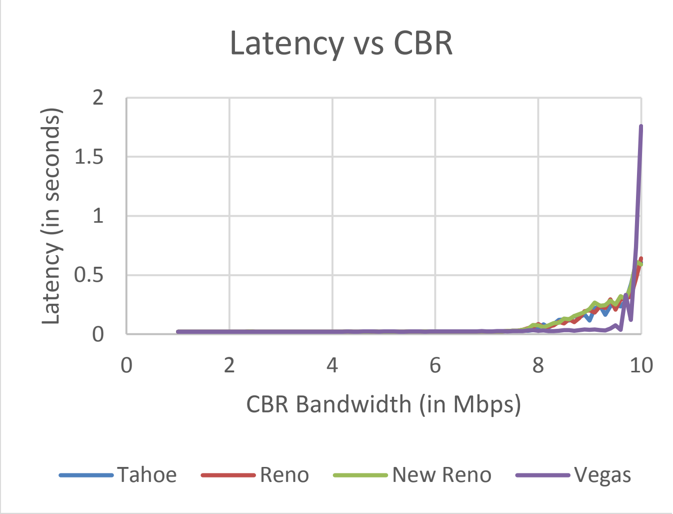

Fig 1.3

Vegas as we know uses RTT value to handle congestion, it estimates the congestion in the link which is better than the other variants where they essentially wait till the first packet to drop. By looking at the plot for number of packets dropped from fig 1.2, Vegas performs the best and has the least packets dropped. The latency plot also supports the fact that Vegas performs the best during congestion. Vegas has the least latency when compared to Tahoe, Reno and New Reno. The average latency for Vegas is 0.038 seconds which is way less than that of Tahoe (0.0607s), Reno (0.0603s) and New Reno (0.0663s). In general TCP New Reno and Vegas have higher average throughput values during congestion with Vegas being better than New Reno and the overall highest average throughput value however is achieved by New Reno (1.68 Mbps). Overall the average throughput values for all the variants differ only slightly from each other. Vegas has the least packet drop rate(2.146067) when compared to Tahoe(6.38), Reno(5.58) and New Reno(7.38). Since Vegas doesn’t allow a lot of packets to be dropped, it handles congestion better than the rest and thus we can conclude that Vegas performs better than the rest of the variants.

VII. ANALYSIS OF EXPERIMENT TWO

To analyze fairness we use the same metrics like in the previous experiment. We analyze the throughput, packet drop and latency plots for Reno/Reno , New Reno/Reno, Vegas/Vegas and New Reno/Vegas. Task is to determine if a particular variant is fair to the other variant while sharing a link. Looking at the throughput plot for Reno/Reno vs CBR the throughput values are almost identical oscillating with each other. Further taking the average throughput values for both the Reno Flows , the first flow has an average throughput of 1.26 Mbps and the second flow has 1.23 Mbps. There is not much of a difference when it comes to the number of packets dropped as well. The average no of packets dropped by Reno flow 1 is 12.3 and Reno flow 2 is 12.5.They oscillate between each other during congestion and this is due to the fact that Reno halves its congestion window after receiving three DUP ACKS, it enters retransmission and recovery phase. While Reno flow1 goes through retransmission and recovery, flow 2 gets that extra bandwidth, so flow 2 rises and vice versa. But not one flow dominates the other entirely. Thus we can say that for Reno/Reno both the variants are fair to each other or Reno is fair to itself when they share a link.

