# Performance Analysis of TCP Variants

Preethi Ayyamperumal
NUID: 001791108

Swati Soni
NUID: 001610740

## I. Introduction

In this paper we describe the results of our simulated experiments performed as a part of an academic project. This purpose of the project is to analyze the performance of TCP variants such as TCP Tahoe, Reno, Vegas and SACK under various load conditions and queuing algorithms. NS-2 network simulator has been used in order to perform various experiments on TCP variants and analyze their resulting behavior.

The original design of the Transmission Control Protocol (TCP) worked reliably, but was unable to provide acceptable performance in large and congested networks. Several TCP variants have been proposed since then with mechanisms to control and avoid congestion. We have performed three different experiments in order to analyze different behavior of various TCP variants by reporting things like average bandwidth, packet drops, and throughput over time, end-to-end latency (average and over time), sequence numbers of packets sent, and the congestion window of all TCP flows. The results have been depicted through various graphs.

## II. Methodology

In order to conduct these experiments, the following steps were followed:

### *Setting Up Environment*

The NS2 network simulator was installed on Ubuntu.

### *Tools Used*

- **NS2:** NS2 Network Simulator is used to create network topologies and conduct experiments over TCP variants as mentioned in the requirements of the project.
- **Python:** As advised in the requirement document of the project Python language is used in order to parse the data in output trace files of network simulator and obtain various results as per the requirement of the experiments.
- **Editor:** Text editor was used in Ubuntu to write different ns2 and python scripts.
- **Microsoft Excel:** MS Excel is used to generate graphs based on the results obtained through python scripts.

### *Simulation Parameters*

The following values are used during the experiments:

1. Network Topology:
   The following topology has been used for the experiments, and the bandwidth of each link is set to 10Mbps as mentioned in the requirements.

```
N1                          N4
  \                        /
   \                      /
    N2__________________N3
   /                      \
  /                        \
N5                          N6
```

2. Queuing disciplines: DropTail and Random Early Drop algorithms are used to perform the experiments.
3. CBR bandwidth in Queuing influence experiment: 10 mb
4. CBR start time: The CBR has been started at the fourth second in Queuing experiment in order to give time for TCP flow to become steady and get significant notable changes in TCP flow on creation of CBR flow.
5. CBR bandwidth range in Congestion Control and Fairness experiment has been varied from 1 mb to 10 mb
6. DropTail has been used as queuing algorithm for Congestion Control and Fairness experiment
7. Default TCP and CBR packet size is 1000 bytes

## III. Experiment 1: TCP Performance Under Congestion

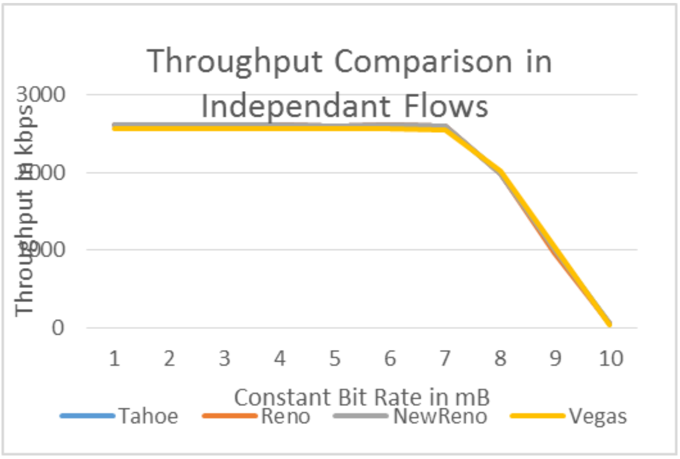
In this experiment, the performance of TCP variants (Tahoe, Reno, NewReno, and Vegas) has been analyzed under the influence of various load conditions. Constant Bit Rate flow has been used to introduce load. The results obtained upon analysis of throughput, packet drop rate, and latency of the TCP stream as a function of bandwidth used by the CBR flow are as follows:

*Average throughput:*

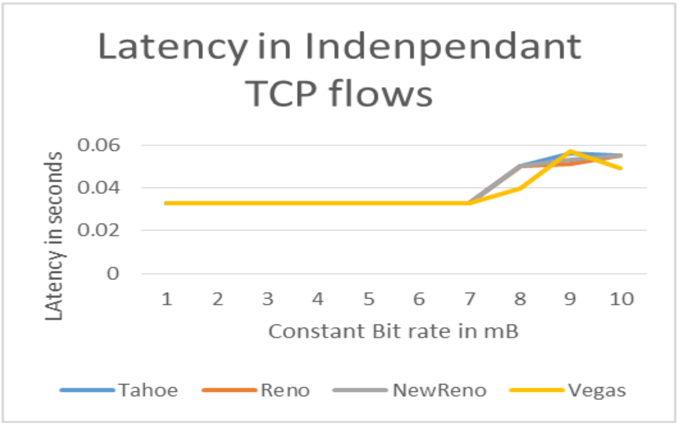
TCP Vegas:
TCP Tahoe, Reno and NewReno have comparably high overall average throughput than TCP Vegas. TCP Vegas have higher throughput than other variants only after bottleneck bandwidth is reached because of the fewer drops than others because of its delay based window.

TCP Tahoe, Reno and NewReno:
Before bottleneck bandwidth is reached when there are no packet loss, TCP Tahoe, Reno and NewReno has same average throughput since it does not enter slow start. Their congestion window oscillates at its optimum size and since there are no retransmission timeout,

Multiple packet Loss and TCP NewReno:
On multiple packet losses Reno's throughput is poorer since it uses one RTT for each dropped packet.
Also it is possible that the congestion window is reduced twice for packet losses occurred in one window. New Reno performs better in this case since it doesn't exit fast-recovery until all the packets which was sent at the time it entered fast recovery is acknowledged.



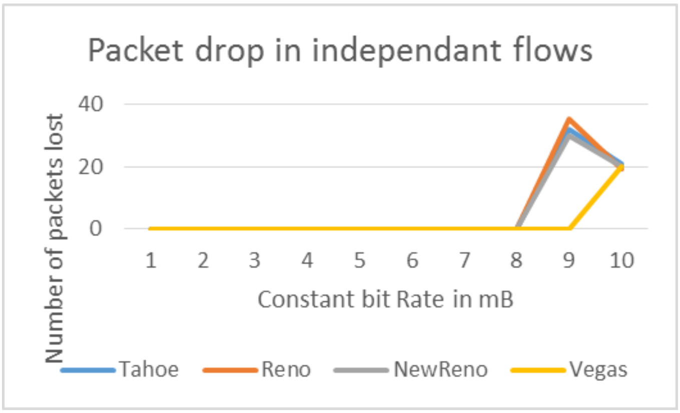
.



*Average latency:*

In End to End latency experiment, TCP Vegas has the lowest average latency because of its efficient congestion avoidance using delay based window by estimating RTT.
Tahoe has the highest latency since for each packet loss it enters slow start thus considerably increasing the end to end latency.
Reno and NewReno has average latency lower than Tahoe since it enters Fast Retransmit and Fast recovery thus reducing the end to end latency time comparatively

*Packet Drops:*
Vegas has the fewest drops in bottleneck bandwidth and it performs significantly better than other variants. This might be due to carefully extracting congestion information from observed RTT and intelligently reacting to it, Vegas avoids the unending cycle of sinking into congestion and recovering from it.
The TCP Variants Tahoe Reno and NewReno have similar loss because all the three algorithms increases its congestion window till it notices a timeout. All the three forces packet drops in a same way.

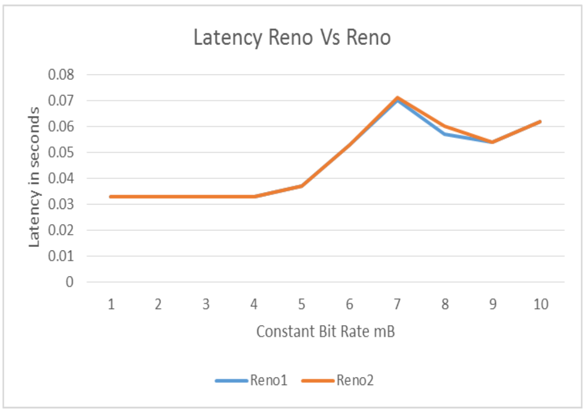


*Overall "best" TCP variant:*
In this experiment with independent TCP flows, NewReno seems to be better in terms of throughput compared to Tahoe, Reno and Vegas. Vegas seems to be perform better in terms of latency and packet loss compared to Tahoe, Reno and NewReno.Vegas Throughput is better than Tahoe, Reno and NewReno if there is a bottleneck bandwidth.   So there is no "best" variant identified. But circumstances like available bandwidth determines the performance of each TCP variant.

