IEEE Symposium on Applications and the Internet (SAINT 2003),
Orlando, Florida, January 2003. LA-UR 01-6506

Enabling Compatibility Between TCP Reno and TCP Vegas*

W. Feng
Computer & Computational Sciences Division
Los Alamos National Laboratory
feng@lanl.gov

S. Vanichpun
Dept. of Electrical & Computer Engineering
University of Maryland, College Park
sarut@glue.umd.edu

Abstract

Despite research showing the superiority of TCP Vegas over TCP Reno, Reno is still the most widely deployed variant of TCP. This predicament is due primarily to the alleged incompatibility of Vegas with Reno. While Vegas in isolation performs better with respect to overall network utilization, stability, fairness, throughput and packet loss, and burstiness; its performance is generally mediocre in any environment where Reno connections exist. Hence, there exists no incentive for any operating system to adopt TCP Vegas.

In this paper, we show that the accepted (default) configuration of Vegas is indeed incompatible with TCP Reno. However, with a careful analysis of how Reno and Vegas use buffer space in routers, Reno and Vegas can be compatible with one another if Vegas is configured properly. Furthermore, we show that overall network performance actually improves with the addition of properly configured Vegas flows competing head-to-head with Reno flows.

Keywords: TCP Reno, TCP Vegas, congestion control, congestion avoidance, compatibility, fairness, convergence.

1 Introduction

To address a series of congestion collapses, Jacobson proposed a congestion-control mechanism in TCP that later became known as TCP Tahoe [10]. Since then, many modifications have been made to TCP, resulting in two more notable variants — TCP Reno [11] and TCP Vegas [3].

TCP Reno, like TCP Tahoe, allows congestion to occur (i.e., induces packet loss) in order to estimate the available bandwidth in the network. Once packet loss is detected, Reno recovers by cutting its window size in half. This behavior causes a periodic oscillation in the window size; an oscillation that many next-generation Internet applications do not tolerate well. Further, recent work shows that this oscillatory behavior induces chaotic behavior into the network [4, 15], thus adversely affecting overall network performance.

In contrast, TCP Vegas generally performs better with respect to overall network utilization [1,3], stability [9,13], fairness [9,13], throughput and packet loss [1,3], and burstiness [4] when the entire network consists of Vegas-only connections. However, research to date has shown that when Reno and Vegas perform head-to-head, Reno generally steals bandwidth from Vegas [1, 13]. Consequently, while Vegas has been around for over five years, its adoption has been non-existent due to perceived incompatibilities between Reno and Vegas.

With a careful analysis of how Reno and Vegas use buffer space in routers, we will show that Reno and Vegas can be compatible with one another if Vegas is configured properly. Further, overall network performance actually improves with the addition of properly configured Vegas flows competing head-to-head with Reno flows, thus encouraging the incremental adoption of Vegas.

2 Congestion-Control Mechanisms

To ensure efficient use of network bandwidth, TCP controls its sending rate based on feedback from the network. In order to control the sending rate, TCP estimates the available bandwidth in the network via a *bandwidth-estimation scheme* [13]. In Tahoe and Reno, the bandwidth-estimation scheme uses packet losses (as an indication of network congestion) to estimate available bandwidth while Vegas uses the difference in the expected and actual sending rates.

2.1 TCP Reno

While there are no packet losses, Reno continues to increase its window size, and hence sending rate, by one packet each round-trip time (RTT), thus allowing congestion to eventually occur. Reno then detects congestion via packet loss and recovers from it by halving the size of the sender window (i.e., halving the sending rate).

2.2 TCP Vegas

Vegas enhances Reno by adopting a bandwidth-estimation scheme that tries to avoid rather than react to

*This work was supported by the U.S. Dept. of Energy's LDRD-ER Program and the U.S. Dept. of Energy's Office of Science SciDAC Program through Los Alamos National Laboratory contract W-7405-ENG-36.

congestion. Specifically, Vegas uses the difference in the expected and actual flow rates to estimate the available bandwidth in the network. When the network is uncongested, the actual flow rate is close to the expected flow rate; otherwise, the actual rate is smaller than the expected rate, indicating that buffer space in the network is filling up and that the network is approaching a congested state. The difference in flow rates can be translated into the difference between the window size and the number of acknowledged packets during the RTT, respectively, i.e., $Diff = (Expected - Actual) \times BaseRTT$ where $Expected$ is the expected rate, $Actual$ is the actual rate, and $BaseRTT$ is the minimum-observed RTT.

To adjust the size of the congestion window ($cwnd$) appropriately, Vegas uses two threshold values, α and β (whose default values are 1 and 3, respectively), to control the adjustment of $cwnd$ at the source host as follows:

$$cwnd \leftarrow \begin{cases} cwnd + 1 & \text{if } Diff < \alpha \\ cwnd - 1 & \text{if } Diff > \beta \\ cwnd & \text{otherwise } (\alpha \leq Diff \leq \beta) \end{cases}$$

Conceptually, Vegas tries to keep at least α packets but no more than β packets queued in the network. Thus, with only one Vegas connection, the window size of Vegas converges to a point that lies between $window + \alpha$ and $window + \beta$ where $window$ is the maximum window size that does not cause any queueing.

