Performance Analysis of TCP Variants

Jiahao Song and Chen Gu

Khoury College of Computer Sciences, Northeastern University

song.jiaha@northeastern.edu, gu.che@northeastern.edu

Abstract

Transmission Control Protocol is a connection-oriented internet protocol that provides reliable, ordered, and error-checked delivery of byte streams between end-to-end applications. Congestion Control is one of TCP's most significant features and it is implemented divergently among all TCP variants. In this study, we aim to compare different TCP variants (Tahoe, Reno, NewReno, Vegas, and SACK) using the NS-2 network simulator to perform experiments that help us understand the behavior of the TCP variants under various load conditions and queuing algorithms.

Introduction

A total of three experiments will be conducted and each of them targets a specific problem. The first experiment evaluates TCP performance under congestion. The second experiment compares the fairness between TCP variants with a similar setup of the first experiment except that we add an extra TCP flow. The third experiment guides us to understand the influence of queuing disciplines such as DropTail and Random Early Drop (RED) on TCP SACK and Reno. The various TCP variants referred to in this paper are described below:

1. **TCP Tahoe**
   TCP Tahoe implements Slow-Start, Congestion Avoidance, and Fast Retransmit. TCP is in Slow-Start when *cwnd*(congestion window) is less than *ssthresh*(slow start thresh). For every ACK the sender receives, *cwnd* increases by 1. It is an exponential growth because *cwnd* doubles its size after a period of RTT(Round Trip Time). When *cwnd* surpasses *ssthresh*, TCP enters Congestion Avoidance, and *cwnd* increases linearly. Fast Retransmit enables the sender to notice that a packet has been lost after receiving a small number of duplicate acknowledgments for the same TCP segment (duplicate ACKs). It then retransmits the packet without waiting for a retransmission timer to expire [1]. Then it goes back to Slow-Start. This increases channel utilization and throughput.
2. **TCP Reno**
   TCP Reno added Fast Recovery to Fast Retransmit. Fast Recovery takes place when a TCP sender receives an initial threshold of duplicate ACKs. Fast Recovery assumes each duplicated ACK received represents a single packet that has left the communication path [1]. Instead of Slow-Start, the sender retransmits one packet and reduces its congestion window by one half, entering congestion avoidance.
3. **TCP NewReno**
   In New-Reno, partial ACKs do not take TCP out of Fast Recovery. Instead, partial ACKs received during Fast Recovery indicate that the packet immediately following the acknowledged packet in the sequence space has been lost which should be retransmitted. Thus, when multiple packets are lost from a single window of data, New-Reno can recover without a retransmission timeout, retransmitting one lost packet per round-trip time until all the lost packets from that window have been retransmitted [1].
4. **TCP Vegas**
   A Vegas source anticipates the onset of congestion by monitoring the difference between the rate it is expecting to see and the rate it is realizing. Vegas' strategy is to adjust the source's sending rate (congestion window) to keep a small number of packets buffered in the routers along the transmission path [2].
5. **TCP SACK**
   During Fast Recovery, SACK maintains a variable called pipe that represents the estimated number of packets that haven't been acknowledged in the path. The sender only sends new or retransmitted data when the estimated number of packets in the path is smaller than the congestion window. The variable pipe is incremented by one when the sender either sends a new packet or retransmits a previous packet. It is decremented by one when the sender receives a duplicate ACK packet with a SACK option reporting that new data has been received at the receiver [1].

