Performance Analysis of TCP variants

Zhikai Ding, Yangyang Huang

zhikai@ccs.neu.edu, hyyearth@ccs.neu.edu

Abstract

Transmission control protocol is an Internet's transport layer, connection-oriented, reliable transport protocol that provides many important functions such as congestion control, retransmission, in-order delivery, etc. It was revised many times since it was published in September 1981, and a lot of implements are set up into TCP for providing better performance. Improved versions of TCP including in Reno, New Reno, Vegas are used to control and avoid congestion. This paper compares these TCP variants by taking simulations under many different environments like different load of network and queuing disciplines in NS-2 simulator, analyzing the reason of the result.

Keywords: TCP variants, TCP performance, congestion, fairness, queuing

1. Introduction

This paper is going to study how do TCP variants, such as Reno, New Reno, Vegas and SACK, perform against congestion. It is important to compare the different performances among TCP variants, since diverse implements will lead to different results in the same environments and various conditions of network also have different impacts on the same version of TCP, comparing the results of same TCP running under different settings and contrasting the results of different TCP variants working in same surroundings can help us to analyze the different performances of every TCP variant, and it also assists us in choosing a best version of TCP in some specific networking environment.

The TCP variants used in this paper are briefly described below.

TCP Tahoe is the original version of TCP, which does not have the fast recovery. It take a completely timeout interval to detect a packet loss, and when congestion detected, it sets its congestion window to 1 and enter the slow start phase.

TCP Reno is improved on the basis of Tahoe, which adds two more implements, fast retransmit and fast recovery. Fast retransmit regards three duplicated ACKs as a signal of packet dropping instead of waiting for time out in Tahoe and retransmit the loss packet immediately. And fast recovery allows congestion window sets to half of threshold value instead of one.

TCP New Reno fixes problem that fast retransmit cannot handle multiple packets. Sender will not enter fast recovery process until all of the loss packets have been acknowledged.

TCP Vegas sets timeout and measures RTT for every packet in transmit buffer. And it detects congestion at beginning stage so that sender can adjust throughput to avoid packet loss.

TCP SACK deals multiple packet loss by sending selective ACKs to let sender knows the lost packets.

We will use NS-2 simulator to create a condition that contains 6 nodes with the topology like figure 1, setting a constant bit rate as a UDP flows that does not care about the loss of packets to occupy some amount of bandwidth. We will value the performance of TCP variants under different congestions by getting the results of throughput, latency, packets loss rate in experiment 1, and compare the fairness of some pairs of TCP variants under the same condition in experiment 2, explaining the fairness by calculation of results. Besides that, influence of queuing is also taken into consider in the experiment 3 and we can know the fairness under different queuing disciplines, difference effects on latencies with different TCP variant, which helps us to analyze the influence.

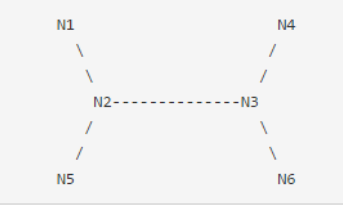

Figure 1 Topology of simulated network

2. Methodology

NS-2 is a discrete event simulator that can simulate standard experiment surroundings on both wired and wireless network, providing functions to simulate many protocols including TCP, UDP, FTP, HTTP, etc. Additionally, it uses Tcl as its script language.

The simulated network contains 6 nodes connected like in figure 1. All connections are set as full duplex links with 10Mbps bandwidth and 10ms delay. And packet size is set to 1000 Byte. Max TCP window size is set to 70 for all experiments, because if we use the default size 20, the max-throughput of TCP is limited to around 2500kbps, and we can not get enough information about TCP while CBR flow's rate changes.

Experiment 1 is for testing the performances of different TCP variants under different congestion environments. We set an unresponsive a constant bit rate source at N2 and sink at N3. At the same time, we add one type of TCP stream from N1 to a sink at N4, getting the results of throughputs, latencies, packet drop rate under the condition of different value of constant bit rate. We are going to set CBR flow from 1Mbps to 10Mbps, increasing 1Mbps each time. For every situation,

we did 10 times experiments and got their average results for this and following experiments.