Selecting α and β holds an implicit tradeoff between network utilization, goodput, and fairness. By using the default settings for these parameters, i.e., $\alpha = 1, \beta = 3$, prior research inadvertently favored Reno over Vegas [8, 13].

3 Compatibility of TCP Reno and TCP Vegas

Prior research demonstrates that Vegas (in isolation) generally performs better than other implementations of TCP [1, 3, 4, 9, 13]. Ahn et al. [1] and Mo et al. [13] also show that when a Vegas connection competes with a Reno connection, Vegas does not receive a fair share of bandwidth due to its conservative congestion-avoidance mechanism.

Here, we show that the "conservative" congestion-avoidance mechanism is *not* to blame for Vegas's inability to grab a fair share of bandwidth. Rather, the alleged incompatibility between Reno and Vegas is due to using the default (mis)configuration of Vegas parameters, i.e., $\alpha = 1$ and $\beta = 3$.

Consider one TCP Vegas connection and one TCP Reno connection over a bottleneck link. At steady state, the Vegas connection keeps (on average) approximately $\bar{q}_v \leq \beta$ packets in the queue while the Reno connection tries to gain as much bandwidth (and queue space) as it can until a packet is lost. Hence, the number of TCP Reno packets in the queue is $q_r \in [0, B - \bar{q}_v]$ where B is the buffer capacity at the bottleneck link. Assuming that the average value of q_r is $\bar{q}_r = \frac{B - \bar{q}_v}{2}$ (e.g., uniform distribution), the ratio of Vegas throughput (λ_v) to Reno throughput (λ_r) is given by

$$\frac{\lambda_v}{\lambda_r} = \frac{\bar{q}_v}{\bar{q}_r} = \frac{2\bar{q}_v}{B - \bar{q}_v}. \qquad (1)$$

Thus, when $B = 3\bar{q}_v$, the ratio of the throughputs is one. As B increases further, Reno is favored. Hasegawa et al. [8] provide a more complete analysis of the throughputs when there are N_v Vegas connections and N_r Reno connections competing for the queue space at the same bottleneck link.

3.1 Analysis of Two Connections

Consider the case when one Vegas connection and one Reno connection share a bottleneck link using a droptail queue. Let the bottleneck link have a transmission rate of μ packets/s and a round-trip propagation delay of τ seconds with queue size B packets. Let q_v and q_r denote the number of Vegas packets and Reno packets in the queue, respectively. Then, in order to allow Vegas to compete with Reno, Vegas must set its α and β parameters so that Equation (1) is such that $\lambda_v/\lambda_r \approx 1$.

Let W, d, and D denote the window size (i.e., $cwnd$), BaseRTT delay, and actual RTT delay of Vegas, respectively. At steady state, Vegas tries to keep the difference ($Diff$) between its actual throughput ($Actual$) and expected throughput ($Expected$) between α and β where $Actual = W/D$ and $Expected = W/d$. Then, as noted in Section 2, we calculate $Diff$ as follows:

$$Diff = (Expected - Actual) \cdot d = W\frac{(D-d)}{D}, \qquad (2)$$

and interpret $Diff$ as the number of packets in the queue, i.e., $Diff \approx \bar{q}_v$.

Let $W_{v_{max}}$ and $W_{r_{max}}$ denote the window sizes of Vegas and Reno when the queue is full, respectively. Since Vegas keeps $q_v \approx \bar{q}_v$ at all time and $Diff \approx \bar{q}_v$, we have

$$W_{v_{max}}\frac{D-d}{D} = \bar{q}_v,$$

or equivalently,

$$\begin{aligned} W_{v_{max}} &= \bar{q}_v\frac{D}{D-d} = \bar{q}_v\frac{\tau + B/\mu}{(B-1)/\mu} \\ &\simeq \bar{q}_v\frac{\tau + B/\mu}{B/\mu}, \quad \text{if } d \simeq \tau, \end{aligned} \qquad (3)$$

where $d = \tau + 1/\mu$ and $D = \tau + B/\mu$ are the values of the BaseRTT delay and actual RTT delay of Vegas when the queue is full, respectively. Therefore, the window size of Reno when the queue is full is given by

$$W_{r_{max}} = B + \mu\tau - W_{v_{max}} \quad \text{if } d \simeq \tau. \qquad (4)$$