Methodology

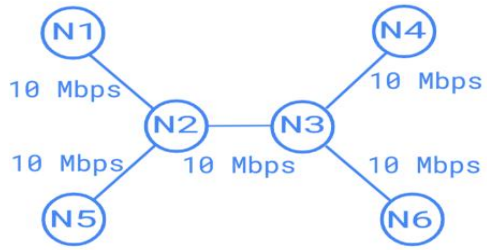


Fig 1: network topology

Terminology

1. **Throughput**
   Throughput is the quantity of data being sent and received within a unit of time. We calculate throughput using this formula:
   Throughput = Total Packet Received Size / Total time
2. **Drop rate**
   Drop rate is the number of dropped packets divided by the number of sent packets. We calculate drop rate using this formula:
   Drop Rate = (Total Packet Sent Size - Total Packet Received Size) / Total Packet Sent Size
3. **Latency**
   Latency is measured as the round-trip delay time in our study, meaning that the one-way latency from source to destination plus the one-way latency from the destination back to the source. We use the sequence number to keep track of a packet that is sent and gets acknowledged. We calculate latency using this formula:
   Latency = Total Round Trip Time / Total Packet Sent

In Experiment 1, we aim to analyze the performance of TCP variants (Tahoe, Reno, NewReno, and Vegas) under the influence of various load conditions. We first set up a network topology (Fig 1) and set the bandwidth between each node to 10Mbps. Then a TCP flow was started from N1 and was supposed to sink at N4. Then we introduced a CBR (Constant Bit Rate) flow to cause congestion. It started at N2 and sank at N3. In this scenario, CBR played the role of an unresponsive UDP flow: it used a specific amount of bandwidth, it did not care about dropped packets, and it did not perform congestion control. We started the CBR flow at a rate of 1 Mbps and recorded the performance of the TCP flow. Then gradually increase CBR flow's rate until 10Mbps which occupied the entire bandwidth. We set the congestion window size to 100 for this and following experiments. We observed and recorded the corresponding changes in TCP's throughput, drop rate, and latency. For each CBR rate, we did multiple experiments, we then aggregated results per CBR rate.

In Experiment 2, we started three flows: one CBR, and two TCP. We added a CBR source at N2 and a sink at N3, then added two TCP streams from N1 to N4 and N5 to N6. With a similar setting to Experiment 1, we also need to modify CBR's flow rate. This time we increase the flow rate up to 15Mbps and let the experiment run for 15 seconds. We would like to explore what happens after CBR's rate reaches its bottleneck capacity. We calculated the throughput, drop rate and latency for each TCP flows in the network.

In Experiment 3, we studied the influence of the queuing discipline used by nodes on the overall throughput of flows. We added one TCP Reno flow (N1 to N4) and one CBR (N5 to N6, 5Mbps) flow. TCP Reno was replaced by TCP SACK and we conducted the experiment two times for different TCP variants. The whole procedure goes as follows: First, start the TCP flow. After 2 seconds, the TCP flow was steady, start the CBR source, and analyze how the TCP and CBR flows change under the different queuing algorithms (DropTail and RED). We calculate the throughput and latency of both TCP streams and CBR as a function of time.

Tools:

We used Python 3.7 to write custom scripts to help us parse the data from numerous trace files. In order to understand the trace files, we referred to the NS2 tutorial provided by our Instructor David Choffnes. We used the NS-2 package installed on the CCIS Linux machines to run all the simulations and Microsoft Excel to plot the graphs.

