

2003

# Deployment Considerations for the TCP Vegas Congestion Control Algorithm

Sonia Fahmy

*Purdue University*, fahmy@cs.purdue.edu

Mridul Sharma

**Report Number:**

03-022

---

Fahmy, Sonia and Sharma, Mridul, "Deployment Considerations for the TCP Vegas Congestion Control Algorithm" (2003). *Computer Science Technical Reports*. Paper 1571.

<http://docs.lib.purdue.edu/cstech/1571>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact [epubs@purdue.edu](mailto:epubs@purdue.edu) for additional information.

**DEPLOYMENT CONSIDERATIONS FOR THE TCP  
VEGAS CONGESTION CONTROL ALGORITHM**

**Sonia Fahmy  
Mridul Sharma**

**Department of Computer Sciences  
Purdue University  
West Lafayette, IN 47907**

**CSD TR #03-022  
June 2003**

**Deployment Considerations for the TCP Vegas Congestion Control Algorithm**  
**Sonia Fahmy and Mridul Sharma**  
**Department of Computer Sciences, Purdue University**  
**West Lafayette, IN 47907-2066**

*Abstract*-- TCP congestion control has been the subject of significant research for more than two decades. Various algorithms including TCP Tahoe, TCP Reno, TCP NewReno, SACK, FACK, and their variations have been proposed to overcome the problem of network congestion and increase network utilization. The TCP Vegas algorithm has been extensively debated in recent years. TCP Vegas' behavior in the presence of competing TCP Reno connections has been the main barrier to its deployment. This paper studies TCP Vegas and investigates why it is less aggressive than TCP Reno in mixed configurations. Simulation experiments have been carried out to demonstrate the problems with the deployment of TCP Vegas in the operational Internet.

## 1. Introduction

The success of the Internet can be largely attributed to the flexibility of the TCP/IP protocol suite. At the heart of the TCP/IP protocol suite success is its ability to deliver service in times of extremely high demand. The key reason behind this is TCP's congestion control algorithms [3]. However, as the demand for access has increased, it becomes important to make efficient use of network resources. The TCP Vegas congestion control algorithm [1][2] is a TCP flavor that has been proposed as an alternative TCP implementation that increases utilization. In this paper, we study the TCP Vegas congestion control algorithm and understand the barriers to its deployment in the operational Internet.

This paper is organized as follows. Section 2 gives a brief overview of TCP Vegas congestion control, and compares TCP Reno to TCP Vegas. Section 3 discusses related work. Section 4 explains the problems hindering the deployment of Vegas in the Internet. Section 5 reports our simulation results. Section 6 discusses two solutions proposed in [12] that may tackle the Vegas deployment barriers. Finally, Section 7 gives some concluding remarks.

## 2. TCP Vegas Congestion Control Algorithm

The TCP Vegas protocol [1] and [2] proposes several new techniques for improving TCP. These include a new timeout mechanism, a novel approach to congestion avoidance that avoids packet loss, and a modified slow start algorithm. TCP Vegas introduces three changes that affect TCP's (fast) retransmission strategy. Vegas reads and records the system clock each time a segment is sent. It measures the round trip time (RTT) for every segment sent. The measurements are based on fine-grained clock values. Using the fine-grained RTT measurements, a time-out period for each segment  $s$  is computed. When a duplicate acknowledgement (ACK) is received, TCP Vegas checks whether the timeout period has expired. If so, the segment is retransmitted. When a non-duplicate ACK that is the first or second after a fast retransmission is received, TCP Vegas again checks for the expiration of the timer and may retransmit another segment. In case of multiple segment loss and more than one fast retransmission, the congestion window is reduced only for the first fast retransmission.

TCP Vegas detects incipient congestion by comparing the measured throughput to its notion of expected throughput. Vegas defines a given connection's BaseRTT as the minimum of all measured round trip times. Expected throughput is defined as:

$$\text{Expected} = \text{CWND} / \text{BaseRTT}$$

where CWND is the size of the current congestion window.