After the queue is full, the probability that a Reno packet will be dropped is

$$p_r = \frac{q_{r_{max}}}{q_{r_{max}} + q_{v_{max}}} = \frac{B - \bar{q}_v}{B} \qquad (5)$$

and the probability that a Vegas packet will be dropped is

$$p_v = \frac{q_{v_{max}}}{q_{r_{max}} + q_{v_{max}}} = \frac{\bar{q}_v}{B}. \quad (6)$$

We now consider two approximations for setting α and β. Approach 1: We assume that the packets dropped are only from the Reno connection and consider only the fast retransmit of Reno while ignoring its timeout mechanism. Therefore, if a Reno packet is dropped, then the Reno window size $W_{r_{max}}$ drops to $\frac{W_{r_{max}}}{2}$. Approach 2: We also consider the case of a Vegas packet being dropped. Hence, if a Reno packet is dropped, its window size evolves the same way as in Approach 1, else if a Vegas packet is dropped, then the Reno window size increases by 1.

3.1.1 Approach 1

Figure 1 shows a simulation of one Reno connection and one Vegas connection at steady state. From this figure, the average window size of the Reno connection ($\bar{W}_r$) is

$$\bar{W}_r = \frac{W_{r_{max}} + \frac{W_{r_{max}}}{2}}{2} = \frac{3}{4} W_{r_{max}}. \quad (7)$$

By definition, the average queue size of the Reno connection is the fraction of $\bar{W}_r$ that is buffered in the queue, i.e.,

$$\bar{q}_r = \bar{W}_r \frac{B}{B + \mu\tau}. \quad (8)$$

Substituting (7) into (8) gives

$$\bar{q}_r = W_{r_{max}} \frac{3B}{4(B + \mu\tau)}. \quad (9)$$

Substituting (9) into (1), the ratio of the throughputs is

$$\begin{aligned} \frac{\lambda_v}{\lambda_r} &= \frac{4\bar{q}_v(B + \mu\tau)}{W_{r_{max}} 3B} \\ &= \frac{4\bar{q}_v}{3(B - \bar{q}_v)}, \end{aligned} \quad (10)$$

where (10) follows from (4) and (3). Now, by setting $\frac{\lambda_v}{\lambda_r} = 1$ and solving for $\bar{q}_v$, we get

$$\bar{q}_v = \frac{3}{7} B.$$

Thus, we set α and β to the following values:

$$\alpha = \lfloor \frac{3}{7} B - 1 \rfloor \quad \text{and} \quad \beta = \lfloor \frac{3}{7} B \rfloor.$$

The motivation for setting α to be one less than β is twofold. First, setting $\alpha = \beta$ introduces stability problems [2], i.e., the congestion window oscillates around the equilibrium value. Second, setting α and β too far apart creates a larger stability region than needed, resulting in connections that can converge to opposite ends of the stability region, thus affecting fairness [2, 9].

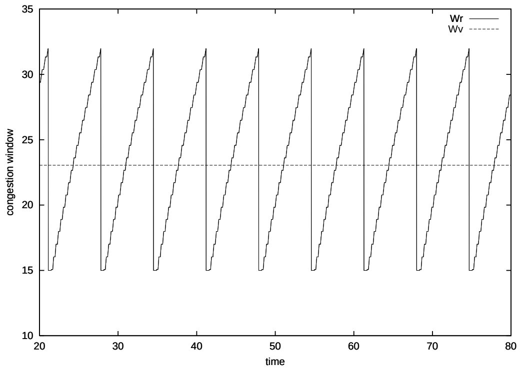

Figure 1. Evolution of the Congestion Window

3.1.2 Approach 2

If a Reno or Vegas packet is dropped, the window size of Reno $\hat{W}_r$ becomes

$$\begin{aligned} \hat{W}_r &= \frac{W_{r_{max}}}{2} p_r + (W_{r_{max}} + 1) p_v \\ &\simeq \frac{W_{r_{max}}}{2} p_r + W_{r_{max}} p_v, \end{aligned} \quad (11)$$

where we assume that $W_{r_{max}} \gg 1$ and that p_r and p_v are given by (5) and (6). From (11), we then approximate the average window size of Reno ($\bar{W}_r$) by

$$\bar{W}_r = \frac{W_{r_{max}} + \hat{W}_r}{2} = W_{r_{max}} \frac{3B + \bar{q}_v}{4B}. \quad (12)$$

As in Section 3.1.1 but with (8) and (12),

$$\bar{q}_r = \bar{W}_r \frac{B}{B + \mu\tau} = W_{r_{max}} \frac{B(3B + \bar{q}_v)}{4B(B + \mu\tau)}. \quad (13)$$

Hence, substituting (13) into (1) gives

$$\frac{\lambda_v}{\lambda_r} = \frac{4\bar{q}_v B(B + \mu\tau)}{W_{r_{max}} B(3B + \bar{q}_v)} = \frac{4\bar{q}_v B}{3B^2 - 2\bar{q}_v B - \bar{q}_v^2}, \quad (14)$$

where (14) follows from (4) and (3). To ensure compatibility between Reno and Vegas, we again set $\lambda_v/\lambda_r = 1$ and solve for $\bar{q}_v$, which yields $\bar{q}_v \simeq 0.4641B$. Therefore,

$$\alpha = \lfloor 0.4641B - 1 \rfloor \quad \text{and} \quad \beta = \lfloor 0.4641B \rfloor.$$

