# Performance Analysis of TCP Variants

He Shi, Shifu Xu
School of Computer and Information Science, Northeastern University, MA

## Introduction

The reason why we will introduce TCP (Transmission Control Protocol) is that it will help transfer data stream between two hosts reliably. However, it was unable to provide very good performance when the network is congested. As far as we known, there are some variants (TCP Tahoe, Reno, NewReno, Vegas, BIC, CUBIC, SACK and others) is able to control the congestion. The purpose of this paper is to analyze the performance of these different TCP variants under different load conditions and queuing algorithms. In this paper, we will use NS-2 network simulator to do experiments.

The first experiment is about comparing four different TCP variants' performance under congestions, and they are Tahoe, Reno, NewReno and Vegas. The result shows that Vegas performs best under congested network. NewReno and Reno perform worse than Vegas, but they perform better than Tahoe. In addition, NewReno always performs better than Reno. The second experiment is about dealing with the fairness between different TCP variants. We find that Reno/Reno is very fair, but NewReno/Reno, Vegas/Vegas and NewReno/Vegas are unfair combinations. The third experiment is about analyzing how TCP Reno and SACK perform using different queuing algorithm: DropTail and RED under various congestion conditions. It is concluded that both Reno and SACK works well under the DropTail queuing algorithm, since the througphput of Reno and SACK under DropTail will not oscillate wildly like they work under RED. But it does not mean that RED is not good. Actually, REC has it strength, and it is lower latency. Using RED or DropTail queuing algorithm should depend on the real network situations.

## Methodology

In order to compare and analyze the performance of different TCP variants, we will use some key parameters like average throughput, average latency and drop rate to compare each TCP variant. The paragraph below will introduce how to execute every experiment one by one. All the experiments will need tool called NS-2 simulator and statistic software Excel.

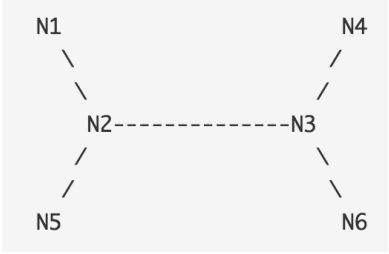


Figure 1.1 Topology

In the first experiment, we will set the network topology like the figure 1.1 using NS-2 simulator firstly. During this procedure, the bandwidth of the link is set to be 10Mb and the queuing algorithm is set to be DropTail. In the tcl code, a random seed is inserted so as to make sure that we can get different result every time. Only in this way can we make the experiment much more trustable. Then, add CBR source at node 2, and add one TCP stream from node 1 to node 4. Next, increase the CBR of UDP stream from 1Mb to 10Mb, and record the result for each TCP variants. We will repeat 100 times for every variant in certain CBR. At last, we will use the python program we designed to parse the value of average throughput, average latency and average drop rate. The formulas for computing throughput, drop rate and average latency are as follows:

$$drop\ rate = \frac{dropped\ packets\ * 100\%}{dropped\ packets + reveived\ packets}$$

$$throughput = \frac{\frac{sum}{sum(received\ packets)\ * packet\ size}}{time}$$

$$average\ latency = \frac{\sum trip\ time\ of\ received\ packet\ i}{number\ of\ received\ packets}$$

In the second experiment, we will analyze the fairness between different TCP variants. There are four pairs and they are Reno/Reno, NewReno/Reno, Vegas/Vegas and NewReno/Vegas. Firstly, start three flows: one CBR, and two TCP. Then, add a CBR source at node 2, and then add two TCP streams from node 1 to node 4 and node 5 to node 6 respectively. Next, we will change the start order of TCP variant in the same combination. Lastly, run parsing algorithm to get average throughput, average latency and average drop rate, and then plot them out. According to the results, we can answer whether these combinations are fair or not.

In the third experiments, we will compare two different queuing algorithms that control how packets in a queue are treated. Firstly, use the same TCP topology from experiment 1 and add one TCP flow from node 1 to node 4 and one CBR/UDP flow from node 5 to node 6. Next, choose a value of CBR that will make the TCP flow steady, and then start the CBR source and analyze how TCP and CBR flows will change under the two following algorithms: DropTail and RED. After that, run the simulation program to perform experiments with TCP Reno and SACK. Lastly, use the parsing program to calculate average throughput, average latency and average drop rate, and then plot them out.