Vegas estimates the current flow throughput by using the actual round trip time:

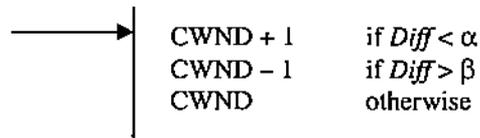
$$\text{Actual} = \text{CWND} / \text{RTT}$$

where RTT is the actual round trip time of a packet.

Vegas compares the actual throughput to the expected throughput and computes the difference as:

$$\text{Diff} = (\text{Expected} - \text{Actual}) \text{BaseRTT}$$

Based on Diff, Vegas updates the window size as follows:  
 CWND



Where  $\alpha$  and  $\beta$  are two thresholds,  $\alpha < \beta$ , roughly corresponding to having too little and too much extra data in the network, respectively.

Vegas also modifies the slow start mechanism. To be able to detect and avoid congestion during slow-start, Vegas allows exponential growth every other RTT. In between, the congestion window stays fixed so that a valid comparison of the expected and actual rates can be made. When the actual rate falls below the expected rate by a certain amount – defined by a threshold  $\gamma$  - Vegas changes from slow start mode to linear increase/ decrease mode.

Table 1 compares TCP Reno to TCP Vegas.

	TCP Reno	TCP Vegas
Slow Start	Exponential growth every RTT	Exponential growth every other RTT
	Comparison to ssthresh triggers change of mode to congestion avoidance	Comparison of expected and actual rates and threshold $\gamma$ trigger change of mode to congestion avoidance
Congestion Avoidance	Reactive: uses loss of segments as a signal that there is congestion in the network	Proactive: Detects the incipient congestion – before losses occur – so losses can be prevented
	Increases window size linearly every RTT	Increases or decreases based on the state of network utilization
Fast Retransmission	Retransmits when it receives n duplicate ACKs (n is typically three)	May retransmit when it receives one duplicate ACK based on its fine-grained timer
	Can patch up one packet loss. Coarse-grained timer will timeout in case of multiple packet loss.	Additional retransmissions take care of multiple packet loss.
	Possible to decrease the congestion window more than once for losses that occur during one RTT interval.	Congestion window is only decreased if the retransmitted segment was previously sent after the last decrease. Hence multiple reductions of congestion window in case of multiple segment loss are prevented
	Congestion window is halved after recovery	Congestion Window is reduced by only $\frac{1}{4}$ after a recovery
Congestion Window Initialization	Congestion window is initialized to one after a time-out	Congestion window is initialized to two after a time-out
Delays and jitter	Long delays and jitter due to the large average queue sizes	Due to small backlogs in the network, TCP Vegas experiences short delays and once the system reaches equilibrium, the delay jitter becomes very small as well
Fairness	TCP Reno is biased against connections with long RTT	TCP Vegas treats connections with long and short RTTs equally

		well
Head to Head transfer: TCP Reno and TCP Vegas	Being aggressive, TCP Reno gets a higher share of bandwidth under light load	Being conservative, TCP Vegas gets a lower share of bandwidth under light load

Table 1: Differences between TCP Reno and TCP Vegas

### 3. Related Work

The new techniques introduced by TCP Vegas for fast retransmission, congestion avoidance, and slow start, their effect on TCP performance, and the TCP Vegas behavior in the presence of competing TCP Reno connections have been investigated by previous researchers. This section gives a brief overview of their work.

Ahn et al. [6] conducted experiments on a wide-area network emulator. Their results show that for all cases of eight different background traffic work-loads and thirty 512 KB transfers, Vegas transmits fewer bytes than Reno. For the highest levels of congestion studied, Vegas obtained slightly higher throughput than Reno. This occurred because Reno sources timed out more frequently than Vegas sources since Reno sources suffer more multiple packet drops. However, as the degree of congestion drops, Vegas sources yield bandwidth to Reno sources, because Reno sources keep more data in the network and decrease the Vegas congestion windows. As a rule of thumb, in head to head transfers, Reno will get 50% higher throughput than Vegas. In all the experiments with heavy congestion, Vegas used the network more efficiently than Reno, and under most workloads, Reno senders received a better than average share of the network bandwidth.