3.2 Analysis of Multiple Connections

In this section, we generalize our two-connection analysis to deal with N_v Vegas connections and N_r Reno connections sharing a bottleneck link of μ packet/s. Let every connection have a round-trip propagation delay of τ seconds while sharing a bottleneck buffer size of B packets. To ensure compatibility between each connection, each connection must equally share the bottleneck buffer, or equivalently, the average queue size of each connection must be identical. We also assume that the connections are synchronous, i.e., at any given instant in time, each Vegas connection i has queue size $q_{v_i} \approx \bar{q}_v \leq \beta \;\forall\; i = 1, 2, \ldots, N_v$ and each Reno connection j has the same average queue size $\bar{q}_r \;\forall\; j = 1, 2, \ldots, N_r$. Thus, Vegas should set its α and β parameters so that $\bar{q}_v/\bar{q}_r \approx 1$.

When the queue is full, the window size of Vegas $W_{v_{max}}$, derived from (2), is

$$\begin{aligned} W_{v_{max}} &= \bar{q}_v \frac{D}{D-d} \\ &\simeq \bar{q}_v \frac{\tau + B/\mu}{B/\mu}, \quad \text{if } d \simeq \tau, \end{aligned} \tag{15}$$

which is the same as (3). And similar to (4), we have

$$N_r W_{r_{max}} = B + \mu\tau - N_v W_{v_{max}}, \quad \text{if } d \simeq \tau, \tag{16}$$

where $W_{r_{max}}$ is the window size of Reno when the queue is full. The dropped probabilities for Reno and Vegas are respectively

$$p_r = \frac{B - N_v \bar{q}_v}{B} \quad \text{and} \quad p_v = \frac{N_v \bar{q}_v}{B}. \tag{17}$$

3.2.1 Approach 1

Using the same approach as in Section 3.1.1, the average window size ($\bar{W}_r$) and the average queue size ($\bar{q}_r$) of each Reno connection is given by

$$\bar{W} = \frac{W_{r_{max}} + \frac{W_{r_{max}}}{2}}{2} = \frac{3}{4} W_{r_{max}},$$

and

$$\bar{q}_r = \bar{W} \frac{B}{B + \mu\tau} = W_{r_{max}} \frac{3B}{4(B + \mu\tau)}, \tag{18}$$

respectively. Substituting (16) into (18), we get

$$\bar{q}_r = \frac{3(B - N_v \bar{q}_v)}{4N_r}. \tag{19}$$

Hence, the ratio of throughputs is given by

$$\frac{\lambda_v}{\lambda_r} = \frac{4N_r \bar{q}_v}{3(B - N_v \bar{q}_v)}. \tag{20}$$

And again, to ensure compatibility between Reno and Vegas, $\lambda_v/\lambda_r = 1$. Solving for $\bar{q}_v$ gives

$$\bar{q}_v = \frac{3B}{4N_r + 3N_v}.$$

Thus, we set α and β as

$$\alpha = \lfloor \frac{3B}{4N_r + 3N_v} - 1 \rfloor \quad \text{and} \quad \beta = \lfloor \frac{3B}{4N_r + 3N_v} \rfloor.$$

3.2.2 Approach 2

As in Section 3.1.1, the window size of each Reno connection after a dropped packet is given by (11), and each average window size is given by (12). Combining (8), (12), (15), and (16) results in

$$\bar{q}_r = \frac{(B - N_v \bar{q}_v)(3B + \bar{q}_v)}{4N_r B}.$$

Substituting the above equation back into (1) gives

$$\frac{\lambda_v}{\lambda_r} = \frac{4N_r B \bar{q}_v}{(B - N_v \bar{q}_v)(3B + \bar{q}_v)}$$

Setting $\frac{\lambda_v}{\lambda_r} = 1$ helps ensure that the Reno and Vegas connection are compatible. Now, solving for $\bar{q}_v$, we have

$$\bar{q}_v = [\frac{\sqrt{(4N_r + 3N_v - 1)^2 + 12N_r} - (4N_r + 3N_v - 1)}{2N_r}]B,$$

and set the Vegas parameters as follows:

$$\alpha = \lfloor [\frac{\sqrt{(4N_r + 3N_v - 1)^2 + 12N_r} - (4N_r + 3N_v - 1)}{2N_r}]B - 1 \rfloor$$

and

$$\beta = \lfloor [\frac{\sqrt{(4N_r + 3N_v - 1)^2 + 12N_r} - (4N_r + 3N_v - 1)}{2N_r}]B \rfloor.$$

4 Experiments

To verify our observations made through a heuristic analysis of the behavior of Reno and Vegas, we run two sets of simulations using the discrete-event simulation *ns*, version 2.1b8a [14].