# Result of Experiment 1

In the experiment one, 4 kinds of TCP variants, Tahoe, Reno, NewReno and Vegas have been tested under the influence of various load conditions. Different variants will utilize different algorithms to realize congestion control. In the Tahoe, it will use slow start, congestion avoidance and fast retransmit to implement congestion control[1]. The fast retransmit algorithm will infer a packet is dropped and retransmit the packet without waiting for a retransmission timer to expire after receiving a small number of duplicate ACKs for the same TCP segments. The Reno TCP implementation retained the enhancements incorporated into Tahoe, but modified the Fast Retransmit operation to include Fast Recovery[1]. For example, if three duplicated ACKs are received, Reno is going to halve the cwnd, halve the slow start threshold, execute fast retransmit and perform fast recovery. Reno's fast recovery algorithm is optimized when a single packet is dropped from a window of data, since the Reno sender will retransmit at most one dropped packet per RTT. Reno improves the behavior of TCP Tahoe when a single packet is dropped, but it will encounter problems when multiple packets are dropped. NewReno algorithm eliminates Reno's wait for a retransmit timer when multiple packets are lost from a window[1]. NewReno can recover without a RTO, retransmitting one lost packet per RTT until all of the dropped packets have been retransmitted. NewReno will always remain in the fast recovery stage until all of the initiated data has been acknowledged. The TCP variant Vegas emphasizes packet delay rather than packet loss, as a signal to help determine the rate at which to send packets[2]. Vegas detects congestion at an incipient stage based on increasing RTT in the connection unlike Reno, NewReno. It should perform well when the network is very congested.

From figure 2.1, we can find that Vegas's throughput performs best than the others. Also, NewReno's throughput performance is better than Reno. And performance of Tahoe and Reno is almost the same when the CBR is greater than 3Mb. From figure 2.2, we can see that the drop rate of Vegas is the least. NewReno's drop rate is a little lesser than Reno. And Tahoe's drop rate is largest. It means that Vegas performs best on drop rate, the behind variants are NewReno, Reno and Tahoe. From figure 2.3, we can see that Vegas's latency is the least. Tahoe and NewReno is almost the same. The largest latency belongs to Reno. According the analysis, Vegas performs best in the congested network. In addition NewReno always performs better than Reno. Tahoe should be the worst variant in the congested network. All the results are totally in our expectation and conform to the characters of each TCP variant.