In Mo et al. [4], the authors show that TCP Vegas, as opposed to TCP Reno, is not biased against connections with long delays. By both simulation and analysis, the authors confirm that TCP Vegas does not receive a fair share of bandwidth in the presence of a TCP Reno connection. Let  $qv(t)$  and  $qr(t)$  denote the number of Vegas and Reno packets in the buffer at time  $t$  and let  $B$  be the buffer size. Assume  $qv(t) = k \leq 3$ ,  $qr(t)$  ranges from  $[0, B-k]$ . Assume that  $qr(t)$  is distributed uniformly in the interval  $[0, B-k]$ . The average queue size of the Reno user is approximately  $(B-k)/2$ . The ratio of TCP Vegas throughput to TCP Reno throughput is roughly  $(2k)/(B-k)$ . When  $B$  is relatively large, which is usually the case, then it is obvious that TCP Reno gets much higher bandwidth than TCP Vegas. The TCP Reno congestion avoidance scheme is aggressive in the sense that it leaves little room in the buffer for other connections, while TCP Vegas is conservative, and tries to occupy little buffer space. When a TCP Vegas connection shares a link with a TCP Reno connection, the TCP Reno connection uses most of the buffer space and the TCP Vegas connection backs off, interpreting this as a sign of network congestion. The authors suggest that Random Early Detection (RED) Gateways could be used to make TCP Vegas more compatible with TCP Reno in a competitive environment. The authors also investigated the possibility of deploying TCP Vegas, adopting gateway control such as deficit round robin (DRR) gateways.

In U. Hengartner et al [5], the authors decompose TCP Vegas into different mechanisms, and assessed the effect of each of these on performance. A  $2^k$  factorial design with replications has been used for evaluation. The authors show that the Vegas performance gains are primarily due to the TCP Vegas new techniques for slow start and congestion recovery, as they are able to avoid timeouts due to multiple segment loss. Therefore, TCP Vegas appears to be quite successful in overcoming a well-known problem of TCP Reno.

### 4. Deployment of TCP Vegas

We have seen in section 2 how TCP Vegas tries to maximize throughput by minimizing retransmissions. In section 5, simulations show that TCP Vegas is indeed able to obtain better performance than TCP Reno. TCP Reno, however, is already widely deployed in the present Internet. If we aim to deploy TCP Vegas in the operational network, we need to consider the scenario when both TCP Reno and TCP Vegas co-exist in the network. There is an unfairness problem when TCP Reno and TCP Vegas share a bottleneck link. Past researchers have observed this problem, as discussed in the last section. The TCP Reno congestion avoidance scheme is aggressive in the sense that it leaves little room in the buffer for other connections,

while TCP Vegas is conservative and tries to occupy little buffer space. When a TCP Vegas connection shares a link with a TCP Reno connection, the TCP Reno connection uses most of the buffer space and the TCP Vegas connection backs off, interpreting this as a sign of network congestion. Simulations in Section 5 demonstrate this problem.

## 5. Simulations

The topology used for our preliminary simulations is depicted in figure 1. All the links are 5 Mbps except for the bottleneck link between r1 and r2, which is 0.5 Mbps. The propagation delay for all links is 2 ms. The queue sizes and the packet drop policies at the routers are different for different simulations and are indicated separately for each one.

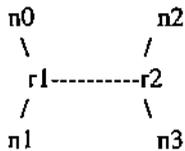


Figure 1. Topology for the simulations

The throughput of each flow (connection) was calculated as follows:

$$\text{Throughput at time } t_0 = (\text{total number of bytes received from time 0 to } t_0) / (t_0)$$

In the next few subsections, we discuss each simulation and its results.