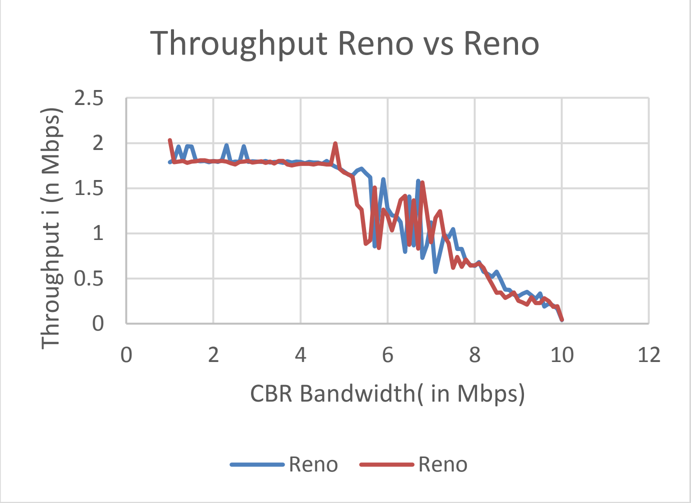

Fig 1.4

To analyze the fairness for NewReno/Reno we look at the throughput plot from fig 1.5. New Reno dominates Reno for almost all values of CBR. The average throughput for New Reno is 1.38 Mbps and Reno is 1.2 Mbps. New Reno clearly dominates Reno in terms of throughput especially during congestion. Since New Reno dominates Reno , we can say that New Reno is not fair to Reno in terms of sharing the bandwidth. This is because Reno waits for three DUP ACKS to retransmit while New Reno waits for only one DUP ACK to retransmit thus getting a higher share of bandwidth.

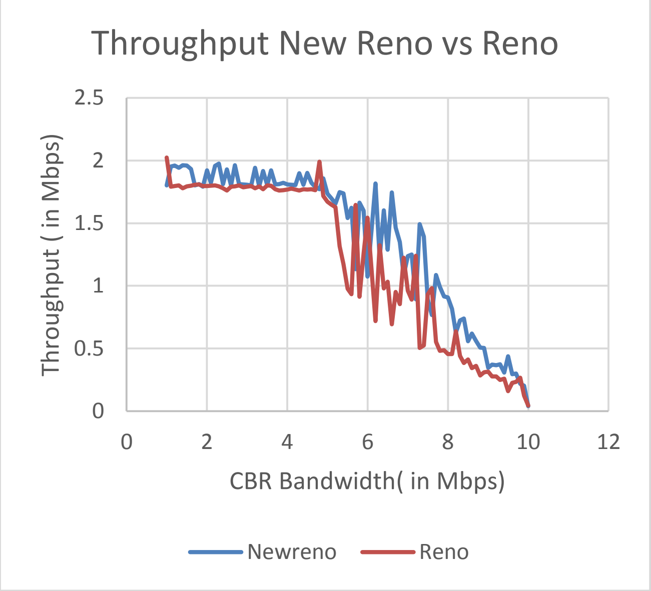

Fig 1.5

To analyze the fairness for Vegas/Vegas we look at the plot for latency. Now since Vegas detects congestion based on the estimation of RTT, it's fair to analyze the plot for latency. Looking at the plot from fig 1.6, both the flows overlap each other for most values of CBR. Further the average latency for flow 1 is 0.07 seconds and flow 2 is 0.08 seconds. The standard deviation for flow 1 is 0.24s and flow 2 is 0.25s. Thus we can conclude that one Vegas flow doesn't dominate the other or that Vegas is fair to itself.