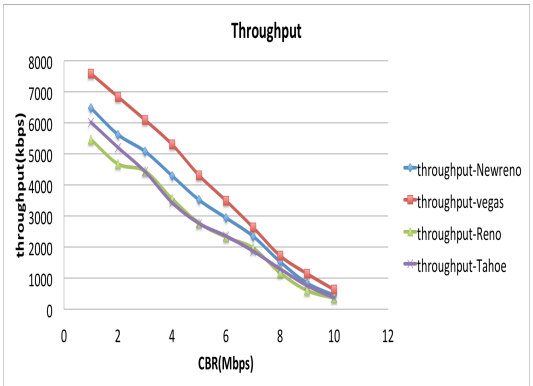


Figure 2.1 TCP variants Throughput

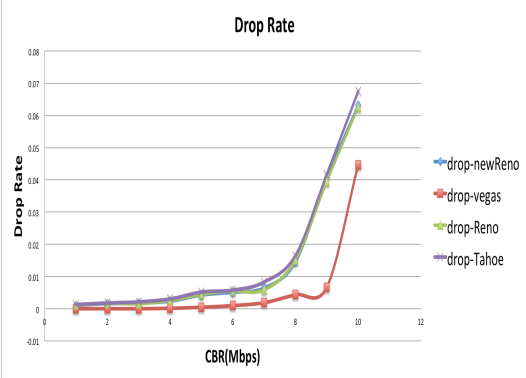


Figure 2.2 TCP variants Drop Rate

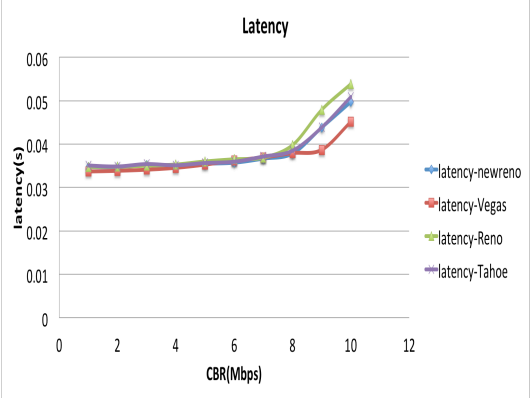


Figure 2.3 TCP variants Latency

## Result of Experiment 2

In the experiment 2, we will continue using the same topology, but we will add another TCP stream from node 5 to node 6. In order to estimate the fairness of each TCP variant, we have compared four pairs of TCP variants, and they are Reno/Reno, NewReno/Reno, Vegas/Vegas and NewReno/Vegas. We have fixed the rate of CBR from 1Mb to 10Mb, and then compared the throughput and drop rate of each TCP variant in the combination under various UDP stream.

In order to judge whether the combination of two TCP variants is fair, we will use throughput to determine that. It is clear that fair combination means two variants share the bandwidth of UDP together. From figure 3.1, the throughput of two Reno is close to each other. It means that Reno/Reno is fair. From figure 3.2 and figure 3.3, we can easily find that the throughput of NewReno is always larger than the throughput of Reno no matter which variant starts running first. It means that this combination is unfair. The reason why this happened is that NewReno will let sender eliminate Reno's wait for a retransmit timer when multiple packets are lost from a window. So NewReno will always has priority to access the bandwidth. From figure 3.4, we can find Vegas/Vegas combination is unfair to each other when CBR <= 5Mb, but they are fair to each other when CBR > 5Mb. When the traffic is low, the starting order of Vegas affects the fairness. However, when the network is very congested, the cwnd become is smaller, so the difference between two Vegas is very small. From figure 3.5 and figure 3.6, we can conclude that the combination between Vegas and NewReno is unfair. No matter Vegas starts first or latter, NewReno will access more bandwidth than Vegas under a majority of CBR. The reason is that Vegas detects congestion basing on RTT and it will detect the congestion in advance. During this stage, NewReno will begin accessing most of the bandwidth since Vegas is not accessing the bandwidth.