### 5.1 Independent TCP Reno and TCP Vegas connections

In our first set of simulations, the performance of TCP Reno and the performance of TCP Vegas were evaluated separately. The queue size at the routers r1 and r2 is 64 packets, and they use simple “Droptail” as the packet drop scheme when the buffers overflow. First, a TCP Reno connection is set up between n0 and n3. Its throughput is measured. Then, a TCP Vegas connection is set up, and its throughput is measured. Figure 1 and figure 2 show the throughput over time for the TCP Reno and TCP Vegas connections respectively. As expected, TCP Reno opens up its window exponentially. Reno soon fills the router buffers and experiences loss. As a result, there is a sharp dip in the graph. Reno then decreases its congestion window to half, and so on. The dips in the throughput graph correspond to TCP Reno experiencing losses.

TCP Vegas detects incipient congestion and adjusts its sending rate accordingly without incurring any losses. Figure 2 shows no dips in throughput, and no losses. TCP Vegas throughput smoothly increases to its optimal value, throughout the duration of the simulation. It can be seen from the graphs that TCP Vegas is able to achieve higher throughput than TCP Reno. Thus, these graphs validate the claim made by previous researchers that TCP Vegas is able to achieve higher throughput with fewer retransmissions than TCP Reno.

### 5.2 TCP Reno and TCP Vegas connections sharing a bottleneck link

In this simulation, the performance of TCP Reno and TCP Vegas is measured when they both share a bottleneck link. Again, the queue size at the routers is 64 packets, and they use Droptail as the packet drop scheme. In the above topology, there is a TCP Reno connection from n0 to n3 and a TCP Vegas connection from n1 to n2. We observe that the TCP Reno connection steals bandwidth from TCP Vegas connection, as depicted in figure 3. There are several reasons for Vegas being conservative (less aggressive) compared to TCP Reno. The Vegas congestion avoidance algorithm moderates its additive growth regimen according to delays. In contrast, Reno’s congestion avoidance regimen keeps growing its window size until a drop occurs. Vegas can add or subtract one segment to the congestion window every RTT. Reno’s congestion avoidance always adds one segment to the congestion window every RTT. Vegas maintains a fine grained retransmit timer for each outstanding segment. The Vegas early retransmit is activated by receiving a

duplicate ACK. When Vegas chooses to retransmit a segment based on the fine grained timer, it reduces its congestion window by a quarter. This means that when Vegas retransmits a segment earlier than Reno, it reduces its congestion window faster than Reno would. The Vegas slow start mechanism limits the exponential growth to every other RTT. Thus, TCP Reno is able to gain throughput at the expense of TCP Vegas.

### 5.3 Effect of varying queue lengths and drop policy at the routers

In our next set of simulations, we varied the queue lengths of the router buffers to see the effect on the performance of TCP Reno and TCP Vegas sharing a bottleneck link. The setup is same as in the previous experiment-- only the queue length is varied. Table 2 illustrates the output of the simulations.

Buffer size at the router	Reno Throughput (Kbps)	Vegas Throughput (Kbps)
6	231	269
9	232	267
15	374	125
25	435	66
50	435	66

Table 2: Results of simulations varying the buffer size at the router

The results show that as the buffer size increases, TCP Reno is able to perform better at the cost of the TCP Vegas performance. When the queue size is small, TCP Reno's faster and unresponsive exponential opening of the congestion window during slow start results in overshooting the available buffer space and losing multiple segments. A timeout typically results from these multiple losses, and throughput severely degrades. On the other hand, by sensing the incipient congestion, TCP Vegas is able to avoid such timeouts and thus perform better than TCP Reno. As the buffer size increases, TCP Reno is able to get sufficient room in the buffer. TCP Vegas misinterprets these full buffers as a sign of network congestion, and due to its conservative approach, it backs off. Reno is then able to grab even more bandwidth. With the increase in buffer size at the router, Reno is able to achieve higher throughput. The table shows that with buffers 25 packets or larger, the throughputs do not change. This is because the sender cannot send faster than the receiver advertised window.

Other researches have also observed that when the queue lengths are small, both TCP Vegas and TCP Reno are able to achieve more or less similar throughputs. Thus, if the connections are given an illusion that the queue length at the router is small, some degree of fairness can be achieved. In RED, if the average queue length lies between the minimum and maximum thresholds, the packets are dropped with linearly increasing probability. If the queue length exceeds the maximum threshold, the packets are dropped with a probability of one. If the values of the two thresholds are reduced, the connections will infer that buffer sizes are small, and higher fairness can be achieved.

