Performance Analysis of TCP Variants

Vinay Vishwanath
Northeastern University, Boston
vishwanath.vi@husky.neu.edu

Nupoor Nuwal
Northeastern University, Boston
nuwal.n@husky.neu.edu

1. Introduction

Transmission control protocol (TCP) has become one of the most ubiquitous protocols in the Internet Stack. TCP provides reliable means of communication from source to destination. It relies on retransmission to provide reliable data delivery from end to end. The need for TCP was realized with the growth of the internet. Aspects like delay, congestion, latency, packet drops, etc. came to be known and there was a need for a reliable mechanism to send and receive packets across the unreliable and constantly changing the internet. TCP establishes a connection and then relies on acknowledgments from the receiver to assert successful packet transmission. Such an approach is slower, but reliable.

Over the years, various approaches were proposed and studied to make TCP more efficient. However, the choice of an ideal TCP wasn't so obvious. Since each TCP performed in certain areas while it lacked in some other ones. This led users to choose a variant of TCP which was more apt for their needs. Thus various TCP implementations came to be introduced. Each different than the other in certain areas. The well-known ones being Tahoe, Reno, NewReno, Vegas, Cubic, SACK and Compound. In this paper, we discuss the variants of TCP and we analyze how one implementation varies from the other. We also analyze how the parameters associated with each variant behave under changing network conditions.

2. Methodology

We build a topology using NS2[1] simulator as it has inbuilt TCP variants we wish to compare and contrast. We modify parameters like link delay, queue buffer, bandwidth, queuing algorithm, the start and stop times of various flows to analyze.[3] We performed our experiments to compute throughput, latency, Round Trip Time(RTT) and Drop Rate.

Throughput is the ratio of the total size of TCP packets received at destination to time elapsed between receiving and sending the packets. Latency is defined as time taken by the packet to traverse from source to destination. Drop Rate is defined as,

$$\text{Drop Rate} = \frac{No\ of\ packets\ sent - No\ of\ packet\ received}{No\ of\ packet\ sent}$$

We use a python script which calls and executes the NS2 Tcl file where we create the topology and set various parameters like link bandwidth, link delay, queue buffer size, start and stop times of the various flows pertaining to the experiment and then processes the output, which is a trace file, to compute the required parameters. The script then writes to MS Excel sheet, to represent data graphically. We make use of pylab module in python to plot the graphs of experiment 3 since it allows us to see parameters under consideration well with time. We conduct a total of three experiments to perform the performance analysis. We compute the Mean, Variance and Standard Deviation for parameters in all variants and then perform a T-Test to prove which variant is better than the other. The formula used to compute T values is as below,

$$T = \frac{mean1 - mean2}{\sqrt{\frac{variance1}{N1} - \frac{variance2}{N2}}}$$

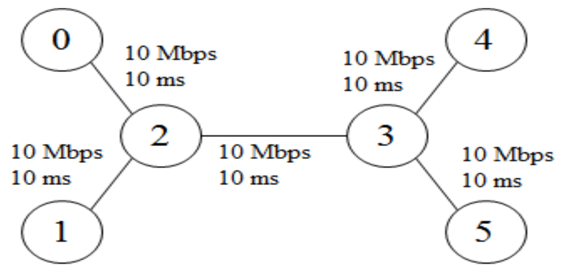

Figure 1: Network Topology

We define a FTP (File Transfer Protocol) source at node 1 and FTP sink at node 4 in experiment 1. This simulates the TCP flow we need. FTP is a standard network protocol operating on the application layer of the TCP Model. FTP is built on Client-Server architecture and utilizes separate control and data connections between the client and server. We also set CBR source at Node 2 and CBR sink at Node 3. Constant Bit Rate (CBR) service category is used for connections that transport traffic at a constant bit rate, where there is an inherent reliance on time synchronization between the traffic source and destination.

3.Experiment

In experiment 1 our focus lies on the throughput achieved by TCP in the presence of a CBR source in the common link. We run a TCP flow from Node 0 to Node 4 and a CBR flow from Node 2 to Node 3. Hence. we have the link between Node 2 and 3 as the common link. The CBR send rate is altered linearly from 1 to 10 Mbps. We see in each iteration; which TCP variant achieves the most throughput. In Figure 2 we see the throughput variation of different variants with increasing CBR.