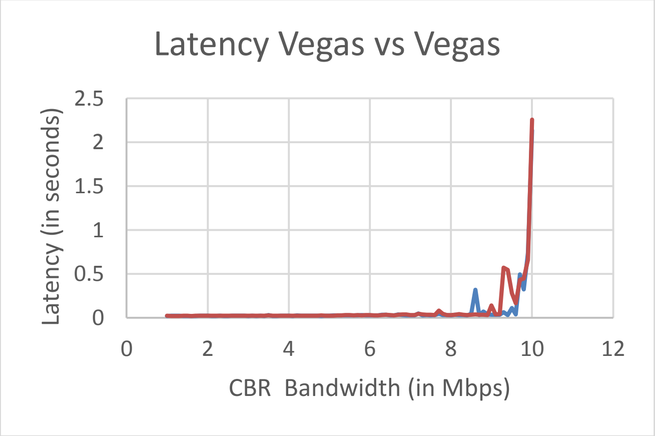

Fig 1.6

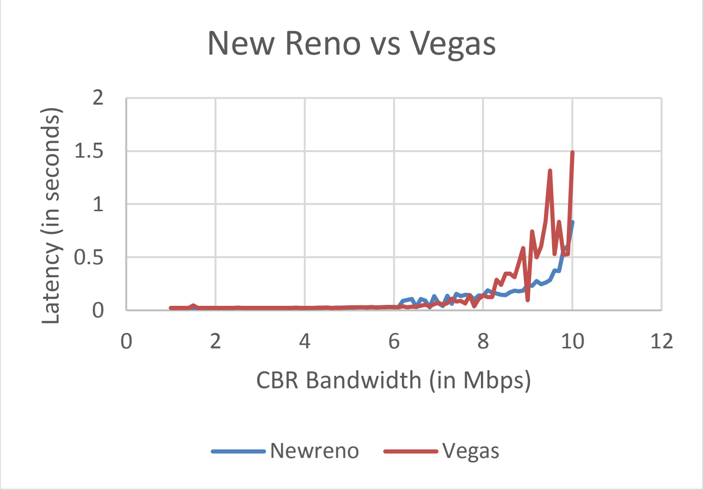

Fig 1.7

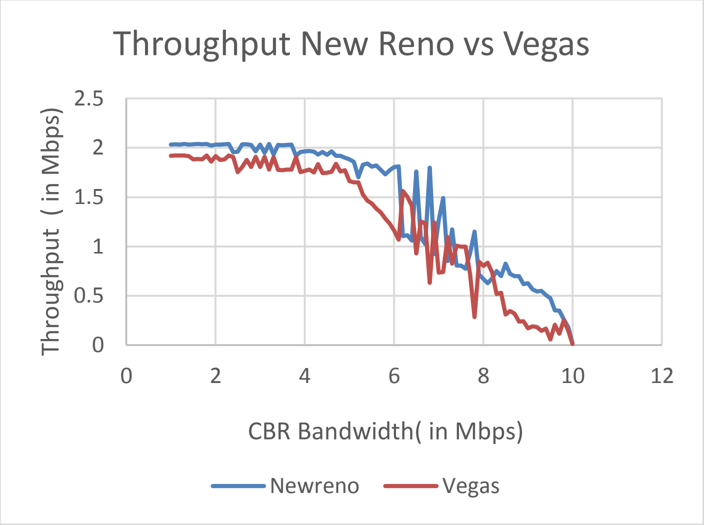

Fig 1.8

To analyze the fairness we look at the latency plot for NewReno/Vegas from 1.7. They pretty much overlap each other during the initial phase, but once congestion sets in, New Reno has a lower latency when compared to Vegas. The average latency for New Reno is 0.098s and 0.14s for Vegas. Further the throughput plot for New Reno/Vegas from fig 1.8 shows that New Reno has a higher throughput when compared to Vegas. The average throughput for New Reno is 1.45 Mbps and 1.2 Mbps for Vegas. Clearly New Reno dominates Vegas in terms of Bandwidth utilization. New Reno increases its congestion window until it encounters a packet drop which is the first signal for congestion while, Vegas handles congestion by estimating the delay in the link and changing the congestion window dynamically. Vegas detects congestion in the link and restricts the increase of congestion window. New Reno thus gets more bandwidth. Thus we can conclude that New Reno dominates Vegas and is unfair to Vegas.

VIII. ANALYSIS OF EXPERIMENT THREE

The most common implementation in router queues on the internet is the droptail queuing mechanism. It works by queuing up packets until a certain amount and then drops all the packets that come after this point. We compare the influence queuing mechanisms by comparing throughput and latency plots for TCP Reno vs CBR and TCP Sack vs CBR when they use droptail queuing mechanism and Random Early Detection queuing mechanism. RED statistically drops packets from flows before it reaches its hard limit. This causes the congested link to grow more gracefully and prevents retransmit synchronization. By allowing some packets to get dropped sooner keeps queue sizes low and latency under control.