We performed another set of simulations with the same setup as in section 5.2, but with RED (Random Early Detection) as the packet drop policy at the routers. Figure 4 shows the throughputs of TCP Reno and TCP Vegas connections over time. We see that some degree of fairness is achieved. This is because packets of the misbehaving flow (TCP Reno in this case) are dropped with a higher probability. From this result, one may expect that the fairness can be further improved if the packet drop probability is further increased. The fairness will increase because the packets of TCP Reno will be dropped more aggressively, thus decreasing the average window size of TCP Reno. The total throughput, however, will degrade as more packets are dropped. Hence, there is an inevitable tradeoff between fairness and total throughput. We can therefore conclude that packet drop schemes such as RED can be modified such that the misbehaving flows are detected and punished without affecting the total throughput considerably. Table 3 summarizes our results.

Packet drop scheme at router	Reno (kbps)	Vegas (kbps)
Droptail	435	65
RED	310	190

Table 3: Effect of using different packet drop schemes at the router

#### 5.4 Effect of number of hops/propagation delay

In the final set of simulations, we varied the number of hops traversed by the connections and observed their effects on the performance of TCP Reno and TCP Vegas. For each simulation set, we increased the number of hops between source and destination by one over the previous simulation set. The additional link(s) has the same characteristics as the bottleneck link. The queue size was 6 packets. Table 4 lists the results.

Num of hops	Reno (Kbps)	Vegas (Kbps)
2	232	268
3	162	338
4	194	306
5	173	327

Table 4: Effect of increasing the number of hops when the queue length is 6

As expected, since the queue length is small, Vegas performs better than Reno in all cases. The performance of Reno seems to slightly degrade with increasing number of hops. Table 5 lists the results obtained when the queue size is 64. In this simulation, the bottleneck link was set to 5 Kbps.

Number of hops	Reno (Kbps)	Vegas (Kbps)
2	4.49	.49
4	4.26	.69
5	4.08	.86

Table 5: Effect of number of hops when the queue size is 64

Since the buffer size is large in this case, Reno outperforms Vegas in all cases. With the increase in the number of hops, there is a small increase in Vegas's throughput. From these simulations, we can conclude that the performance of TCP Reno and TCP Vegas is not significantly affected by the number of hops or propagation delay.

We plan to conduct more extensive experiments with multiple TCP Reno and Vegas connections, and different round trip times. Scenarios such as flows starting and stopping at different times are also interesting. It will also be interesting to study the behavior of TCP Reno and TCP Vegas when the background/application traffic models vary dynamically during the experiment.

## 6. Recently Proposed Solutions

We are currently evaluating several proposed solutions to tackle the unfairness problem when TCP Reno and TCP Vegas share the same bottleneck links. Some of these proposals are discussed in this section. In [12], the authors address the fairness issues between TCP Reno and TCP Vegas. The authors consider a migration path for a protocol from being an immature one to one being deployed in the operational network. The authors suggest two possible ways to improve fairness between TCP Reno and TCP Vegas. One is to modify the congestion control algorithm of TCP Vegas, and the other is to modify the RED algorithm for detecting packets from TCP Reno connections. The network model chosen for analysis and simulations has multiple TCP Reno and TCP Vegas senders. All these are connected to a router with drop tail or RED. There is a single receiver that receives from all senders through this router. One way the authors claim to improve the fairness between TCP Reno and TCP Vegas is to make TCP Vegas behave like TCP Reno in the presence of TCP Reno connections. This new Vegas is called Vegas+. Vegas+ has

two modes of updating its congestion window. In moderate mode, Vegas+ updates its congestion window exactly as the original Vegas. However, in aggressive mode, Vegas+ behavior is identical to TCP Reno. The switching between the two modes is determined by two new variables  $count$  and  $count_{max}$ . When an ACK arrives the RTT value and the window size are called  $rtt$  and  $ws$  respectively. Let the previous RTT value and window size be  $rtt_p$  and  $ws_p$ . Now,  $count$  is updated according to following algorithm:

```

count = count + 1          if rtt > rttp and ws = wsp
count = count - 1        if rtt < rttp and ws = wsp
count = count / 2        if a duplicate ACK is received and the fine grained timer expires
count = 0                 if the coarse grained retransmission timer expires