The throughput reduces after CBR reaches 7 Mbps. This is the point when TCP encounters congestion and it reduces its congestion window subsequently. The uniqueness of the TCP variants is defined by the manner in which congestion window is decreased. TCP detects the onset of congestion by maintaining a counter which is set each time a packet is sent out. It calculates this value based on the formula [4]

*EstimatedRTT = (1-α) * PreviousEstimatedRTT + α * Sample RTT*

Where, α is usually 1/8

*Retransmission time-out (RTO) = Sample RTT * β*

Where, β is usually 2

In the event of a timeout or three duplicate ACKs, TCP Tahoe enters the slow start phase by reducing its congestion window to 1 and reducing the sstresh to half of the congestion window. In the slow start phase, TCP grows it congestion window exponentially, in an effort to quickly get to the point just before where the congestion occurred. When the congestion window exceeds the sstresh, Tahoe enters a Congestion Avoidance phase. In this phase, the congestion window is increased more cautiously using the formula,

Increment = MSS × (MSS/Congestion Window)

Congestion Window +=Increment

Where MSS is the Maximum Segment Size of TCP.

TCP Reno and NewReno enter a fast recovery mode by reducing the sstresh to half of the value when congestion occurred and the congestion window is also reduced to half (and not to 1 as in with TCP Tahoe implementation). In the fast recovery phase TCP increases its congestion window for every ACK it receives. It stays in this mode until a new ACK is received which brings TCP out of fast recovery into Congestion Avoidance. In most cases TCP follows the saw tooth waveform and probing for the sstresh and tries to keep itself between congestion avoidance and fast recovery mode. A timeout in the fast recovery would take the TCP into the slow start phase. The retransmit time-out is considered as the recovery method of last resort. The increase of congestion window in congestion avoidance phase is termed as Additive Increase and reduction of the window by half on receipt of three duplicate ACKs is termed as Multiplicative decrease (AIMD).TCP uses acknowledgments to trigger (or clock) its increase in congestion window size, hence TCP is said to be self-clocking. In case of an event where no ACK is received, TCP loses clock synchronization and must wait for a time-out to fall back to slow start. [2]

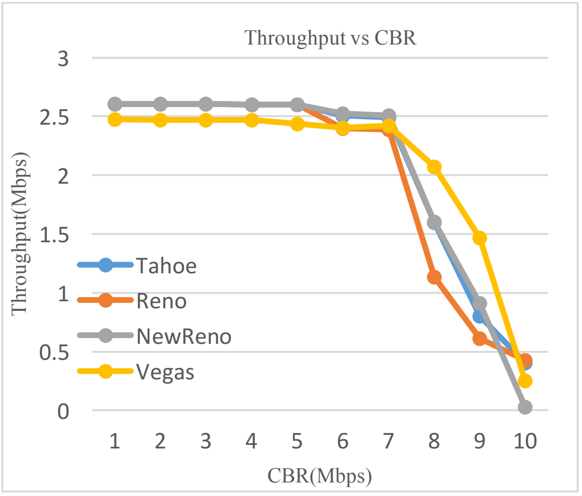

Figure 2: Throughput of TCP Variants

TCP Vegas on contrast has a different approach to detect congestion. TCP Vegas maintains the round trip times for every packet. It senses longer RTT's as the onset of congestion. In Figure 2, we observe that TCP NewReno performs better than Tahoe, Reno and Vegas in terms of achieving most throughput in conditions where CBR rate has not yet reached bottleneck conditions (i.e. for CBR less than 7 Mbps). This is can be justified since TCP NewReno is most aggressive in the manner it increases its congestion window. It stays in the congestion avoidance state longer than the other variants and continues to increase its congestion window consequently. TCP Vegas provides more throughput in high congestion.

The T-Test values for throughput are as follows:
Vegas over NewReno=0.09
Vegas over Reno=0.26
Vegas over Tahoe=0.036

From Figure 3, we see that, with increase in CBR rate, latency and RTT of the packets increases since there are more packet drops as the CBR rate increases, leading to more retransmissions.