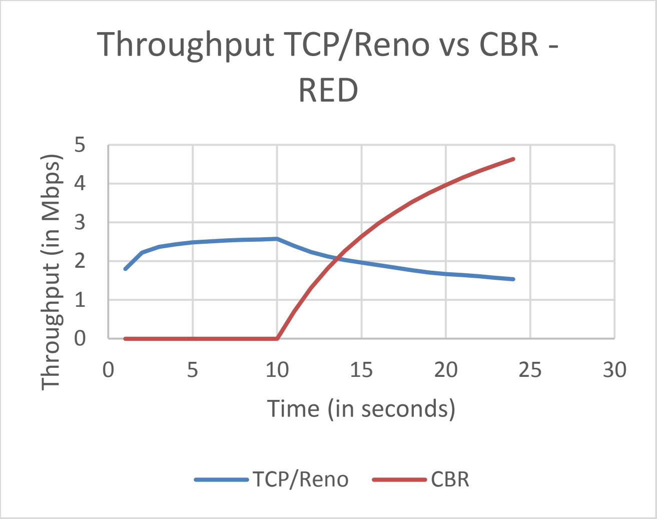

Fig 1.9

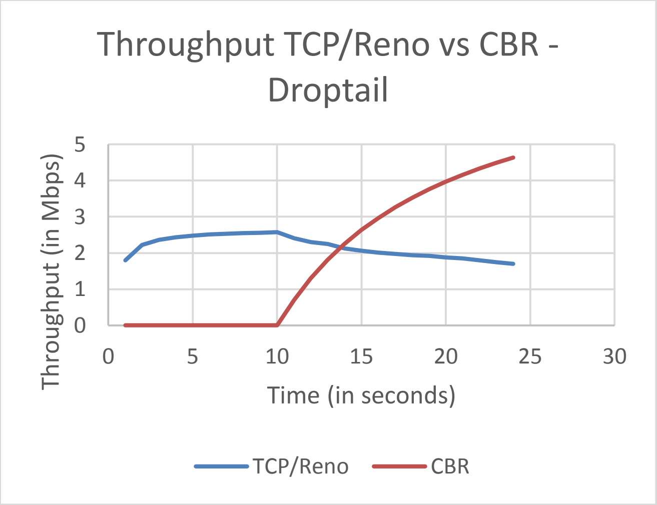

Fig 2.0

RED is more fair than droptail, in the sense that it does not possess a bias against bursty traffic(We introduce bursty traffic when CBR is introduced) when there is congestion in the link. The more a host transmits, the more likely it is that its packets are dropped as the probability of a host's packet being dropped is proportional to the amount of data it has in a queue. Early detection helps avoid TCP global synchronization. To conclude droptail mechanism does not provide fair bandwidth to each flow. RED in terms of fairness is better than droptail.

The end to end latency for droptail queuing mechanism is higher than that for RED. This is because once the droptail queue reaches its limit, it enters retransmit synchronization. When retransmit synchronization occurs, the sudden burst of drops from a router that has reached its limit will cause a delayed burst of retransmits, which will over fill the congested router again leading to higher latency. Since RED statistically drops packets from flows before it reaches its hard limit, the congested link grows more gracefully and prevents retransmit synchronization. This basically drops packets in a controlled manner achieving low end-to-end latency.

By looking closely at the throughput plots for Reno vs CBR for both droptail and RED, the creation of CBR flow causes a dip in the throughput for TCP/Reno. The creation of CBR flow causes congestion in the network. This makes the existing TCP flow to drop packets thus reducing its throughput.

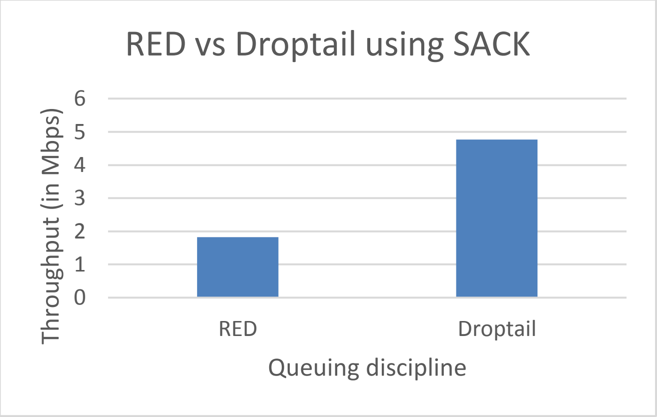

Fig 2.1

Analyzing the throughput plots for TCP/Sack vs CBR using droptail and RED. For RED the throughput value is 1.828 Mbps while using droptail yields a higher throughput of 4.77 Mbps. Further the total number of packets dropped when RED is used is 40 and only 31 when droptail is used. RED is thus not a good option for TCP/sack because of its higher packet drop and low average throughput when compared to using droptail for TCP Sack.