4.1 Network Topologies & Parameters

We consider two networks based on the generic topology shown in Figure 2 and parametric details in Table 1. The first network comes from [13] to use as a point of reference. The second network models the grid [7] between Los Alamos and Sandia National Laboratories.

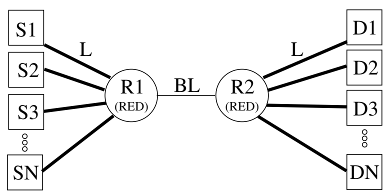

Figure 2. Generic Topology

Network		Reference	Grid
Node Pairs (N)		2	50
Links L	Bandwidth	10 Mb/s	100 Mb/s
	Delay	4 ms	1 ms
Link BL	Bandwidth	1.55 Mb/s	155 Mb/s
	Delay	4 ms	1.5 ms

Table 1. Parameters of Simulated Networks

For all our simulations, each connection starts an FTP session at time 0 and ends at 200 seconds with the packet size fixed at the standard 1500-byte Ethernet size. As in [13], we measure the number of ACKs that each connection receives where ACK_R and ACK_V represent the number of ACKs from Reno and Vegas, respectively. In the general case of N_r Reno connections and N_v Vegas connections, ACK_R (resp. ACK_V) show the average number of ACKs over all N_r Reno (resp. N_v Vegas) connections.

4.2 Reference Network

We use the same network that [13] did in order to (i) confirm their results and (ii) confirm our analytic observations made in Section 3. In confirming our analytic results, we show that Reno and Vegas are indeed compatible and that overall network performance improves by distributing bandwidth more evenly across *all* connections while still maintaining high overall throughput.

The remainder of this section examines the performance of our analytic heuristics in the case of two connections competing head-to-head and in the case of multiple connections competing. Within each case, we test our analytic heuristics of Approach 1 and 2.

4.2.1 One Reno vs. One Vegas (Head-to-Head)

The experimental set-up here is similar to [13]. In general, our results confirm the conclusions drawn by [13]. That is, as the buffer size increases, Reno uses more of the buffer to steal bandwidth from Vegas as Vegas is throttled by the "misconfigured" α and β parameters. In fact, Table 2 shows that in no case does Vegas get better throughput than Reno and that as the buffer size increases up to 100, Reno achieves 21 times higher throughput than Vegas.

Buffer	ACK_R	ACK_V	$\frac{ACK_R}{ACK_V}$
7	16,555	9,131	1.813
10	20,763	4,972	4.176
15	18,581	7,209	2.577
25	20,980	4,823	4.350
50	23,678	2,096	11.297
100	24,544	1,179	20.818

Table 2. Reno vs. Default (Mis)configured Vegas ($\alpha = 1, \beta = 3$)

Tables 3 and 4 illustrate that with the proper configuration of α and β, Vegas competes well with Reno. For the smaller buffer sizes, Vegas performs almost twice as well; this behavior occurs due to the aggressive nature of Reno's congestion control, i.e., always increasing its window even though the buffer space is small. As the buffer sizes get larger and Vegas's α and β parameters adapt accordingly, we see that Reno and Vegas each get their fair share of bandwidth, i.e., the "fairness ratio" $ACK_R/ACK_V \approx 1$. Thus, these simulations confirm that our analytic heuristics from Sections 3.1.1 and 3.1.2 enable Reno and Vegas to be compatible with each other. In addition, the heuristics enhance overall network performance by distributing bandwidth more evenly across Reno and Vegas while keeping overall network throughput high.

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
7	9,006	16,678	3	0.540
10	12,241	13,444	4	0.910
15	12,471	13,330	6	0.936
25	12,868	12,937	10	0.995
50	12,688	13,091	21	0.969
100	13,143	12,585	42	1.044

Table 3. Reno vs. Vegas with $\alpha = \lfloor \frac{3}{7}B - 1 \rfloor$, $\beta = \lfloor \frac{3}{7}B \rfloor$ (Approach 1)

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
7	9,006	16,678	3	0.540
10	12,241	13,444	4	0.910
15	12,471	13,330	6	0.936
25	12,868	12,937	10	0.995
50	11,747	14,032	23	0.837
100	12,233	13,495	46	0.906

Table 4. Reno vs. Vegas with $\alpha = \lfloor 0.4641B - 1 \rfloor$, $\beta = \lfloor 0.4641B \rfloor$ (Approach 2)

4.2.2 Multiple Reno vs. Multiple Vegas (Fixed Buffer)

The set of tests performed here are identical to those in Section 4.2.1 with two exceptions: (i) There are 10 different TCP connections vying for network bandwidth. (ii) The buffer size is fixed at 250 packets.