## IV. Experiment 2: Fairness Between TCP Variants

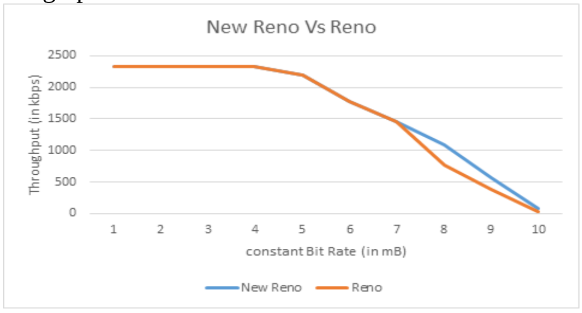
This experiment has been conducted to analyze the fairness between different TCP variants. For this, three flows has been started: one CBR (N2-N3), and two TCP(N1-N4 and N5-N6). The following results are obtained for different TCP pairs upon analysis of the average throughput, packet loss rate and latency of each flow as a function of the bandwidth used by the CBR flow:

*Fairness in Flows*
Reno is fair to other Reno flow as the Reno algorithm increases its congestion window size proportionately till it notices packet loss and the do the multiplicative decrease in the Congestion window and after that both flows enters into fast retransmit.
Vegas is fair to other Vegas flow since both the flows maintain delay based window and perform congestion avoidance in the same way. Hence they don't dominate each other.

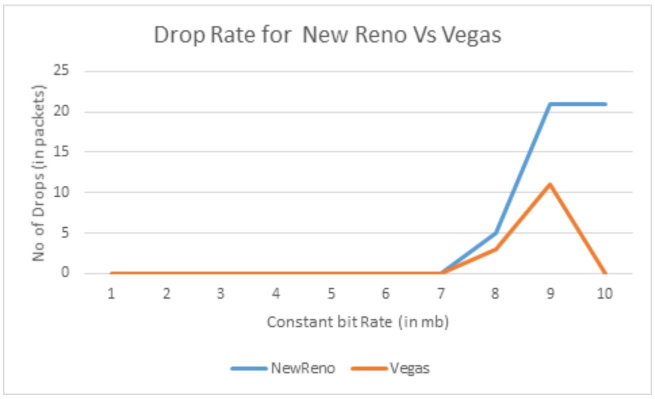


*Unfair Flows and Reasons for Unfairness*

- NewReno is fairer to Reno only if the bandwidth is sufficiently high. In case of bottleneck bandwidth, NewReno does not deflate the window ,in order to exit fast recovery because of ‘partial ACKs’ whereas Reno deflates the window till it receives an ACK for the highest sequence number sent. Thus NewReno consistently uses more bandwidth than Reno. This leads to fewer packet drops and slight lower average end to end for NewReno compared to Reno shown in the graph below.



- NewReno is never fair to Vegas as the performance of Vegas deteriorates because Vegas reduces its sending rate before NewReno, as it detects the congestion in advance using its delay based congestion window and hence provides more bandwidth to co-flow TCP NewReno. NewReno has improved fast retransmit so it naturally occupies more bandwidth than Vegas. Thus NewReno is responsible for increase in end to end average latency but Vegas has fewer packet loss as it detects congestion using its delay based congestion window as shown in below figure.



## V. Experiment 3: Influence of Queuing

In this experiment, the influence of queuing disciplines like DropTail and Random Early Drop (RED) used by nodes on the overall throughput of flows has been observed. This experiment has been performed with TCP Reno and SACK.