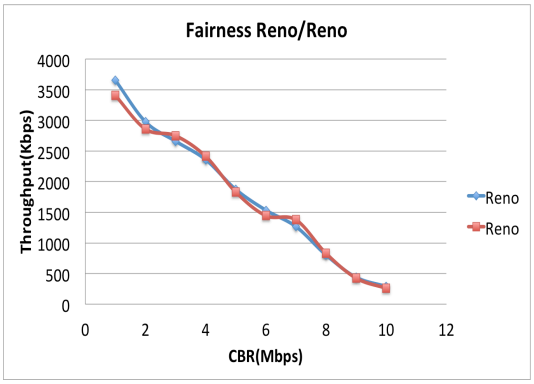


Figure 3.1 Fairness Reno/Reno

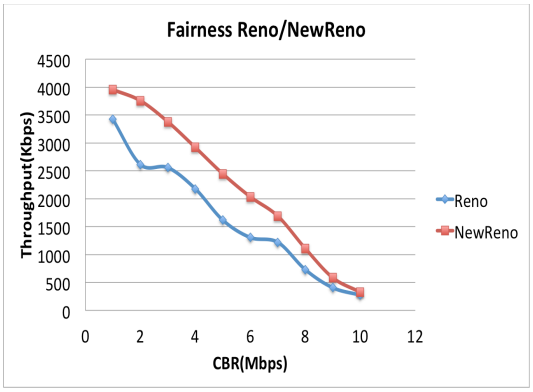


Figure 3.2 Fairness Reno

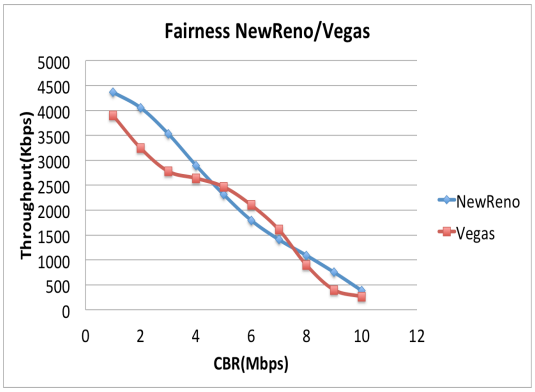


Figure 3.5 Fairness NewReno/Vegas

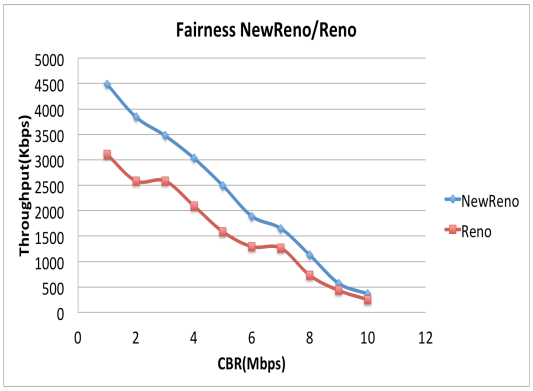


Figure 3.3 Fairness NewReno/Reno

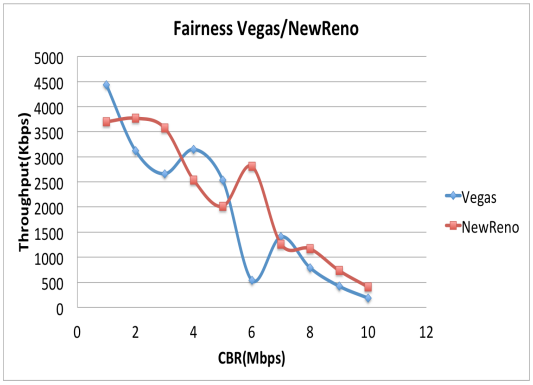


Figure 3.6 Fairness Vegas/NewReno

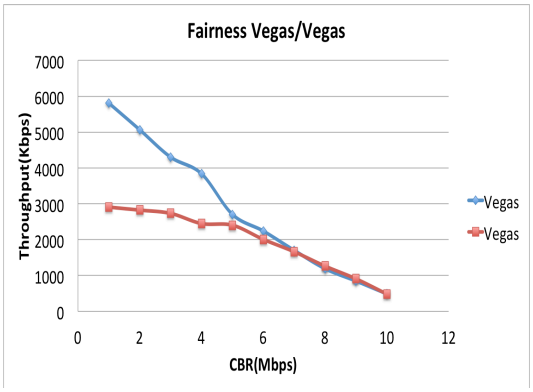


Figure 3.4 Fairness Vegas/Vegas

# Result of Experiment 3

In the experiment 3, we have simulated how packets in the queue are treated under two different disciplines: DropTail and Random Early Drop (RED). DropTail is a passive queue management algorithm that only sets a maximum length for each queue at router. The router decide when to drop the packets[3]. In RED algorithm, dropping is based on the threshold values: minimum threshold and maximum threshold. RED monitors the average queue size avg, and checks whether it lies between some minimum threshold and maximum threshold. If it does, then arriving packet is dropped or marked with a probability, which is an increasing function of the average queue size. If avg exceed maximum, all the packet arrived will be dropped[4]. We have also compare how two different TCP variants Reno and SACK perform under various queuing algorithms.