Using the formulas for α and β from Sections 3.2.1 and 3.2.2, we again demonstrate that Vegas can be properly configured to be compatible with Reno. Tables 5 and 6 show that the fairness ratio (i.e., ACK_R/ACK_V) is close to one in all cases. This is in stark contrast to Table 2 where the fairness ratio is 20.818, meaning that Reno gets that many times more bandwidth than Vegas.

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
2	8	3,153	2,407	23	1.309
4	6	2,500	2,600	22	0.962
6	4	2,645	2,427	20	1.090
8	2	2,542	2,622	19	0.969

Table 5. N_r Reno vs. N_v Vegas with α and β via Approach 1 and $B = 250$

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
2	8	2,952	2,457	24	1.201
4	6	2,500	2,600	22	0.962
6	4	2,584	2,517	21	1.027
8	2	2,542	2,622	19	0.969

Table 6. N_r Reno vs. N_v Vegas with α and β via Approach 2 and $B = 250$

4.2.3 Multiple Reno vs. Multiple Vegas

In this set of tests, we fix the different types and number of connections while varying the size of the buffer. And as evidenced by Tables 7 and 8, we again demonstrate that with a properly configured Vegas, Vegas is compatible with Reno. Except for the first row of each table where Vegas beats Reno, Reno and Vegas each get their fair share of bandwidth as indicated by the fairness ratio of approximately one in each row of Tables 7 and 8.

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
100	1,746	3,399	8	0.514
200	2,571	2,556	17	1.006
300	2,690	2,416	25	1.113
400	2,648	2,437	34	1.087
500	2,744	2,321	42	1.182

Table 7. Five Reno vs. Five Vegas with α and β via Approach 1

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
100	1,746	3,399	8	0.514
200	2,571	2,556	17	1.006
300	2,621	2,486	26	1.054
400	2,648	2,437	34	1.087
500	2,719	2,346	43	1.159

Table 8. Five Reno vs. Five Vegas with α and β via Approach 2

4.3 Grid Network

The experimental results for head-to-head competition on the grid network are shown in Tables 9 and 10. We observe that all the fairness ratios lie in the interval [0.703,1.465]. These results are in stark contrast to Table 2 where Reno is run against the default Vegas configuration, resulting in fairness ratios that lie in the interval [1.813,20.818].

For some cases, e.g., buffer size = 300 in Table 9, Reno achieves better throughput than Vegas. Why does this happen? Before steady state is reached, a Vegas packet is dropped, and a subsequent timeout occurs, thus providing Reno the opportunity to aggressively grab network bandwidth than Vegas relinquishes. The same reasoning can be used to explain Table 10 when the buffer size is 200.

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
25	1,088,323	1,279,521	10	0.851
50	1,132,451	1,377,165	21	0.822
100	1,237,816	1,337,189	42	0.926
200	1,190,797	1,310,759	85	0.908
300	1,566,684	1,004,753	128	1.459
400	1,543,070	1,322,325	171	1.167

Table 9. Reno vs. Vegas with $\alpha = \lfloor \frac{3}{7}B - 1 \rfloor$, $\beta = \lfloor \frac{3}{7}B \rfloor$ (Approach 1)

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
25	1,077,134	1,448,115	11	0.744
50	1,051,922	1,495,565	23	0.703
100	1,080,863	1,489,905	46	0.725
200	1,506,751	1,028,086	92	1.465
300	1,455,002	1,115,923	139	1.303
400	1,333,429	1,083,871	185	1.230

Table 10. Reno vs. Vegas with $\alpha = \lfloor 0.4641B - 1 \rfloor$, $\beta = \lfloor 0.4641B \rfloor$ (Approach 2)

Tables 11 through 14 show the experimental results for multiple TCP connections and different buffer sizes. The results in these tables closely verify the analytic heuristics that we developed in Section 3. In fact, with the exception of one data point, all the fairness ratios lie within 5% of the ideal fairness ratio of one, i.e., [0.955,1.068].

5 Fairness of TCP Reno vs. TCP Vegas

As shown in [5, 6, 12], Reno favors connections with shorter delays. In contrast, Mo et al. [13] demonstrate that Vegas does not suffer from this delay bias via a closed, fluid model and simulation; however, they do not consider the fairness between Reno and Vegas because of the demonstrated incompatibility of Reno and Vegas (in its default configuration, i.e,. $\alpha = 1$ and $\beta = 3$). By using the closed, fluid-model approximation in [13], we graphically illustrate the fairness of Reno and Vegas at steady state in the bottleneck link.