In experiment 2, we set two TCP streams, one source at N1 and sink at N4, and the other one source at N5 and sink at N6. Besides that, set one UDP flow source at N2 and sink at N3. Run different pairs of TCP streams, including Reno and Reno, New Reno and Reno, Vegas and Vegas, New Reno and Vegas under different values of CBR like experiment 1. UDP will begin at 0.1s, and one TCP start its flow at 3s, the other one begin at 1s, 3s and 5s respectively, which simulate the situation that one TCP is affected by the other when this TCP start before, at the same time with and after this TCP. And we only consider the data in 8s~15s when two TCP flow are relatively steady. Then we use the same way to get throughput, latency and packet loss rate as experiment 1 and compare them.

The purpose of experiment 3 is to analyze the different influence on the performance of TCP between different queuing disciplines, Drop Tail and Random Early Drop. Set TCP stream source at N1 and sink at N4, and set UDP flow source at N5 and sink at N6. TCP stream will begin at 0.1s, and CBR flow begins at 10s to ensure TCP enters its steady phase. Acquire the result of performance of TCP under different queuing algorithms.

3. Results for Experiment 1

The goal of this experiment is to analyze and compare the performance of TCP variants (Tahoe, Reno, New Reno and Vegas). We use throughput, packet drop rate and latency to evaluate the performance by observing them as functions of bandwidth of the CBR flow. In the topology as Figure1, CBR flow from N2 to N3 and TCP flows from N1 to N4.

The throughput of TCP variants are showed in figure 3.1. To calculate the throughput, we use the following formula:

Throughput = (number of received packets) * byte per packet * bits per byte / total time / 1024.

As shown in the figure, TCP variants start with the similar high throughput. The throughput begins around 8300, because we set the window size to 70, based on the 0.06s RTT at CBR = 1mb, it's theoretical packet sending rate is 70/0.06 = 1166, and theoretical throughput is 1166*1000*8/1/1024=9109, considering we've count the TCP start time in, when the flow is not steady, real throughput should be less than that. And Vegas starts with a slightly low throughput because of its delay-based window. When CBR is greater than 5mb, the congestion gets more severe, due to the different kinds of congestion control, Vegas maintains a relatively big delay window and gets slightly higher with other TCP variants.

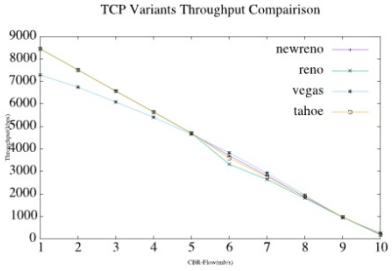

Figure 3.1 Throughput-CBR Flow

Figure 3.2 shows the results of latency. Here the latency means the time needed for a received packet sent from source to destination. We can see that all TCP variants except Vegas have the same RTT when CBR < 5mb because of the low congestion. After that, since every TCP variants have different congestion control algorithm, their latency varies. And for these three variants, Reno does better. As for Vegas, this variant's delay window has a great impact on latency since it detects congestion and changes its window earlier.

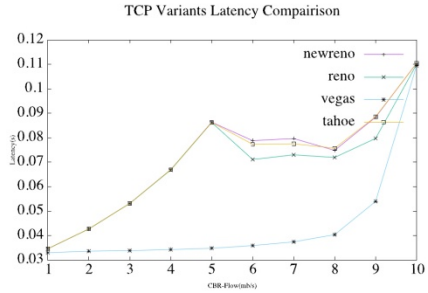

Figure 3.2 Latency-CBR Flow

Figure 3.3 shows the comparison of packet loss rate, we can see that when CBR > 5mb, Except Vegas, all TCP variants' packet loss rates are greater than 0, indicating the congestion begins. We can know from the result that Vegas < New Reno < Reno < Tahoe. Vegas, owing to its delay based congestion control algorithm, it has way less packet loss rate than others.

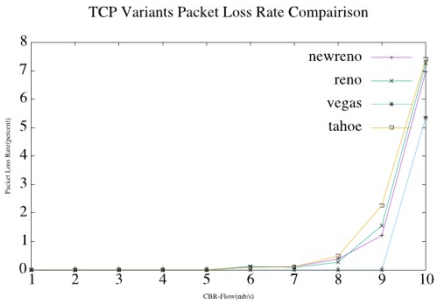

Figure 3.3 Packet Drop Rate-CBR Flow

Among the four TCP variants, Vegas is the best version of TCP according to the result of throughput, latency and packet drop rate based on the result in experiment 1. Its delay window and congestion detection at early stage help it against congestion more successfully.

