Proceedings of the 9th International Symposium on High Performance
Distributed Computing (HPDC'01), August, 2001.

A Case for TCP Vegas in High-Performance Computational Grids*

Eric Weigle† and Wu-chun Feng†‡

† *Research & Development in Advanced Network Technology (RADIANT)*
Los Alamos National Laboratory
Los Alamos, NM 87545

‡ *Department of Computer & Information Science*
The Ohio State University
Columbus, OH 43210

{*ehw, feng*}*@lanl.gov*

Abstract

Computational grids such as the Information Power Grid [1], Particle Physics Data Grid [2], and Earth System Grid [3] depend on TCP to provide reliable communication between nodes across a wide-area network (WAN). Of the available TCP implementations, TCP Reno and its variants are the most widely deployed; however, Reno's performance in computational grids is mediocre at best.

Due to conflicting results in the evaluation of TCP implementations [4, 5, 6, 7, 8, 9, 10, 11, 12, 13], we present a detailed simulation study that unifies the conflicting results and demonstrates the limitations of earlier work. We focus on the two most debated versions of TCP — Reno and Vegas. Using real traffic distributions, we show that Vegas performs well over modern high-performance links and better than Reno with the proper selection of the Vegas parameters α and β. Our results exhibit ways to significantly enhance the performance of distributed computational grids that rely on TCP.

Keywords: computational grid, distributed computing, networking, TCP, Reno, Vegas.

1. Introduction

Studying congestion-control algorithms in TCP is problematic due to their complexity and inconsistencies between different implementations of the same algorithm. Recent work has shown that TCP Reno's congestion control induces chaotic behavior in a network [13, 14, 15], thus adversely affecting all network performance. Some argue that Vegas is more stable [5, 6, 9] while others argue that Reno performs better [11, 16]. Most agree that Vegas performs better under certain circumstances.

Unfortunately, many prior studies on TCP suffer from unrealistic simplifications. Here we extend prior research in three areas: (1) using more realistic traffic- and flow-generation models, (2) using real-world networks, and (3) showing how to manipulate the parameters α and β in TCP Vegas to enhance its performance.

Our primary goal is to show that there exists an α, β pair such that Vegas outperforms Reno in direct competition. While most researchers agree that networks with all Vegas flows perform better than networks with all Reno flows, they also claim that incremental adoption of Vegas in a predominantly Reno environment is doomed because the performance of Vegas in such an environment is pitiful. To address this claim, we demonstrate that overall network performance actually improves with the addition of Vegas flows (with judiciously set values for α and β) competing head-to-head with Reno flows. The performance improvements we exhibit justify the use of TCP Vegas in a computational grid and lend credence to the idea that incremental adoption of Vegas is possible.

To this end, we first present requisite background information. Next, we discuss our network topologies and traffic models. Then, we present the results and analyses of our experiments. We close with related work and our conclusions.

*This work was supported by the U.S. Dept. of Energy through Los Alamos National Laboratory contract W-7405-ENG-36. This paper is LA-UR 01-3420.

2. Background

To create useful models for our simulations, we must understand the diversity of TCP algorithms, the effects of RED routers, and the simulation tool that we use, *ns*.

2.1. TCP Algorithms

The various TCP algorithms have different congestion-control mechanisms, and implementations of these algorithms may or may not implement features such as selective acknowledgements or window scaling. For our baseline, we use TCP Reno with the following standard features: a congestion window obeying additive increase by one/multiplicative decrease by one half, slow start, initial window of one, fast retransmit, fast recovery, and window scaling.

The TCP Vegas algorithm [6] extends TCP Reno by trying to avoid rather than react to congestion. When the congestion window increases in size, the expected sending rate (ESR) increases as well. But if the actual sending rate (ASR) stays roughly the same, then there is *not* enough bandwidth available to send at ESR. Thus, any increase in the size of the congestion window will only fill buffer space in the network rather than improve performance. Vegas attempts to detect this phenomenon and avoid congestion by adjusting the congestion-window size, and hence ESR, to adapt to the available bandwidth.

To adjust the window size appropriately, Vegas uses two threshold values, α and β. These values are traditionally set to 1 and 3, respectively, and control the operation of Vegas as follows:

- Let $Diff = ESR - ASR$.
- If $Diff < \alpha$, increase the congestion window linearly during the next round-trip time (RTT).
- If $Diff > \beta$, decrease the window linearly during the next RTT.
- Otherwise, do not change the congestion window.

Conceptually, $\alpha = 1$ and $\beta = 3$ implies that each Vegas flow tries to keep at least one packet but no more than three packets queued in the network. Selecting these parameters holds an implicit tradeoff between network utilization, goodput, and fairness; by using only the default settings for these parameters, prior work inadvertently favored Reno over Vegas [11]. Here, we make these choices explicit.