One TCP flow (N1-N4) and one CBR/UDP (N5-N6) flow has been created in the topology for this experiment. First, the TCP flow has been started. Once the TCP flow gets steady, the CBR source starts. Based on the analysis done on how the TCP and CBR flows change under different queuing disciplines, the following results has been obtained:

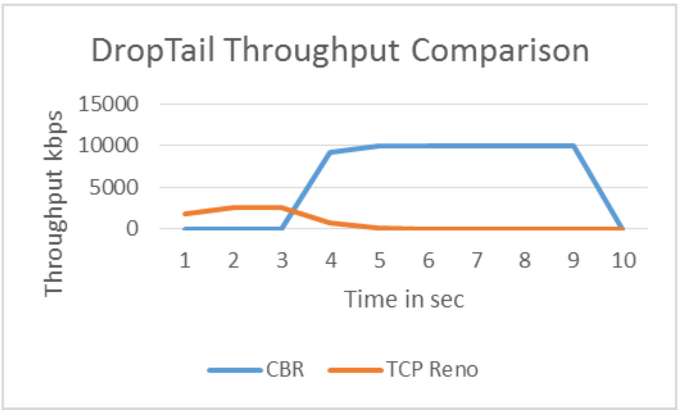
*Impact of queuing discipline on fairness between flows*
DropTail queuing mechanism does not provide fair bandwidth to each TCP and CBR/UDP flow for both TCP Reno and TCP SACK. While, RED queuing discipline does provide fair bandwidth to each flow for both TCP Reno and TCP SACK.

*End-to-end latency for the flows*
There is no significant change in the end-to-end latency of the TCP and CBR flows between DropTail and RED. It almost remains the same.

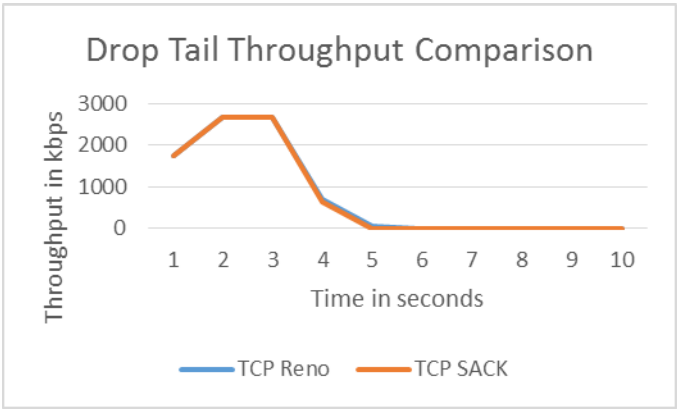
*Reaction of TCP flow to the creation of the CBR flow*
On the creation of the CBR flow, the TCP flow’s throughput decreases with time. Different results have been observed for the TCP flow in TCP Reno and SACK which are as follows:

*TCP Reno and Droptail:* The throughput of TCP flow keeps on decreasing over time and after two seconds of creation of the CBR flow, the throughput becomes 0.
Drop-Tail mechanism when performed on Reno works well until the start of UDP flow. The throughput is maximum over time in TCP Reno working on Drop-Tail mechanism as compared to the other variants and mechanisms.
With the start of UDP flow, the throughput keeps on decreasing with time.
In Drop-Tail mechanism each packet is treated identically and when queue filled to its maximum capacity the newly

incoming packets are dropped until queue have sufficient space to accept incoming traffic. When a queue is filled the router start to discard all extra packets thus dropping the tail of mechanism. The loss of packets causes the sender to enter slow start which decreases the throughput and thus increases its congestion window.

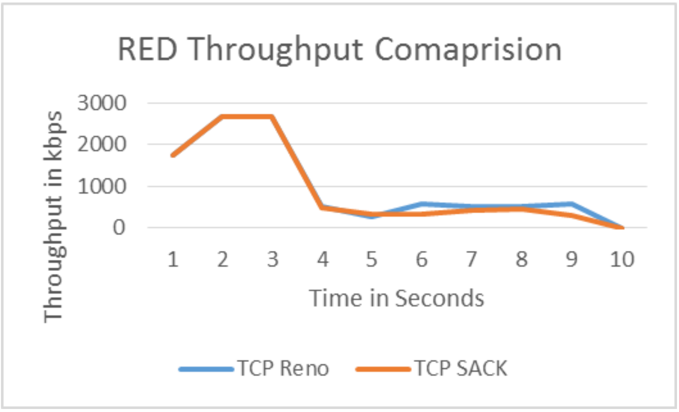


*TCP SACK and* Droptail: The throughput of the TCP flow keeps on decreasing over time and after two seconds of creation of the CBR flow, the throughput becomes 0.
Drop-Tail mechanism works in the same way for TCP SACK as for TCP Reno. The throughput starts decreasing with time with the start of UDP flow. The decrease in TCP SACK is more than TCP Reno because their behavior is different when multiple packets are dropped from one window of data.
As in Reno, the SACK TCP implementation enters Fast Recovery when the data sender receives initial threshold of duplicate acknowledgements. During Fast Recovery, SACK maintains a variable called pipe that represents the estimated number of packets outstanding in the path. This differs from the mechanisms in the Reno implementation.