4. Results for Experiment 2

In this experiment, we are going to check the fairness between different TCP variants pair under various load of network conditions.

Figure 4.1.1 and figure 4.1.2 show the throughput of pair of Reno flows when one flow starts before the other and two Reno flows begin at the same time respectively under different UDP CBR flows. And latency and packet loss rate information are shown through figure 4.1.3 and figure 4.1.4

For Reno-Reno pair, it is obviously fair as we can see from the graph that the throughputs, latencies, packet loss rates are almost same for both Reno 1 and Reno 2 no matter how value of CBR changes and whether one Reno begins before the other. It is because both of their same rate of window size increasing, same time that congestion occurs, both fast retransmit and fast recovery processes work alike with each other.

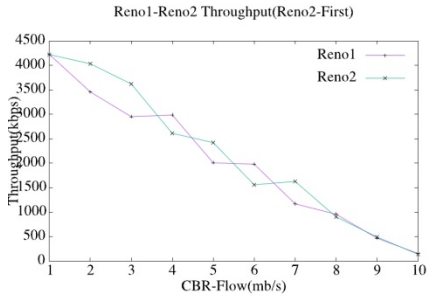

Figure 4.1.1 Reno1-Reno2 Throughput (Reno1-First)

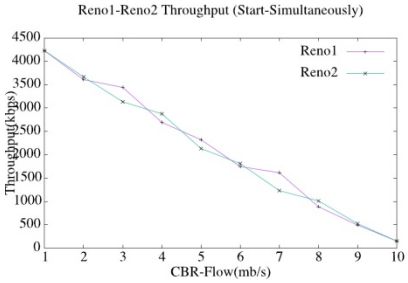

Figure 4.1.2 Reno1-Reno2 Throughput (Start-Simultaneously)

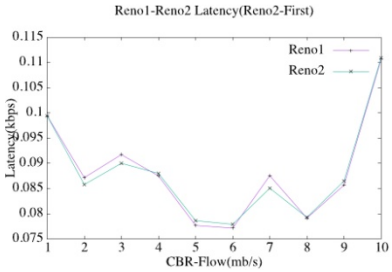

Figure 4.1.3 Reno1-Reno2 Latency (Reno1-First)

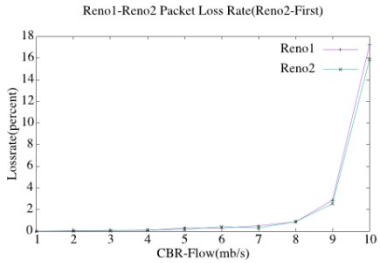

Figure 4.1.4 Reno1-Reno2 Loss Rate (Reno1-First)

For pair of New Reno and Reno, New Reno takes better performance on throughput, latency and packet loss rate than Reno no matter which TCP begin first. When multiple packets lost, Reno needs more time in fast retransmit and fast recovery, since New Reno send all the dropping packets one time once sender detect the packet loss and Reno just send one lost packet per RTT.

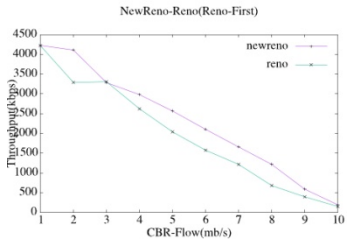

Figure 4.2.1 New Reno-Reno Throughput (New Reno-First)

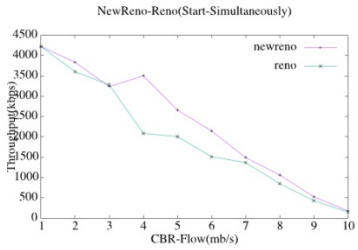

Figure 4.2.2 New Reno-Reno Throughput (Start-Simultaneously)

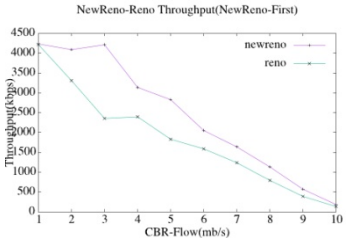

Figure 4.2.3 New Reno-Reno Throughput (Reno-First)

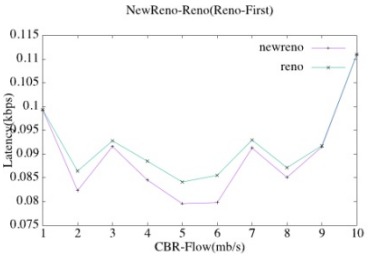

Figure 4.2.4 New Reno-Reno Latency (Reno First)