We see least variance in the latency and RTT values of TCP Vegas, which is justified because TCP Vegas is most conservative about how it sends packets hence Vegas also reports the least Drop rate among all TCP variants as is seen from Figure 4

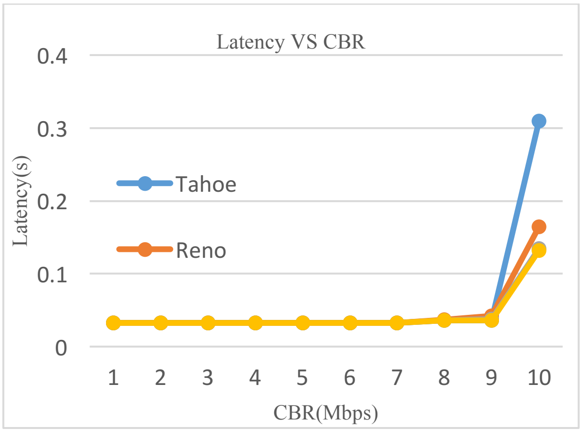

Figure 3: Latency of TCP Variants

TCP Vegas's conservative congestion window approach enables it to have lesser packet drops than all other TCP variants.

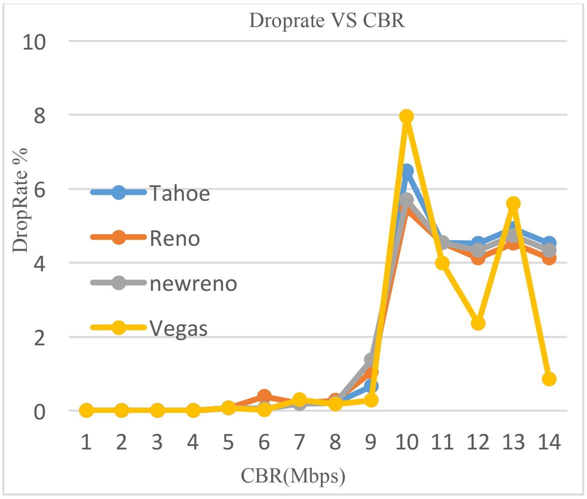

Figure 4: Drop rate of TCP Variants

We can infer from Experiment 1 that there is no single variant which is the best in all the scenarios. TCP NewReno achieves the highest throughput in low congestion conditions whereas, TCP Vegas is able to get more throughput when the congestion increases. Also TCP Vegas has the least latency and drop rates in congested links.

4. Experiment 2

In experiment 2 we try to analyze the fairness of TCP variants. TCP fairness is a must in todays' world where each one uses a different TCP variant than the other. The TCP fairness mechanism is required to do justice to each of those users. We see the throughput, latency, RTT and Droprate achieved by comparing two TCP flows in presence of a CBR in the common link. We make trials with all possible start and end time combinations for the three flows and see if the bandwidth is eventually distributed more or less equally between the two TCP flows. We modify the two flows to be of different TCP variants and check the fairness. In these experiments we set all link bandwidths to 10Mbps and link delays to 10ms. We also maintain the same queuing mechanism of Drop Tail at all nodes and we set the queue buffer size to 10 Mbps.

We observe that while the two TCP flows of the same variant, they are fair to each other in consuming the common link's bandwidth as seen in Figure 5 and 6. Reno1 and Reno2 as they share same Congestion Avoidance technique to handle the data and also there is not much change in the variance (VarReno1=0.617 and VarReno2=0.605) as seen in Figure 5. Thus illustrating their fairness towards each other.