```

Vegas+ transits from one mode to the other as follows:

```

count = countmax      Transit from moderate to aggressive mode
count = 0                Transit from aggressive to moderate mode

```

The authors claim that the increase in RTT when the window size remains unchanged is due to TCP Reno connections. Hence, when  $count$  reaches a certain threshold, Vegas+ starts behaving like TCP Reno to achieve fairness. When  $count$  becomes zero, it is most probably due to Vegas+ itself and hence it goes into moderate mode.

Other proposed solutions use router-based mechanisms. As previously discussed, Random Early Detection (RED) increases fairness among Reno and Vegas connections. Researchers have developed the Stabilized RED (SRED) algorithm to implement mechanisms to detect TCP Reno connections, and to drop more packets from TCP Reno connections. In SRED, a fixed-size table is maintained, each entry of which contains information about incoming packets (i.e., source-destination address and possibly port numbers). If such table (called zombie list (ZL)) is full on packet arrival, the router randomly selects one entry from the list. If the information in the selected entry is identical to the packet information, it is called a hit and a count related to that entry is incremented. Misbehaving flows can be identified by observing the hits and count for each entry of the zombie list.

To drop more packets from misbehaving connections, ZL-RED performs packet drop in two steps. In the first step, an incoming packet is dropped with probability  $p_1$ . Probability determination follows the algorithm presented in reference [7]. In the second step, the router drops the packet with probability  $p_2$  if the packet has not been dropped in the first step. Probability  $p_2$  is determined as:

$$p_2 = \text{hit}_{\text{drop}} / (1 + P(t)), \quad \text{if Hit}(t) = 1 \text{ and } q_{\text{len}} P(t) \geq 2.0$$

$$p_2 = 0, \quad \text{otherwise}$$

where  $\text{Hit}(t) = 1$ , if the incoming packet hits the zombie list,  $\text{Hit}(t) = 0$  otherwise.  $q_{\text{len}}$  is the number of packets in the router buffer when the packet arrives at the router.  $P(t)$  is the probability with which the incoming packet hits.  $\text{hit}_{\text{drop}}$  is a control parameter of the ZL-RED algorithm. It affects the packet drop probability of misbehaving connections.

One difficulty with these mechanisms is the determination of the control parameters  $count_{max}$  and  $\text{hit}_{\text{drop}}$ . A low value of  $count_{max}$  can make Vegas enter aggressive mode, even when there are no TCP Reno connections. On the other hand, a high value of  $count$  may not be able to achieve substantial fairness. In case of congestion in the network, Vegas+ will experience increase in RTT. As Vegas has a mechanism to detect incipient congestion, there may be no loss and hence the window size may remain same. In this case, Vegas+ can interpret this as the existence of TCP Reno. Vegas+ will thus start behaving as aggressive as TCP Reno. If this happens, the whole purpose of the Vegas innovative congestion control techniques will be defeated. This particular case needs to be studied. We must ensure that while attempting to achieve fairness between TCP Reno and TCP Vegas, we do not lose the original characteristics of Vegas, which are the root cause of Vegas improved performance over Reno.

In addition, the effect of using the ZL-RED technique when there are multiple TCP Reno flows needs to be studied. In this case, will these TCP Reno connections be able to take advantage of any excess bandwidth available (if that is the case)? We need to investigate the effects of using ZL-RED on the total throughput achieved. Fairness should not be achieved at the cost of total throughput.