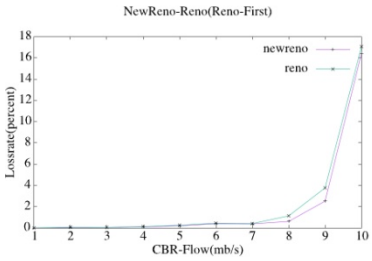

Figure 4.2.5 New Reno-Reno Loss Rate (New Reno-First)

For the pair of Vegas and Vegas, although they are same algorithm, the one who starts first will occupy more bandwidth when the load of the network is not so heavy. It is because when the one starts first, it takes some amount of bandwidth, then the other one begin its flow, it detects the congestion situation before it send packet and in order to avoid congestion, it will decrease its throughput because the Vegas who goes first have been taken a lot of bandwidth. Since Vegas is designed to prevent congestion from happening and guarantee the low latency and packet loss rate, both of two Vegas flows perform almost same latency and packet loss rate no matter which one goes first.

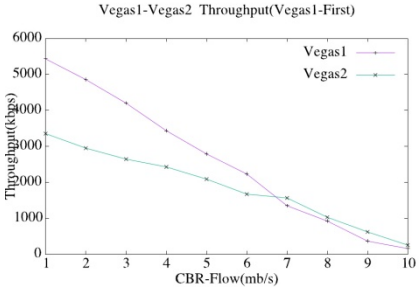

Figure 4.3.1 Vegas1-Vegas2 Throughput (Vegas1-First)

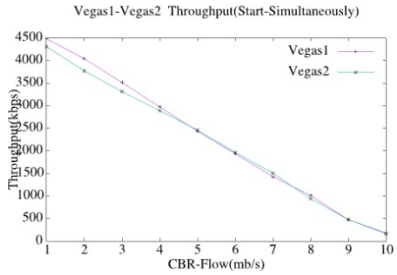

Figure 4.3.2 Vegas1-Vegas2 Throughput (Start-Simultaneously)

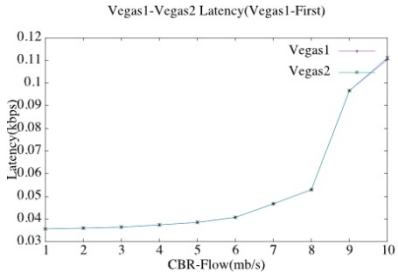

Figure 4.3.3 Vegas1-Vegas2 Latency (Vegas1-First)

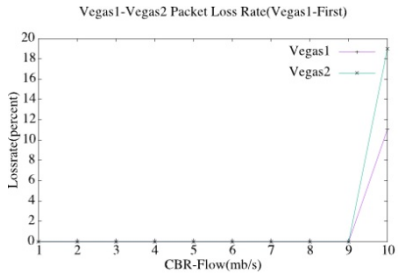

Figure 4.3.4 Vegas1-Vegas2 Loss Rate (Vegas1-First)

Between New Reno and Vegas, it is obviously unfair according to the result of the experiment. We can see that New Reno occupies more bandwidth than Vegas. The reason for this situation is that New Reno performs fast retransmit and fast recovery process when there is congestion and packets loss, sender can deal the problem with high throughput, while Vegas detects congestion at very beginning stage and adjust its throughput. In this case, Vegas will decrease the throughput and takes less bandwidth, while New Reno is able to retransmit its loss packets with higher throughput thanks to the action of Vegas and occupies more bandwidth. But Vegas get lower latency and packet loss rate by sacrificing the throughput.

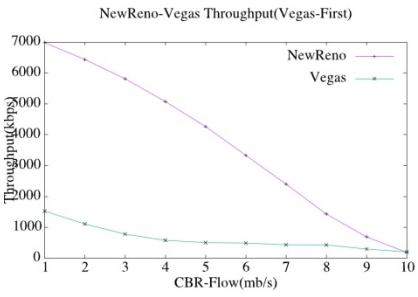

Figure 4.4.1 New Reno-Vegas Throughput (New Reno-First)

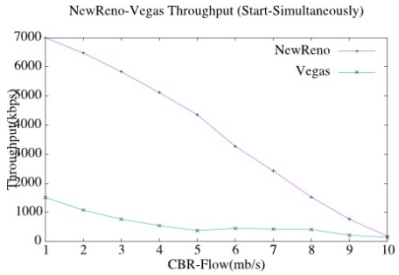

Figure 4.4.2 New Reno-Vegas Throughput (Start-Simultaneously)