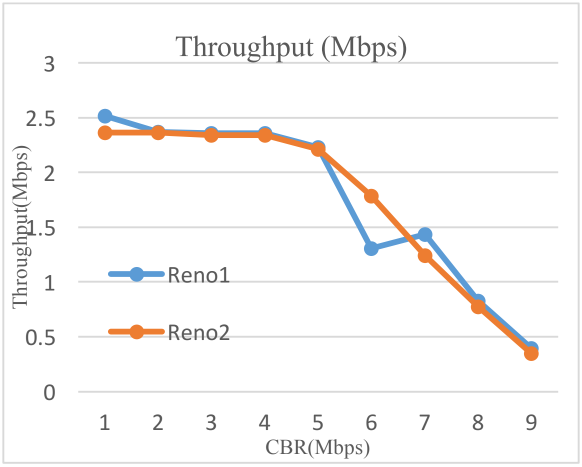

Figure 5: Throughput for Reno/Reno starting together and CBR at 10

We see in Figure 6 that Vegas1 and Vegas2 has almost same throughput and hence they are fair to each other. Similarly, in case of RTT as seen in Figure 9.

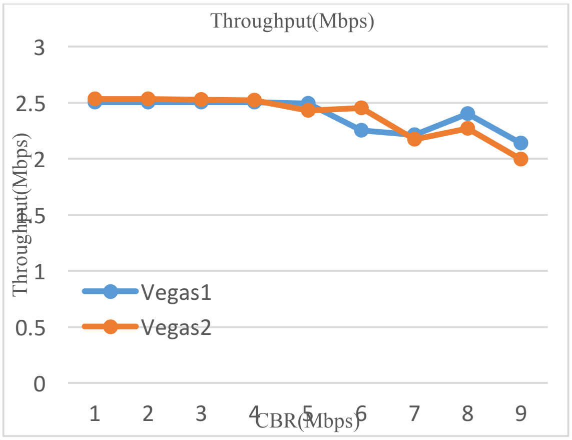

Figure 6: Throughput for Vegas/Vegas starting together and CBR at 10

In Figure 7, we observe New Reno has more throughput than Vegas as Vegas decreases its throughput when it detects congestion to reduce the loss of packet [6] and New Reno increases its window size cautiously in fast recovery. Thus are unfair to each other.