Results for Experiment 1

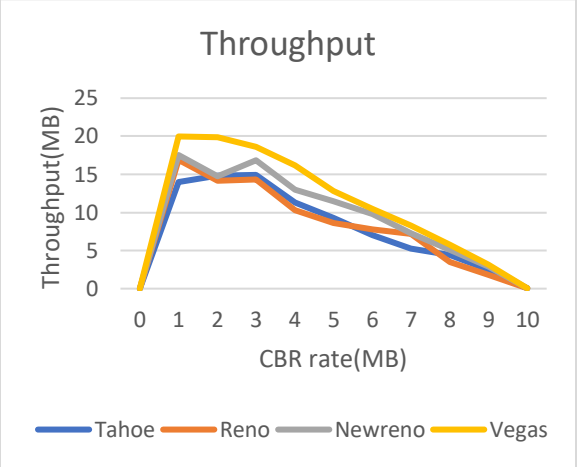


Fig 2: TCP variants throughput

Fig 2 shows that the throughput for all TCP variants tends to decrease as CBR flow rate increases from 1 to 10. The CBR does not impact TCP throughput heavily at the beginning of the experiment as we can see that when CBR rate changes from 0 to 1 Mbps, the throughput is still climbing. Among all 4 variants, Vegas has an overall better average throughput than others. The reason for this is that TCP Vegas enhances the congestion avoidance algorithm of TCP Reno. In other words, TCP Vegas dynamically increases or decreases its sending window size according to observed RTTs (Round Trip Times) of sending packets[3], whereas TCP Tahoe/Reno continues increasing its window size until packet loss is detected. We also calculated the standard deviation for these four variants and found out that Vegas has the highest stddev which is 7.98. It indicates that Vegas's performance is not very consistent with its average throughput.

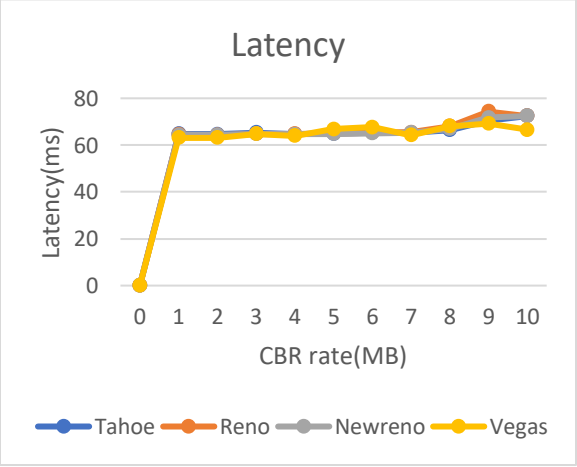


Fig 3: TCP variants latency

Fig 3 shows that the latency for all TCP variants tends to increase as CBR flow rate increases from 1 to 10. Although latency for Vegas becomes the highest among all the variants when CBR rate hits 5 and 6, it maintains an overall better performance than others. Vegas' ability to predict congestion makes it stand out. The stddev for Vegas is 19.93 which is the lowest among the four variants, meaning that it is more consistent in latency.

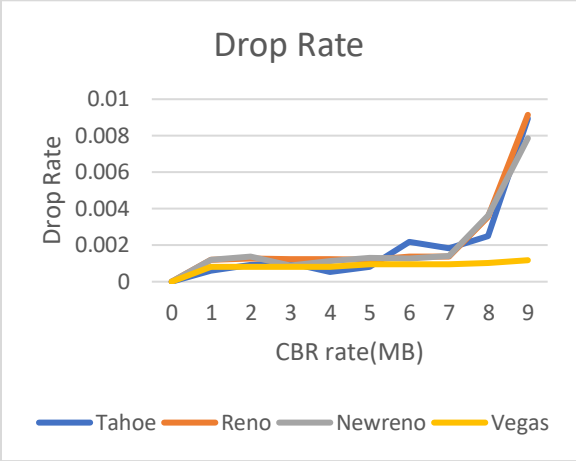


Fig 4: TCP variants drop rate

Fig 4 shows that the drop rate for TCP Tahoe, TCP Reno, and TCP NewReno reaches their peaks when CBR flow rate is at 8. However, TCP Vegas is not heavily affected and its drop rate remains fairly low even when CBR flow rate increases. The stddev for Vegas is 0.00032 which is the lowest among the four variants, meaning that it has an overall most consistent performance in drop rate.