In order to get the steady TCP flow, we choose CBR == 5Mb. From the figure 4.1, the queuing algorithm is set to be DropTail, and we can see that the throughput of both Reno and SACK performs the same when t <= 6s. In addition, the throughput goes up quickly when 1s<= t <= 2s; it keeps steady when 2s<= t <= 5s; it then goes down quickly when 5s <= t <= 6s. When 6s <= t <= 10s, they start waving, but the throughput of Reno is a little higher than the throughput of SACK. From figure 4.2, the queuing algorithm is set to be RED, and we can see that SACK arrives its summit quickly from t == 1s to t == 4s and goes down to steady status from t == 4s to t == 6s. However, the Reno arrives the summit quickly from t == 1s to t == 5s. When the t changes from 6s to 10s, the throughput of both Reno and SACK stays the changes little, but the throughput of Reno will always be a little bit higher than SACK. From figure 4.3 and figure 4.4, we now start controlling the TCP variant to compare the difference of different queuing algorithms. From figure 4.3, we can see that the Reno flow goes up sharply from 1s to 2s, then stays the summit from 2s to 5s, and goes down sharply from 5s to 6s under the condition of using DropTail as its queuing algorithm. Under the condition of using RED as queuing algorithm, the throughput of Reno goes up sharply from 1s to 5s; then it goes down sharply from 5s to 6s; after that it keeps waving from 7s to 10s. The throughput of Reno under Droptail will always be a little bit higher than the throughput of Reno under RED. From figure 4.4, we start using SACK variant rather than Reno. Under the condition of DropTail queuing algorithm, the throughput of SACK goes up from 1s to 2s; then keeps the summit from 2s to 5s; and decreases sharply from 5s to 6s. However, under the condition of using RED queuing algorithm, the throughput of the SACK goes up sharply from 1s to 4s and then goes down sharply from 4s to 6s. When CBR changes from 6s to 10s, the throughput of SACK under DropTail is always a little bit higher than the throughput of SACK under RED.