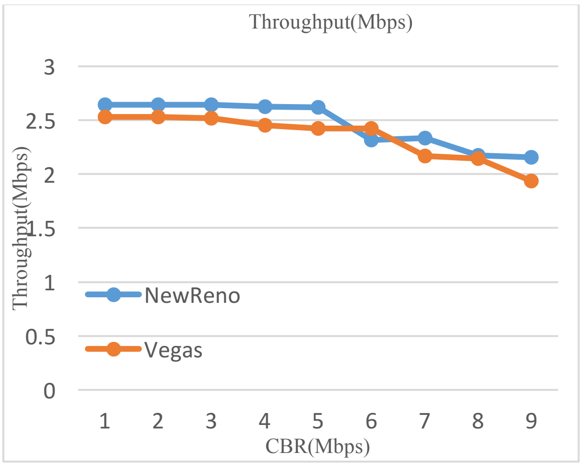

Figure 7: Throughput NewReno/Vegas starting together and CBR at 10

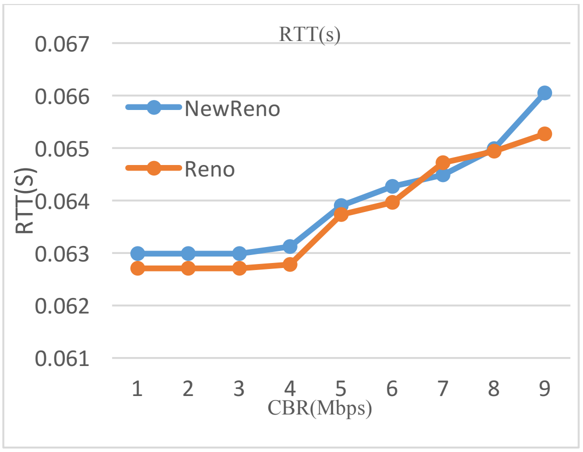

Figure 8: RTT NewReno/Reno starting together and CBR at 10

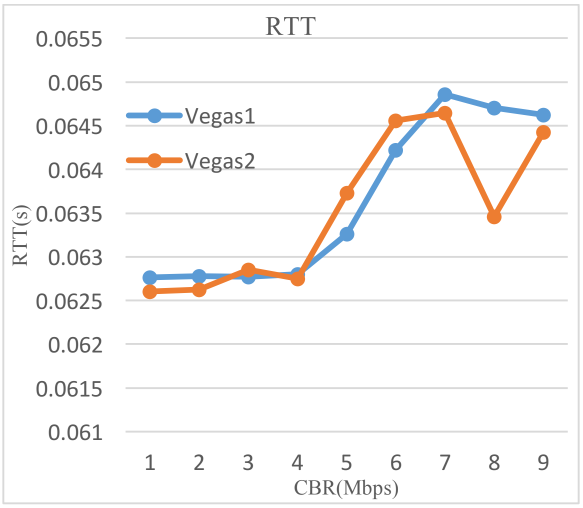

Figure 9: RTT Vegas/Vegas staring together and CBR at 20

In Figure 8 RTT of NewReno is lower than Reno as number of retransmission in Reno is more which leads to more queuing delay.

We also observe that the RTT values corresponding to New Reno are lower than that of Vegas as seen in Figure 10. The retransmissions due to packet drops increase the RTT of the flows. The lower RTT values can be associated to lower drop rates as can be seen in Figure 11.

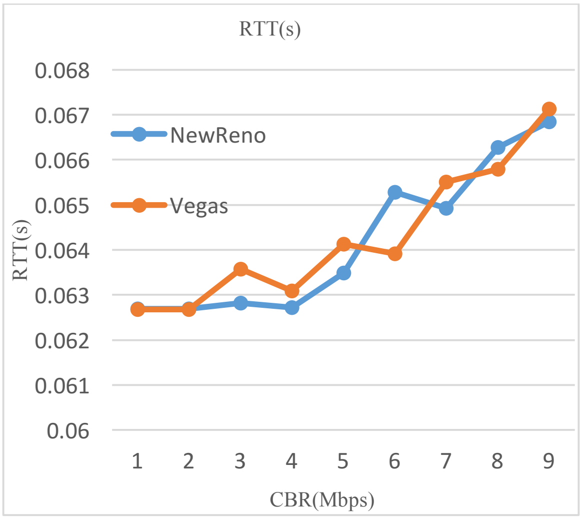

Figure 10: RTT NewReno/Vegas starting together and CBR at 20

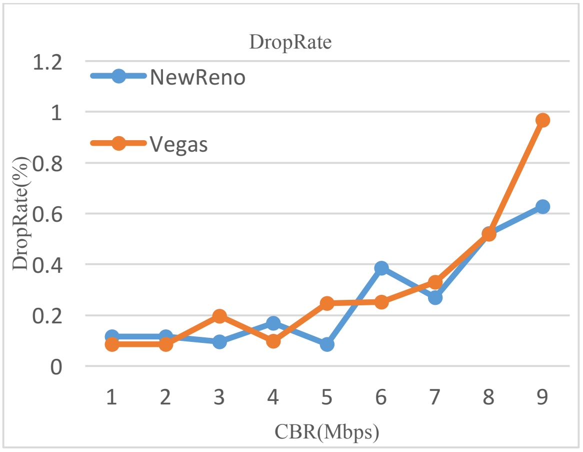

Figure 11: Drop Rate NewReno/Vegas starting together and CBR at 20

Since each TCP variant acts differently to the congestion event, by changing their congestion windows in a different fashion, we see that some variants outperform the other and are not fair to each other. We can infer that throughput is linked to RTT in an inverse fashion. The lower the RTT for a Variant, the more data it is able to send, hence it has a higher throughput. Hence different TCP variants are not fair to each other. But they try to converge to a more or less similar value over time.

5. Experiment 3

In experiment 3 we study the two most commonly deployed queuing mechanisms viz. Drop Tail and RED [5]. We analyze how the traffic flows are affected in presence of the queuing algorithm employed at the routers. The primary goal of Queue Management is controlling average queuing delay, while still maintaining high link utilization. The secondary goals are:

1. Improving fairness (e.g., by reducing biases against bursty low-bandwidth flows)
2. Reducing unnecessary packet drops and global synchronization
3. Accommodating transient congestion (lasting less than a round-trip time)

Drop Tail is a simple Active Queuing Management (AQM) algorithm used in many routers. It doesn't differentiate traffic from different sources. As long as the queue is filled up, it will drop subsequent packets arrived. In other words, drop the tail of sequence of packets.