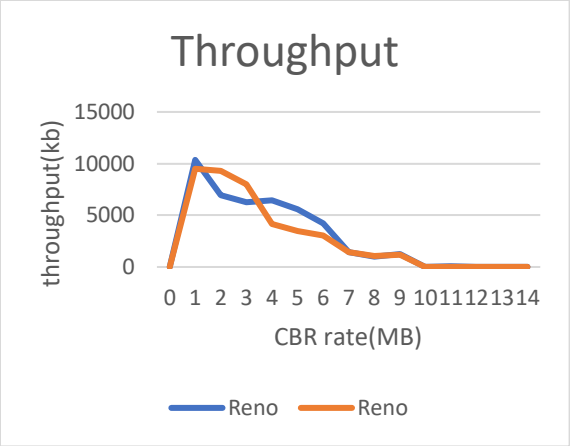
We may conclude that TCP Vegas is the overall best TCP variant in Experiment 1. Although Vegas is not the most consistent variant when it comes to average throughput, its performance remains strong across all three fields we proposed to measure.

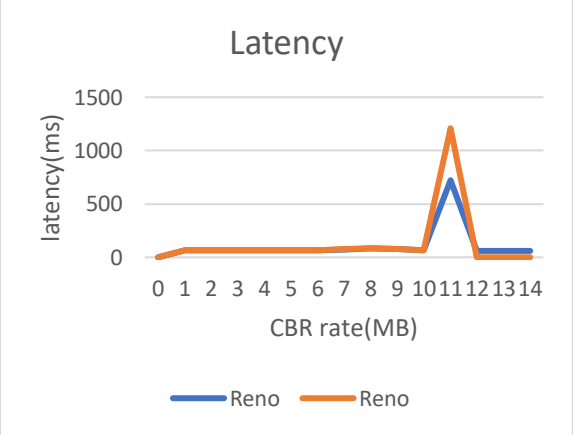
Results for Experiment 2

In this experiment we aim to analyze the fairness between different TCP variants:

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

We increased the max CBR flow rate to 15Mbps and stopped it after 15 seconds of starting to observe the fairness between the same TCP pairs and different TCP pairs.





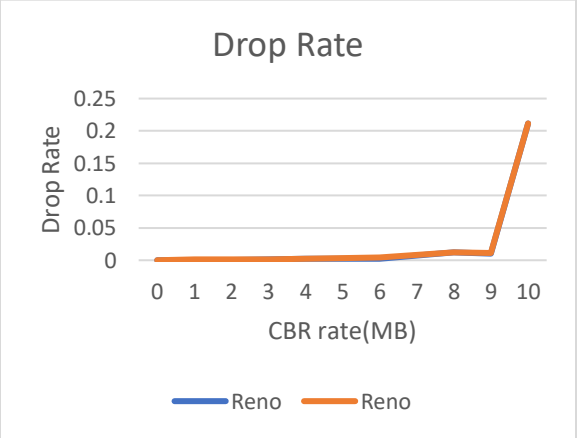
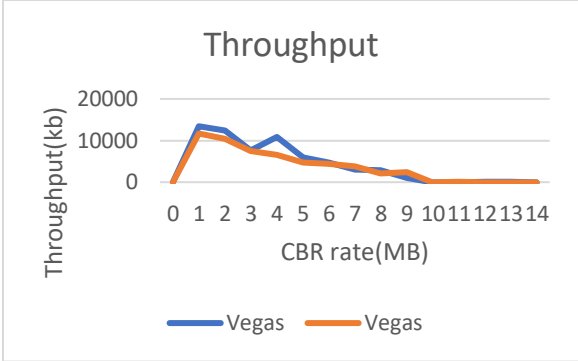


Fig 5: Reno/Reno three measures

Fig 5 shows that differences still exist but not very crucial among the Reno/Reno pair. We notice that the Reno/Reno pairs are almost identical in latency and drop rate. The two TCP flows exhibit similar performances in throughput. We may conclude that Reno/Reno is fair.