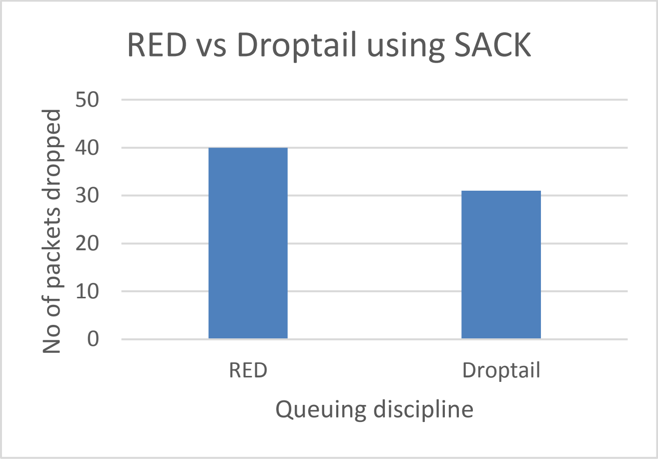

Fig 2.2

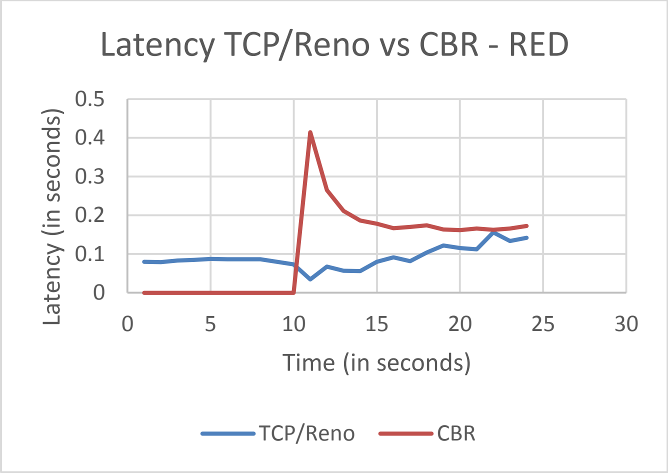

Fig 2.3

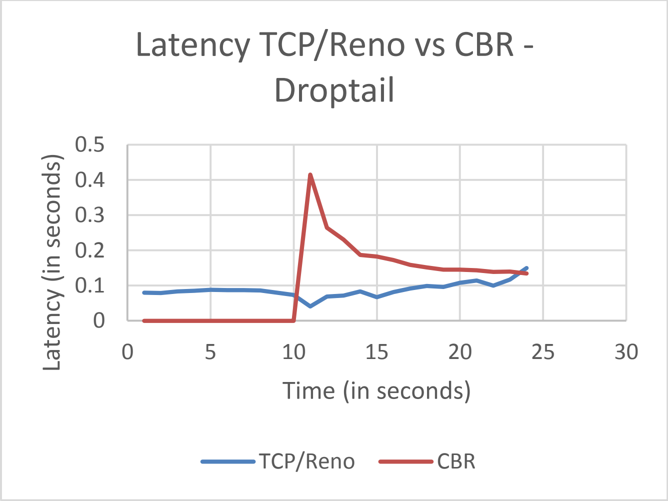

Fig 2.4

IX. CONCLUSION

To conclude which variant performs better we analyzed the performance of TCP Tahoe, Reno, New Reno and Vegas under congestion, analyzed if different combinations of TCP variants are fair to each other and finally compared the influence of using droptail and RED queuing mechanism. From the first experiment Vegas is clearly the best choice variant because of its low latency, high throughput and low packet drop when compared to Tahoe, Reno and New Reno during congestion. From the second experiment we observed from the NewReno/Reno plot that New Reno dominates Reno and that New Reno isn't fair to Reno variant. Similarly the same holds for NewReno/Vegas where New Reno is not fair to Vegas. Reno/Reno is fair to each other because the algorithm they use to handle congestion is essentially the same. The same explanation holds true for Vegas/Vegas. Finally from the third experiment we can infer that RED queuing mechanism is better than droptail in terms of handling bursty traffic because RED statistically drops packets and controls the flow of packets while droptail just drops all the packets when its queue size reaches its limit. However the same is not true when the TCP/SACK variant is used with RED.

X. REFERENCES

[1].http://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project 2/SACKRENEVEGAS.pdf

[2] http://www.icir.org/floyd/red.html

[3] http://home.iitj.ac.in/~ramana/sacks.pdf

[4] TCP Vegas: New Techniques for Congestion Detection and Avoidance L. Brakmo, S. O'Malley and L. Peterson *Proceedings of the SIGCOMM '94 Symposium* , August 1994, pg. 24-35.

[5] Issues in TCP Vegas Richard J. La, Jean Walrand, and Venkat Anantharam Department of Electrical Engineering and Computer Sciences University of California at Berkeley

[6] Random Early Detection Gateways for Congestion Avoidance Sally Floyd and Van Jacobson Lawrence