In the steady state, let $W_r(t)$ and $W_v(t)$ be the window sizes of Reno and Vegas at time t, respectively. By assum-

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
2	8	248,824	260,565	93	0.955
4	6	258,381	257,747	88	1.002
6	4	256,206	260,656	83	0.983
8	2	259,968	244,561	78	1.063

Table 11. N_r Reno vs. N_v Vegas with α and β via Approach 1 and $B = 1,000$

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
2	8	223,790	266,824	96	0.839
4	6	253,251	261,166	89	0.970
6	4	255,889	261,131	84	0.980
8	2	260,206	243,617	79	1.068

Table 12. N_r Reno vs. N_v Vegas with α and β via Approach 2 and $B = 1,000$

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
500	259,967	256,245	42	1.015
750	257,939	258,098	64	0.999
1,000	257,946	258,247	85	0.999
1,250	255,283	259,859	107	0.982
1,500	255,258	259,820	128	0.982

Table 13. Five Reno vs. Five Vegas with α and β via Approach 1

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
500	257,894	258,318	43	0.998
750	255,366	260,672	65	0.979
1,000	253,942	262,251	87	0.968
1,250	253,966	261,168	108	0.972
1,500	252,422	262,657	130	0.961

Table 14. Five Reno and Five Vegas with α and β by Approach 2

ing that the throughput and the queue size of each connection is relatively constant, we have

$$W_i(t) = q_i(t) + \lambda_i(t) \cdot d_i, \quad i = r, v, \tag{21}$$

where $\lambda_i(t), i = r, v$, $q_i(t), i = r, v$ and $d_i, i = r, v$ are the throughput, queue size, and BaseRTT, respectively, for Reno and Vegas connections. Moreover, we also have

$$\lambda_i(t) = \frac{W_i(t)}{p(t) + d_i}, \quad i = r, v, \tag{22}$$

where $p(t)$ denotes the queueing delay of the bottleneck link at time t. Let μ and B denote the capacity and the buffer size of the bottleneck link. With large enough B, we assume that the link is fully utilized, so we have

$$\frac{W_r(t)}{p(t) + d_r} + \frac{W_v(t)}{p(t) + d_v} = \mu. \tag{23}$$

For simplicity, we let $d_r = d_v = d$. By (23),

$$p(t) = \frac{W_r + W_v - \mu d}{\mu}. \tag{24}$$

Hence, from (21), (22), and (24) and given the window size of Reno is W_r and the queue size of Vegas is q_v; we can compute the window size of Vegas (W_v) at steady state:

$$\begin{aligned} W_v(t, q_v(t)) &= \frac{\mu d + q_v(t) - W_r}{2} \\ &+ \frac{\sqrt{(\mu d + q_v(t) - W_r)^2 + 4 q_v(t) W_r}}{2} \end{aligned}$$

Similarly, given W_v and q_r, we also have

$$\begin{aligned} W_r(t, q_r(t)) &= \frac{\mu d + q_r(t) - W_v}{2} \\ &+ \frac{\sqrt{(\mu d + q_r(t) - W_v)^2 + 4 q_r(t) W_v}}{2} \end{aligned}$$

In order to show the stability region of Reno and Vegas, we first note that Vegas keeps its q_v between α and β; hence, its window size will be stable if it lies between $W_v(t, \alpha)$ and $W_v(t, \beta)$. Furthermore, the window size of Reno can be varied as a function of $q_r(t)$ where the two extreme cases are $q_r(t) = 0$ and $q_r(t) = B - q_v(t)$. Therefore, by (25) and the fact that $q_r(t) = W_r(t) \cdot \frac{B}{\mu d + B}$, $W_r(t) \in [\mu d - W_v(t), \mu d + B - W_v(t)]$ for any given $W_v(t)$. However, when Reno incurs a packet loss, its window size is halved. If multiple losses do not occur, Reno resumes its linear increase up to the maximum value; therefore, we assume that the size of the Reno window at the stability region lies between $\max(\frac{\mu d + B - W_v(t)}{2}, \mu d - W_v(t))$ and $\mu d + B - W_v(t)$. For convenience, we denote $Rmax = \mu d + B - W_v(t)$ and $Rmin = \mu d - W_v(t)$

As stated above, we plot the graph of the stability region for one Reno connection versus one Vegas connection over the "Reference Network" with $B = 10$. Figure 3 shows the stability region of two Vegas connections where the line $alpha_i$ and $beta_i$, $i = 1, 2$ denote $\{(W_1, W_2) | W_i - \lambda_i d = \alpha\}$ and $\{(W_1, W_2) | W_i - \lambda_i d = \beta\}$, respectively. Furthermore, the fairness line is the line such that the window sizes of both connections are the same, i.e., $\{(W_1, W_2) | \lambda_1 = \lambda_2\}$. In this case, the fairness line passes right through the stability region (also referred to as the convergence region); hence, by using only Vegas in the network, all connections get fair throughput. Figure 4 shows one Reno connection versus one Vegas connection with default parameters of $\alpha = 1$ and $\beta = 3$. In this case, the convergence region of Reno and Vegas lies between the lines $Rmax$, $Rmax/2$, $alpha$ and $beta$. However, the fairness line hardly passes through the convergence region; hence, it introduces unfairness, and Reno's throughput is higher than Vegas's. However, when both α and β are set appropriately, Figure 5 shows that the fairness line clearly goes through the middle of the convergence region.