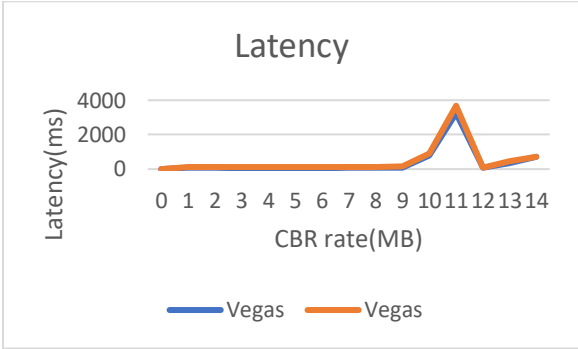
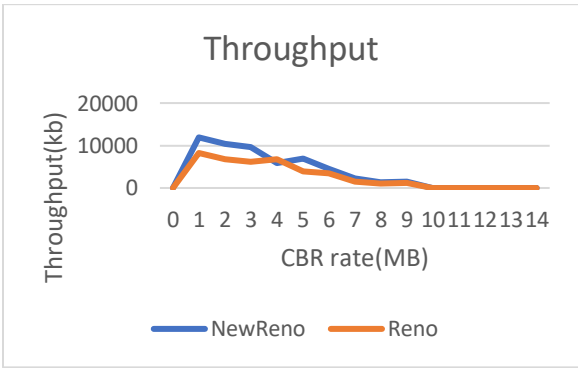


Fig 6: Vegas/Vegas three measures

Fig 6 shows that Vegas/Vegas pairs are almost identical in latency and they have similar performance in throughput. We may conclude that Vegas/Vegas pair is fair as well. Therefore TCP flows with the same TCP variant have high fairness in Experiment 2.



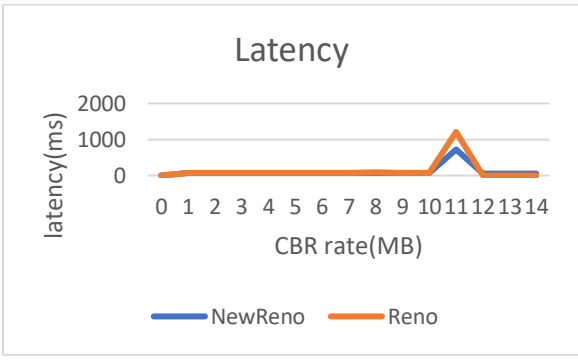


Fig 7: NewReno/Reno three measures

Fig 7 shows that in NewReno/Reno pair, NewReno has more throughput and less latency. Here we may discuss how NewReno modifies Reno's algorithm to become a better solution. NewReno can detect multiple packet losses. Like Reno, NewReno also enters into fast-retransmit when it receives multiple duplicate packets, however, it differs from Reno in that it doesn't exit Fast Recovery until all the data which was sent out at the time it entered Fast Recovery is acknowledged. Therefore it overcomes the problem faced by Reno of reducing the *cwnd* multiple times.