Random Early Detection or Random Early Drop (RED) is another AQM. It monitors the average queue size and makes decision about the packet (either drop or mark) based on statistical probabilities.

RED operates in three regions as shown in Figure 12.

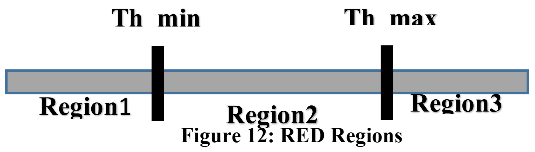

Figure 12: RED Regions

When the average queue size is lower than Th_min, all packets are accepted; when the average queue size is between Th_min and Th_max, RED drops or marks a packet by a probability, which is calculated based on the average queue size, and the number of packets transmitted since the last dropped/marked packet in this flow. When average queue size grows beyond Th_max, all packets will be dropped/marked. RED monitors the average queue size to detect congestion, drops/marks the packet randomly to avoid TCP global synchronization, and allows fluctuations in the actual queue size to accommodate bursty traffic and transient congestion. Hence RED is cautious about the packets it fills in the buffer.

We conduct experiments to study the Drop Tail and RED algorithms using TCP Reno and TCP SACK variants. SACK differs from Reno in which it maintains a variable called pipe which it increments when the sender sends a new packet or retransmits an old one. And the pipe is decremented when the sender receives a duplicate ACK packet with the SACK option reporting that new data has been received at the receiver. The SACK report tells the sender of all the packets not received by it. The sender maintains a database of acknowledgements from previous SACK options. When the sender is allowed to send a packet, it retransmits the next packet from the list of packets which it assumes to be missing at the receiver. But in case of loss of retransmitted packet, SACK has to wait for the timeout and then begin with slow start all over again.

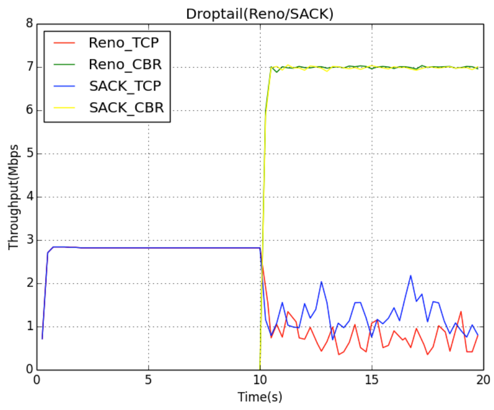

Figure 13: Throughput vs Time for Drop Tail Algorithm

In Figure 13, we see that as soon as the CBR flow starts at 10 seconds, the TCP throughput falls down drastically. The occurrence of multiple packet drops within the same window, makes the TCP variants enter slow start. For the same AQM (i.e. Drop Tail) TCP SACK achieves higher throughput than Reno. This is because SACK uses report to convey to the sender about the packets received and the packets dropped. Hence the sender is able to send only the required packet to the receiver without the need to retransmit the other already received packets as in the case with Reno.

In Figure 14, we see the the throughput variations of SACK and Reno while RED AQM is employed. We see the two flows achieving more or less similar throughputs even after the CBR flow has been initiated. RED also achieves higher throughput values for the two flows as compared to Drop Tail.

Since Drop Tail's FIFO buffer is flooded by a burst of CBR at 10 seconds, hence the TCP packets incoming at that instance find the buffer to be full. Since the throughput values corresponding to two flows using RED is higher than Drop Tail. RED on finding a huge burst of data packets on onset of CBR flow, avoids TCP global synchronization, since it is able to calculate receivers queue size and take preventive actions before a drop event occurs. This phenomenon occurs when sudden burst of traffic (CBR) causes multiple packet loss across TCP flows present on the common link. In event of global synchronization, instead of discarding many segments from one connection, Drop Tail discards one segment from several connections reducing the throughput on all of the links.