## 7. Conclusions

In this paper, we have reestablished that TCP Vegas is able to achieve better performance than TCP Reno. However, when we consider TCP Reno and TCP Vegas connections sharing bottleneck links, we find that Reno steals bandwidth from Vegas. Different approaches proposed by researchers have been studied. The question of which approach is better is still an open research problem. This paper has established a basis for further research work in the direction of deployment of TCP Vegas in the current Internet.

### Annotated Bibliography

[1] L.S. Brakmo, S.W.O'Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In Proc. Of ACM SIGCOMM '94, pages 24-35, London, October 1994.

This paper proposes an alternative implementation of TCP that employs three key techniques for congestion detection and avoidance.

[2] L.S. Brakmo and L.L. Peterson. TCP Vegas: End-to-End Congestion Avoidance on a global Internet. IEEE Journal on Selected Areas in Communications, 13(8): 1465-1480, Oct 1995

This paper is a more detailed version of [1].

[3] V. Jacobson. Congestion Avoidance and Control. In Proceedings of ACM SIGCOMM 88, pages 314-329, August 1988

Jacobson discusses all the algorithms for congestion avoidance and control.

[4] J. Mo, R.J. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In Proceedings of INFOCOM '99, pages 1556-1563, March 1999.

The authors show that TCP Vegas, as opposed to TCP Reno, is not biased against connections with long delays, and that TCP Vegas does not receive a fair share of bandwidth in the presence of a TCP Reno.

[5] U. Hengartner, J. Bolliger and T. Gross. TCP Vegas Revisited. In Proceedings of the Conference on Computer Communications (IEEE Infocom), Tel Aviv, Israel, March 2000

This papers presents a detailed performance evaluation of TCP Vegas by decomposing Vegas into various novel mechanisms proposed.

[6] J. S. Ahn, P. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: Emulation and Experiment. In Proceedings of ACM SIGCOMM '95, pages 185-195, August 1995.

The authors strengthen the claims about TCP Vegas made in [1]. They conclude that Vegas offers improved throughput over Reno while reducing packet losses and subsequent retransmitted segments.

[7] P. B. Danzig, Z. Liu, and L. Yan. An Evaluation of TCP Vegas by Live Emulation. Technical Report 91-588, Computer Science Department, USC, 1994.

This paper presents evidence that the Vegas performance enhancements to TCP flow control proposed do not always simultaneously yield both increased throughput and decreased load on the network.

[8] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara. Comparisons of Packet Scheduling Algorithms for Fair Service Among Connections on the Internet. In Proceedings of the Conference on Computer Communications (IEEE Infocom), Tel Aviv, Israel, March 2000.

The authors investigate TCP Vegas fairness. They make clear that TCP Vegas cannot help improve fairness among connections in FIFO and RED.

[9] S. Fahmy and T. P. Karwa. TCP Congestion Control: Overview and Survey of Ongoing Research. Technical Report CSD-TR-01-016, Purdue University, 2001.

The paper gives an overview of the various TCP congestion control algorithms and compares their performance via simulation. It also discusses the recent and ongoing work on these algorithms.

[10] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and Stability of Congestion Control Mechanisms of TCP. In Proceedings of INFOCOM '99, pages 1329-1336, March 1999.

This paper shows that while TCP Vegas can provide almost fair service among connections, there is some unfairness caused by the essential nature of TCP Vegas.

[11] T. Bonald. Comparison of TCP Reno and TCP Vegas via Fluid Approximation. Technical Report, Inria, Number RR-3563, p. 34.

As the title says, this papers compares TCP Reno and TCP Vegas using fluid approximation modeling.

[12] G. Hasegawa, M. Murata, and Kenji Kurata. Analysis and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet. ICNP 2000 Nov. 2000

The authors propose two approaches for improving the fairness between TCP Reno and TCP Vegas. Both analysis and simulation experiments have been used for evaluating fairness, and validating the effectiveness of the proposed mechanisms.