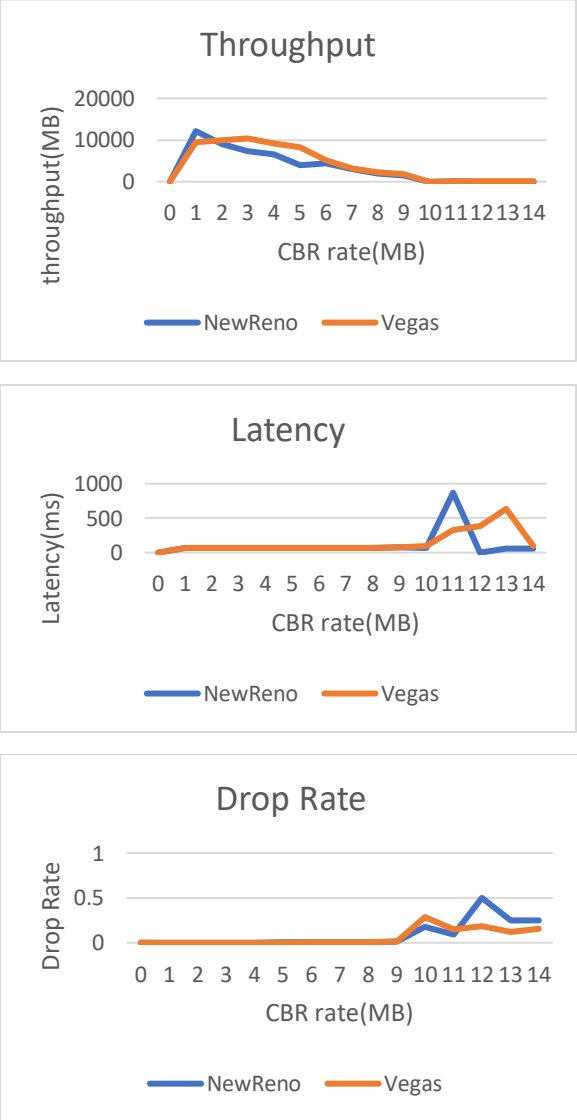


Fig 8: NewReno/Vegas three measures

Fig 8 shows that Vegas has more throughput than NewReno when CBR flow rate increases from 2 to 6, after that, their throughput are nearly the same. when CBR flow rate hits 11, latency for NewReno changes drastically. The response from Vegas is relatively smooth. Vegas also has less drop rate when CBR flow rate exceeds 11. We do see a drop rate spike when CBR flow rate is increasing from 9 to 10, but after that, the drop rate for Vegas decreases. On the opposite side, NewReno's drop rate spikes when the rate increases from 11 to 13. Vegas' capability of anticipating congestion at an early stage based on increasing RTT values of the packets in the connection makes it stand out in gaining more throughput and less drop rate compared to NewReno.

Overall, among the four pairs of TCP variants, Reno/Reno and Vegas/Vegas are fair to each other while NewReno/Reno and NewReno/Vegas are unfair to each other.

Results for Experiment 3

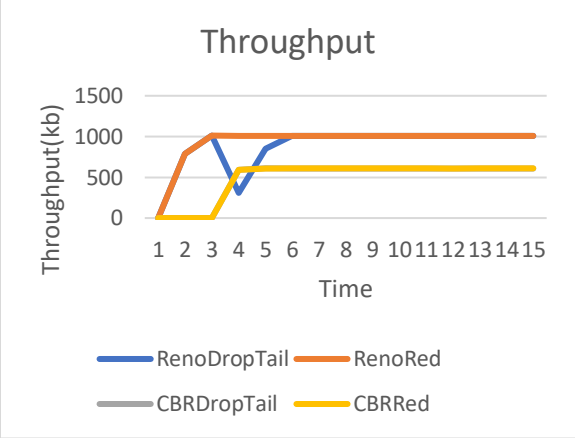


Fig 9: Reno throughput with two queuing algo

Fig 9 shows that only RED provides fair bandwidth to Reno. DropTail causes a significant decrease in throughput for Reno when CBR flow was first introduced 3 seconds after TCP flow starts.

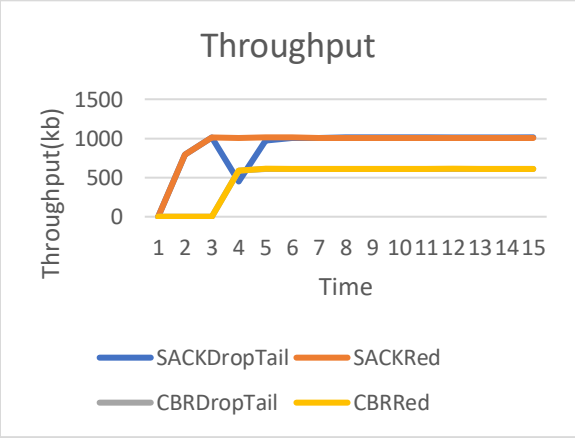


Fig 10: SACK throughput with two queuing algo

Fig 10 shows that only RED provides fair bandwidth to SACK. DropTail causes a significant decrease in throughput for Reno when CBR flow was first introduced 3 seconds after TCP flow starts.