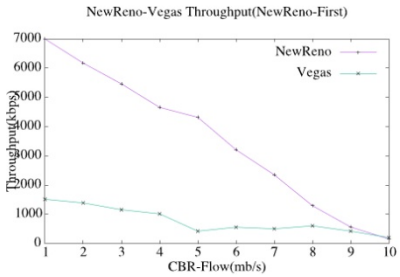

Figure 4.4.3 New Reno-Vegas Throughput (Vegas-First)

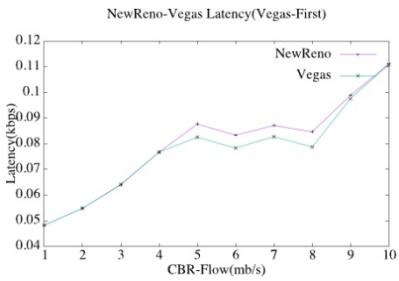

Figure 4.4.4 New Reno-Vegas Latency (New Reno-First)

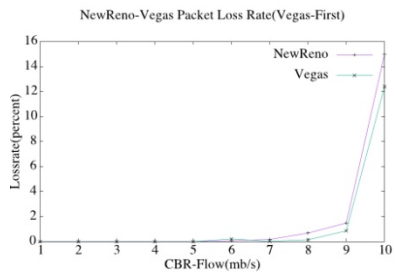

Figure 4.4.5 New Reno-Vegas Loss Rate (New Reno-First)

5. Results for Experiment 3

This experiment is to compare and contrast the influence produced by different queuing disciplines. The results are shown below in figure.

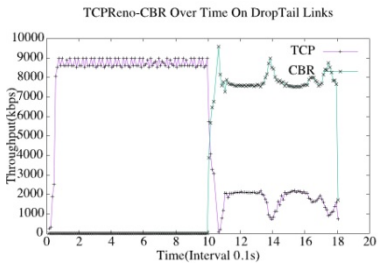

Figure 5.1.1 Reno-CBR Over Time on Drop Tail Links

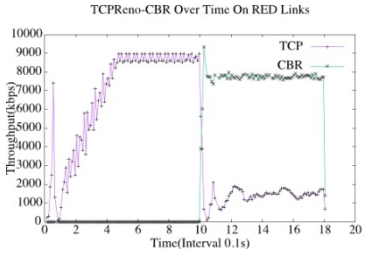

Figure 5.1.2 Reno-CBR Over Time On RED Links

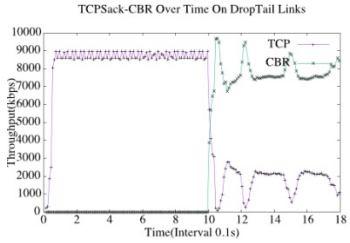

Figure 5.2.1 SACK-CBR Over Time On Drop Tail Links

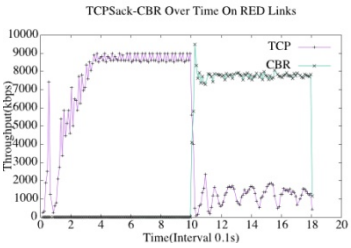

Figure 5.2.2 SACK-CBR Over Time On RED Links

Now we are going discuss the situation that Reno is used in the network. When the Drop Tail is implemented into the network before CBR running, TCP throughput increases rapidly and takes most of bandwidth until it reaches the max TCP window size. Then the throughput maintains about 8800kbps. When UDP begins at 10s, Reno times out because a burst of packets were put into the network which causes severe congestion and performs slow start again. Then the network leaves the congestion state because of low congestion window size of TCP.

While for the RED discipline, some packets are dropped when the queue average length reaches the range between MaxThreshold and MinThreshold of RED algorithm, which will reduce TCP throughput. So it takes some time for TCP to enter the stable stage. When UDP start running, TCP enter slow start too. The throughput of TCP and UDP are smoother when RED is set in the network than Drop Tail is used. It may because RED randomly drop packet to reduce the throughput before queue is overload. While Drop Tail will not let sender knows that there is congestion until the queue is full which causes severe congestion and lowers throughput sharply.