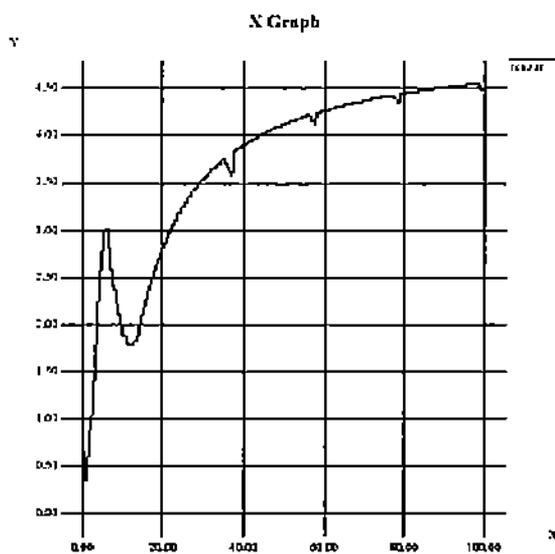


Figure 1: Independent Reno Connection

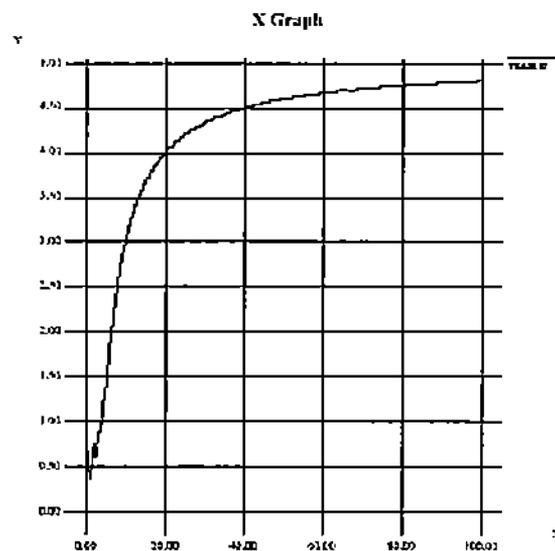


Figure 2: Independent Vegas Connection

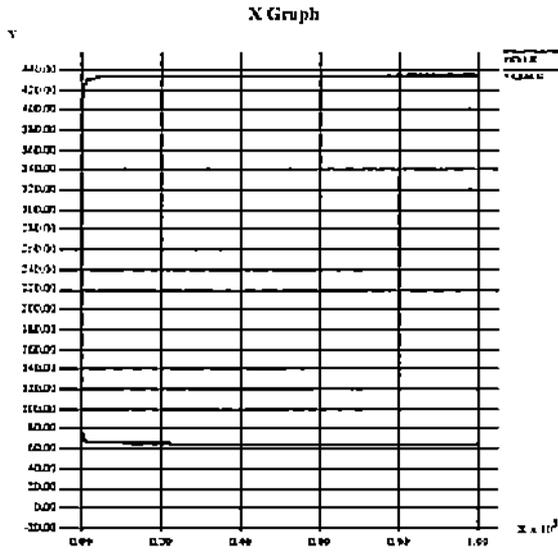


Figure 3: Reno and Vegas with Droptail at the router

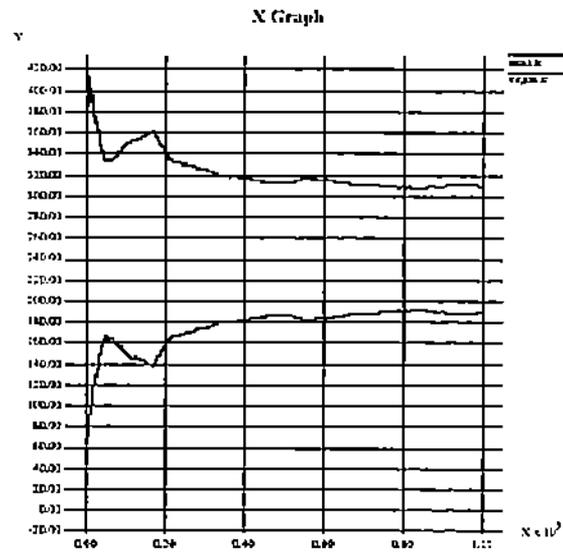


Figure 4: Reno and Vegas with RED at the router