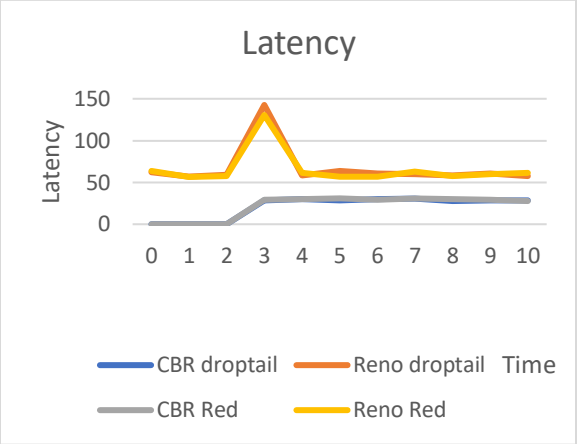


Fig 11: Reno latency with two queuing algo

Fig 11 shows that Reno in both queuing algorithms demonstrates a similar level of latency when CBR flow was first introduced at 2 seconds after TCP flow starts.

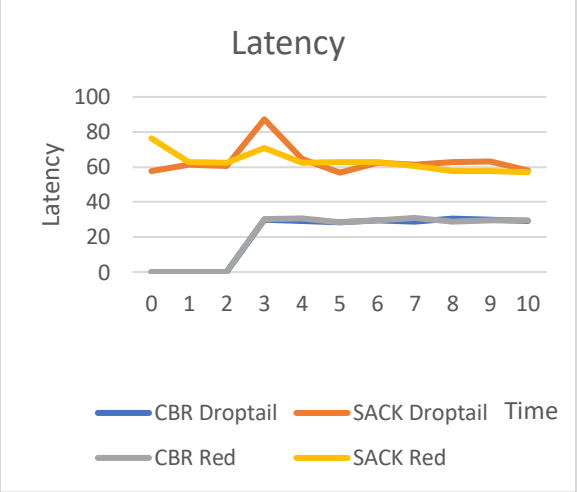


Fig 12: SACK latency with two queuing algo

Fig 12 shows that SACK in DropTail demonstrates a higher level of latency than in RED when CBR flow was introduced at 2 seconds.

Overall, we may conclude that the queuing algorithm of RED has a better performance than DropTail in Experiment 3 as it has more throughput and less latency. RED is a good idea while dealing with SACK.

Conclusion

In the above experiments we used the NS-2 network simulator to gain a better knowledge of different TCP variants, followings are our observations:

1. From Experiment 1 we conclude that TCP Vegas performs better in a congested network with the highest throughput, above-average level of latency, and lowest drop rate.
2. From Experiment 2, we conclude that TCP pairs with the same variants are fair to each other, and pairs with different variants are unfair to each other.
3. From Experiment 3, we conclude that RED is overall a better queuing algorithm compared to DropTail.

We realize that our experiment is limited and rather simple because not so many variables were introduced. TCP and UDP collision in the real-world will be much more complicated than what we did in these experiments. However, we do understand different TCP variants better now and we witnessed the evolution of TCP algorithm in dealing with congestions. For future studies, we will explore more complicated network topology and advanced TCP variants.

References

[1] [FF96] K. Fall, and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", July 1996. URL "https://david.choffnes.com/classes/cs4700fa20/papers/tcp-sim.pdf"

[2] S. Low, L. Peterson and L. Wang, "Understanding TCP Vegas: Theory and Practice", URL "https://www.cs.princeton.edu/research/techreps/TR-616-00", Jan. 2000.

[3] L.S.Brakmo, L.L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", URL https://cseweb.ucsd.edu/classes/wi01/cse222/papers/brakmo-vegas-jsac95.pdf

[4] Mo, J., La, R., Anantharam, V., and Walrand, J. 1999. Analysis and comparison of TCP Reno and Vegas. In Proceedings of IEEE Infocom (Mar.). URL "https://people.eecs.berkeley.edu/~wlr/Papers/AnantharamLaMoWalrand.pdf"