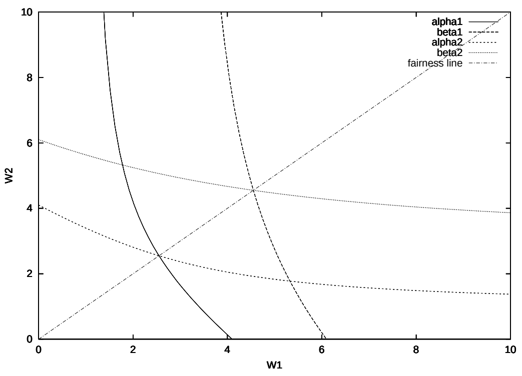

Figure 3. Two TCP Vegas Windows with $\alpha = 1$ and $\beta = 3$

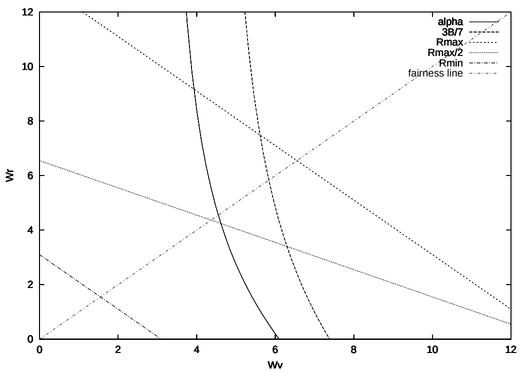

Figure 5. TCP Reno (W_r) and TCP Vegas (W_v) with $B = 10$, $\alpha = 3$, $\beta = 4$, and $3B/7 = 4.285$

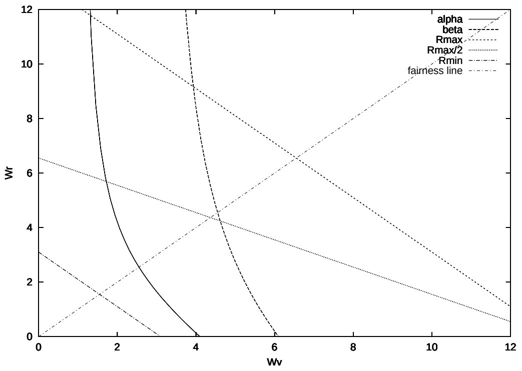

Figure 4. TCP Reno (W_r) and TCP Vegas (W_v) Windows with $B = 10$, $\alpha = 1$, and $\beta = 3$

6 Conclusion

Prior research demonstrated the incompatibility of TCP Reno and TCP Vegas. In this paper, we showed that the incompatibility of Reno and Vegas is *not* inherent to their congestion-control algorithms but an artifact of misconfiguring Vegas's congestion-avoidance parameters.

In particular, we showed how inappropriate the default values of α and β in Vegas are (when in competition with Reno), explained the relationship of these parameters to variations in network performance, and demonstrated how to set the parameters appropriately so that Reno and Vegas are compatible.

References

[1] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas:Emulation and Experiment. In *Proc. of ACM SIGCOMM 1995*, August 1995.

[2] C. Boutremans and J. L. Boudec. A Note on the Fairness of TCP Vegas. In *Proc. of International Zurich Seminar on Broadband Communications*, February 2000.

[3] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communication*, October 1995.

[4] W. Feng and P. Tinnakornsrisuphap. The Failure of TCP in High-Performance Computational Grids. In *Proc. of SC 2000: High-Performance Networking and Computing Conf.*, November 2000.

[5] S. Floyd and V. Jacobson. Connection with Multiple Congested Gateways in Packet-Switched Networks, Part 1: One-Way Traffic. *ACM CCR*, August 1991.

[6] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[7] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[8] G. Hasegawa, K. Kurata, and M. Murata. Analysis and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet. In *Proc. of IEEE ICNP2000*, November 2000.

[9] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and Stability of Congestion Control Mechanisms of TCP. In *Proc. of IEEE INFOCOM 1999*, March 1999.

[10] V. Jacobson. Congestion Avoidance and Control. *ACM CCR*, August 1988.

[11] V. Jacobson. Modified TCP Congestion Avoidance Algorithm. Technical report, Technical report, April 1990.

[12] A. Mankin. Random Drop Congestion Control. In *Proc. of ACM SIGCOMM 1990*, September 1990.

[13] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *Proc. of IEEE INFOCOM 1999*, March 1999.

[14] ns. Network Simulator, version 2.1b8a. http://www.isi.edu/nsnam/ns.

[15] A. Veres and M. Boda. The Chaotic Nature of TCP Congestion Control. In *Proc. of IEEE INFOCOM 2000*, March 2000.