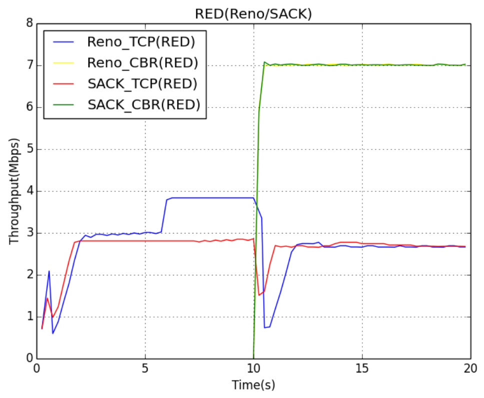

Figure 14: Throughput vs Time for RED Algorithm

We also see that RED is able to recover from multiple packet loss event earlier than Drop Tail.
Figure 15 shows a bar graph of latency achieved by Reno and SACK under Drop Tail and RED algorithms. Drop Tail results in higher packet drop compared to RED. This leads to more retransmission while using Drop Tail which ultimately increases queuing delay and latency. Thus SACK with RED Algorithm is a good idea.

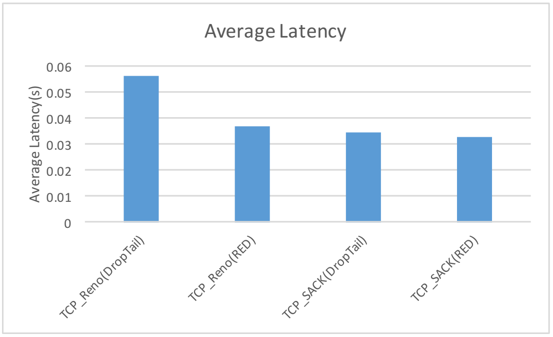

Figure 15: Average Latency for RED and Drop tail Algorithm

Conclusions

Conducting these experiments helped us understand the subtle differences in the TCP variants. The analysis we performed in the experiments made us understand how and why certain things are implemented the way they are.
TCP variants are implemented keeping in mind what purpose they are intended for. There is clearly no one particular variant which serves the best in all scenarios. There is a constant trade of to be made between the various outputs one can with TCP.
From Experiment 1 we infer that TCP Vegas is able to achieve most throughput amongst all other variants when the congestion due to CBR increases. Vegas is able to sense the onset of congestion early, since it uses delay to estimate congestion, unlike TCP Tahoe, TCP Reno and TCP NewReno, which use packet drops as an indication.
From Experiment 2 we infer that TCP variants are not really fair to each other. The similar variants are fair to each other. But different variants, are not fair since, each of the variants congestion control mechanism is different than the other.
Experiment 3 helped us learn how queue management plays a vital role in the congestion avoidance. It is extremely essential to have a robust and dynamic queue management algorithm which is able to handle the dynamic nature of today's internet. From our findings we infer that RED queuing algorithm is a better alternative to Drop Tail. RED avoids TCP Global synchronization and thus allows the TCP SACK and Reno flows to achieve higher throughputs. The TCP flows are able to recover from packet drops quicker while using RED algorithm. SACK TCP is able to outperform Reno in cases when either of the Active Queue Mechanism were used due to the selective retransmissions in SACK TCP.
This area continues to be the focus of many studies. Since TCP is such an integral aspect of internet today, there is constant effort ongoing to improvise TCP and achieve higher throughput, security, and link utilization, at the same time keeping packet drops and latencies to values as low as possible.

References

[1] Marc Greis' Tutorial for UCB/LBNL/VINT Network Simulator. http://www.isi.edu/nsnam/ns/tutorial

[2] Yuvaraju B N & Dr. Niranjan N Chiplunkar for Scenario Based Performance Analysis of Variants of TCP Using NS2-Simulator.

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

[4] Kurose Ross Computer Networking A Top-Down Approach.

[5] Mohammad Abu Obaida,Md. Sanaullah Miah and Md. Abu Horaira for Random Early Discard (RED-AQM) Performance Analysis in Terms of TCP Variants and Network Parameters: Instability in High-Bandwidth-Delay Network.

[6] Luigi A. Grieco and Saverio Mascolo Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari for Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control