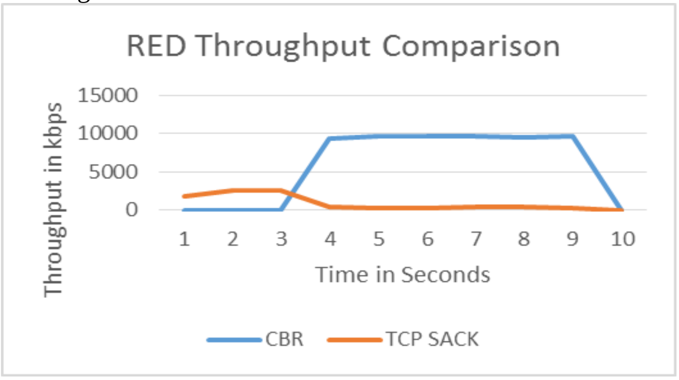


*TCP Reno and RED:* The performance of TCP flow decreases initially on the creation of the CBR flow but the throughput is preserved over time after two seconds of creation of the CBR flow. It also increases at some instants.

RED mechanism when performed on Reno produces the same results as that of Droptail mechanism performed on TCP Reno and SACK and RED performed on TCP SACK. When the UDP flow starts, initially it produces throughput less than that of Droptail mechanism on Reno but with time it performs better than Droptail mechanism.
As the queue size increase the probability for discarding a packet also increase. When buffer is full probability becomes equal to 1 and all incoming packets are dropped.
RED is capable to evade global synchronization of TCP flows, preserve high throughput as well as a low delay and attains fairness over multiple TCP connections, etc. It is the most common mechanism to stop congestive collapses.
Overall, TCP Reno works better than TCP SACK on RED mechanism.



*TCP SACK and RED:* The performance of TCP flow decreases initially on the creation of the CBR flow but the throughput is preserved over time after two seconds of creation of the CBR flow. It also increases at some instants.
TCP SACK performs the same way on RED mechanism initially as it does on DropTail mechanism. But with the start of UDP flow, initially TCP SACK works better on Droptail mechanism than on RED mechanism. As soon as the time passes, the throughput starts reducing for TCP SACK working on Droptail but the TCP SACK working on RED keeps on preserving the high throughput.
TCP SACK's performance is less than TCP Reno when both working on RED mechanism.



*Performance of SACK on using RED*

RED is a good idea while dealing with SACK because it performs better on RED mechanism than on Droptail mechanism. Though its performance is less than TCP Reno in both cases but the results are better on RED mechanism than on Droptail mechanism.

## VI. Conclusion

To conclude, following are the final observations based on the performed experiments:

- TCP NewReno performs better compared to other variants in terms of throughput under the influence of various load conditions. TCP Vegas performs better at bottleneck bandwidth.
- NewReno is unfair to Reno and Vegas. Vegas is fair to all other variants.
- RED queuing algorithm results in better performance of TCP Reno and SACK on creation of CBR flow than DropTail algorithm.

## VII. References

[1] https://sites.google.com/a/seecs.edu.pk/network-technologies-tcp-ip-suite/home/performance-analysis-of-impact-of-various-queuing-mechanisms-on-mpeg-traffic/working-mechanism-of-fq-red-sfq-drr-and-drop-tail-queues

[2] https://blackboard.neu.edu/bbcswebdav/pid-10696974-dt-content-rid-17208477_1/xid-17208477_1

[3] http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/A4-TCP-Sim/TCP-Throughput.html

[4] http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/A4-TCP-Sim/TCP-Throughput.html

[5] http://ftp.cs.princeton.edu/techreports/2000/616.pdf

[6] http://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf

[7] https://www.isoc.org/inet2000/cdproceedings/2d/2d_2.htm

[8] http://www.eecs.berkeley.edu/~fox/summaries/networks/tcp_comp.html

[9] http://nile.wpi.edu/NS/

[10] http://www.isi.edu/nsnam/ns/tutorial/index.html

[11] http://www.isi.edu/nsnam/ns/ns-documentation.html