Then when it comes to SACK case, we can see that TCP throughput goes up and down more frequently especially when RED is implemented. It is because SACK can deal with multiple packets loss. When it meets congestion and multiple packets loss, throughput will decrease. Then when sender begins to retransmit, it will send a burst of packet that includes loss packets into network and the load of network will go up suddenly which may cause congestion again.

Table 5.1 to 5.4 presents the average of throughput, latency and packet drop rate in the duration between 14s and 18s.

Table 5.1.1 Performance of flows (Reno and Drop Tail implemented)

	TCP	UDP
Throughput(kpbs)	1715	7970
Latency(s)	0.0709645207252	0.0496607706821
Packet loss rate(%)	0.431167693156	0.443412188392

Table 5.1.2 Performance of flows (Reno and RED implemented)

	TCP	UDP
Throughput(kpbs)	1532	7901
Latency(s)	0.0369867574697	0.0146766795263
Packet Loss Rate(%)	0.692999426431	1.07173029337

Table 5.2.1 Performance of flows (SACK and Drop Tail implemented)

	TCP	UDP
Throughput(kpbs)	1746	7953
Latency(s)	0.0817387493366	0.0537454079956
Packet Loss Rate(%)	0.706164893567	0.505155652743

Table 5.2.2 Performance of flows (SACK and RED implemented)

	TCP	UDP
Throughput(kpbs)	1243	7912
Latency(s)	0.0376785971078	0.0146207714941
Packet loss Rate(%)	0.72110144132	1.06196208997

Theoretically, RED can provide more fairness for pair of flows than Drop Tail, because when it comes to the congestion case, congestion causes packets dropping. When TCP get this signal, it will retransmit the packet and decrease the throughput for the congestion control. While UDP does not deploy congestion control algorithm, which means that it will not reduce the throughput. While RED is implemented in the network, and when the average length in queue reaches between the MinThreshold and MaxThreshold of RED, every packet that arrive receiver will get a possibility to be dropping. Although it also reduce the throughput of TCP, because of random packet dropping mechanism, number of packets come from UDP will be more than TCP because more UDP packets arrive receiver, which perform better fairness in some way. But according to our result, it cannot be fair when TCP competes with UDP. There is no congestion control that implemented in UDP which is used to limit the packet sending rate while TCP does after all. And we can see that throughput of UDP is five times of TCP while UDP packet loss rate cannot reach five times to TCP, although it is higher comparing with Drop Tail.

For the latency, RED provide lower and more smooth latency because the random early drop helps to keep the low average queue length that makes latency lower. While Drop Tail deal with congestion problem when it occurs. Under this condition, queue will maintain high length for a period of time that raises latency.

It is hard to say whether it is good idea to use RED dealing with SACK. As we can see from the result, RED under SACK produces lower throughput than RED under Reno, but the latency is also lower when SACK is used with RED. We think that it depends on the requirement for the network to evaluate the algorithm. If we require high and steady throughput, RED is not a good choice for SACK. But if we focus more on lower latency, we think RED is a good idea.

6. Conclusion

This paper is mainly focus on the performance of variants TCP under different congestion situations and the competition performance of TCP pairs run through same link.

Basically, Vegas shows better performance on the aspects of throughput, latency and packet loss when there is no other TCP competitors in the network. But when it comes the case that there are more than one TCP is running in the network, Vegas lose bandwidth to its competitor when congestion problem becomes severe because in order to prevent congestion from happening, Vegas always reduce throughput before congestion occurs, while other algorithm will not deal with congestion until it happens. Then Vegas will take less bandwidth while other TCPs occupy more bandwidth when the congestion is coming. But Vegas guarantee the lower packet loss rate and latency at least. Besides that, feature of dealing with multiple packets loss in New Reno gives better performance comparing with Reno. This feature helps New Reno have higher throughput than Reno. For the comparison between the Drop Tail and RED, RED drops packets randomly before congestion happening to keep the queue under low average length which helps to reduce the latency and packet loss rate.

Vegas have a lot of good feature regardless of competing with other TCP under the same network. This drawback makes it passive when competing with other TCP. It will be better if there is a way to improve the performance of competition in the future.

Reference

[1] Kevin Fall and Sally Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP”, ACM SIGCOMM Computer Communication Review Homepage archive, Volume 26 Issue 3, July 1996

[2] “TCP congestion-avoidance algorithm”, http://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm

[3] “Random Early Detection”, http://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.adv-qdisc.red.html