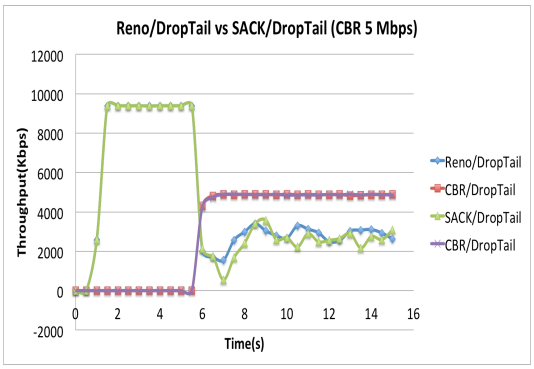

Figure 4.1 Reno/DropTail vs SACK/DropTail

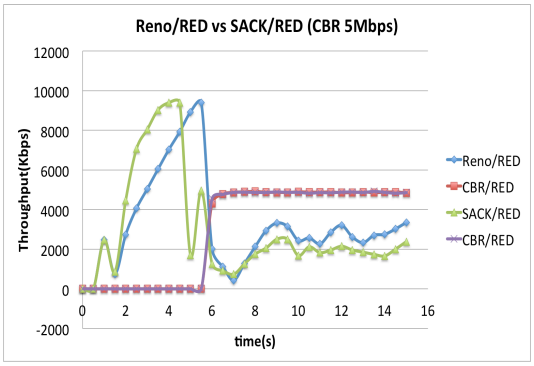

Figure 4.2 Reno/RED vs SACK/RED

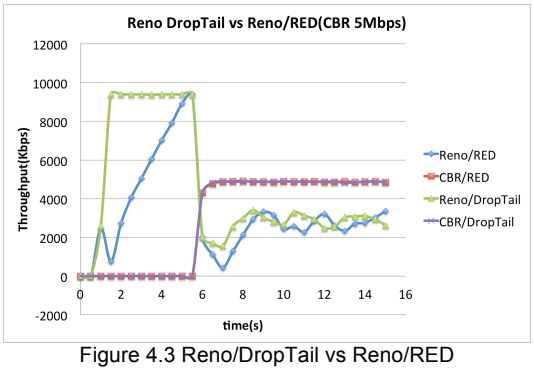

Figure 4.3 Reno/DropTail vs Reno/RED

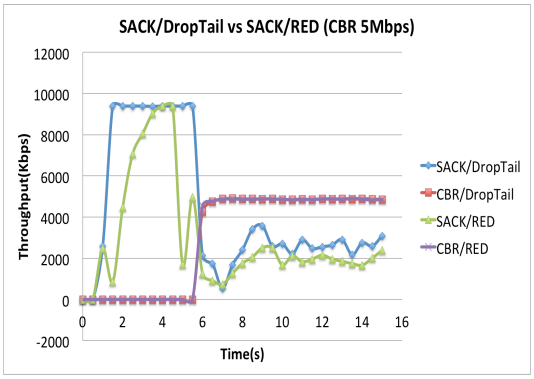

Figure 4.4 SACK/DropTail vs SACK/RED

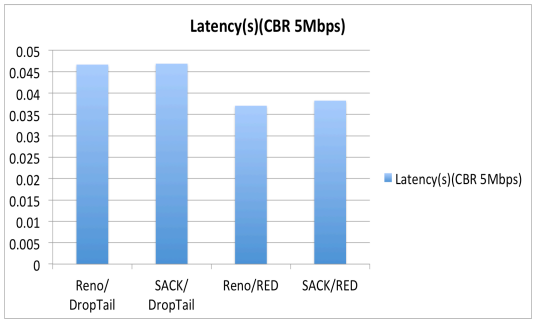

Figure 4.5 Latency Comparison

(1) According the figure 4.1, we can see that Reno and SACK are fair from 1s to 15s under the DropTail queuing algorithm. From the figure 4.2, we can also see that Reno and SACK are fair to each other from 1s to 15s under the RED queuing algorithm. It shows that each queuing discipline provides fair bandwidth to each TCP flow. (2) From figure 4.5, the latency of RED is lesser than the latency of DropTail. (3) TCP flow will start congestion control when the CBR of UDP is increasing, and the throughput of TCP will decrease. (4) From figure 4.4, we can easy see that RED is not a good idea to deal with SACK, because SACK will oscillate wildly in a RED environment.

Reno under RED does not have good performance, since Reno is used when multiple packets are dropped[5]. SACK is better, since it is used when single packet is dropped. From the figure 4.2, we can also find that SACK performs better than Reno under the same RED queuing algorithm.

Figure 4.3 and figure 4.4 demonstrate that there are oscillations in the network using RED queuing method. From the analysis of the two different queuing strategy, we can see that the reason of oscillations in the RED queuing strategy is that when the queue size is between the th_min and th_max, RED will drop the packets by probability to control the queue size which leads to the oscillation of throughput. This property leads to one of our conclusion that in a network required stable throughput or bandwidth, RED is not a good queuing strategy, especially when the th_min and th_max isn't well defined. However, we cannot conclude that RED is worse than DropTail. Although the queue size changing property leads to the oscillation in the network, it lowers the latency in the network. Like we have analyzed, Latency is well controlled using RED strategy. Because when the congestion appears in the network, RED will decrease the window size which decrease the number of packets the end host will send.

Therefore, we cannot conclude which queuing strategy is better. It depends on the network situation or what the goal we want to achieve – to have lower latency or to have stable throughput.

## Conclusion

In this paper, we have done some experiments to analyze the performance of different TCP variants. Different TCP variant will perform differently according to the congestion status of the network. Vegas variant performs best among all the tested variants. The combination of two TCP variants is not ideally fair all the time. The combination of Reno and Reno is the fair, but the combination of NewReno/Reno, Vegas/Vegas and NewReno/Vegas is not fair. DropTail queuing algorithm is fairer than the RED. Every TCP Variant has its own strengths and weaknesses, which means that we have to implement them according to different circumstances.

## References

[1] Simulation-based Comparisons of Tahoe, Reno and SACK TCP[J], Kevin Fall, Sally Floyd
[2] http://en.wikipedia.org/wiki/TCP_Vegas
[3] Comparison and Analysis of Drop Tail and RED Queuing Methodology in PIM-DM Multicasting Network[J], Ashish Kumar, Ajay K Sharma, Arum Singh, B R Amebedkar
[4] NS Simulator for Beginners, Lecture notes
[5] Performance Analysis of RED With Different TCP Congestion Control Mechanism, Esha Kumar, Shivanka Chugh, S.P Ghrer