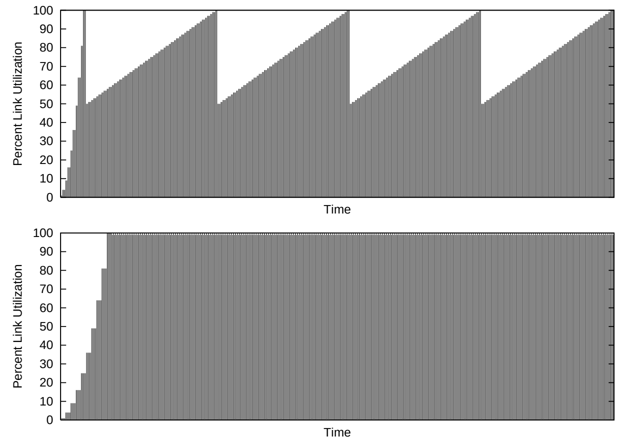

Figure 1. Ideal link utilization of TCP Reno and Vegas

Figure 1 shows the behavior of Reno and Vegas with respect to link utilization in the simplest case, i.e., a single Reno or Vegas flow between two hosts with no network buffering (there are no routers) and the possibility of buffer overflows (loss) on each host. One problem with TCP Reno is quite apparent; its congestion-control algorithm is inherently over-aggressive. Reno probes the network state by inducing packet loss, resulting in a multiplicative decrease (1/2) in the amount of data that it is allowed to send in the subsequent round-trip time (RTT) interval. It then enters an additive-increase phase to again probe the network state. As soon as Reno reaches the optimal bandwidth; it again passes it, drops a packet, and halves its sending rate. While Reno continues slowly climbing toward the optimal bandwidth, Vegas continues to send data near the optimal bandwidth.

2.2. Random Early Detection

To enhance TCP performance, Floyd and Jacobson [17] introduced random early detection (RED) gateways to detect incipient congestion. RED gateways maintain a weighted average of the queue length ($length$), a minimum and maximum threshold (RED_{min} and RED_{max}, respectively), and an early-drop probability P. Packets are then queued as follows:

- If ($length < RED_{min}$), queue all packets.
- If ($length > RED_{min}$ and $length < RED_{max}$), drop packets with probability P.
- If ($length > RED_{max}$), drop all packets.

One modification is to use a variable, rather than a constant, drop probability P in the second case. Then P ranges from 0 to 1 as the average queue size ranges from RED_{min} to RED_{max}; a simple linear normalizing formula can be used for this purpose. We use this variation in our simulations, with the effective formula:

$$P = \frac{length - RED_{min}}{RED_{max} - RED_{min}}$$

RED can improve fairness and overall network performance [17], and most routers in the current Internet implement it. As most distributed computational grids will be connected via sections of the Internet, this behavior must be considered.

2.3. Simulation using *ns*

For our experiments, we use the discrete-event simulator *ns*, version 2.1b7a [18]. *ns* implements much of what we wish to study; however, we have extended its functionality. First, we introduced new code to create the traffic distributions discussed in Section 3. Second, we added instrumentation to the existing classes for TCP Reno and Vegas so that the effects of our new code could be monitored more precisely. We verified the correctness of our changes by running the test suites provided with *ns* as well as several of our own programs. We verified the correctness of our traffic generator by manual analysis of packet traces.

3. Experiments

Our experiments can be summarized as a search of the feature space:

$$network \times traffic \times \alpha \times \beta.$$

Given only three topologies and three traffic distributions, we can perform a brute-force search for α, β values under which Vegas performs well. (The above features obviously do not capture all possible parameters, but they are the most representative.) Given the large amount of data that a brute-force search generates, we will derive simple heuristics to guide future exploration and protocol tuning. In contrast to other work [11, 16], our experiments show that with intelligent choices for α and β, TCP Vegas outperforms Reno in practically all circumstances.

3.1. Network Topologies and Parameters

We consider four networks; the topology of each is based on the generic "butterfly" topology of Figure 2 with details given in Table 1. The first network is that given in [11], while the second and third networks model current and future computational grids between Los Alamos and Sandia National Laboratories, and the fourth shows performance over a WAN grid with a 7.8-MB bandwidth-delay product. Our results, in addition to [14], will show how poorly the current TCP protocols will scale to next-generation computational grids.

Table 1. Parameters of Simulated Networks

	Network	1	2	3	4
	Nodes (N)	2	50	100	100
Links	Bandwidth	10	100	1000	1000
L	Delay	4	1	1	15
Link	Bandwidth	1.55	155	622	622
BL	Delay	4	1.5	1.5	20
Routers	Buffer	4-50	33554	44739	44739
R1,	RED_{min}	3-10	6711	8948	8948
R2	RED_{max}	6-20	26843	35791	35791

Nodes (N) is the number of source or destination hosts, meaning that the total number of nodes in the simulation is $2N + 2$ (N source nodes, N destination nodes, and 2 router nodes). Bandwidth is in megabits per second (Mbps or 1,000,000 bps); Delay is in milliseconds (ms); and Buffer sizes and thresholds are in packets (pkts). We simulate standard 1500-Byte ethernet packets. Note that when abbreviated size units are given, we follow the convention that a capital 'B' refers to Bytes, while a lowercase 'b' refers to bits.

3.2. Network Traffic

By default, one traffic flow (a traffic generator and an associated TCP Reno or TCP Vegas transport protocol) runs on each source node (S_i) and transmits to its respective destination node (D_i). When results with 'reverse path' or 'two-way' traffic are given, this means that each destination node D_i also acts as a source node, sending traffic along the reverse path to source node S_i. Two-way traffic doubles the total number of flows and introduces two behaviors: ack loss and compression. When reverse path traffic is introduced, ack packets must compete for bandwidth and queue space. This means that some ack packets may be dropped as queues become full (loss), and their inter-packet timing may be altered as they are queued and later released in a burst (compression). In some situations, these behaviors can significantly affect results [19], and we present them when this is the case.

For our experiments, there are always equal numbers of Reno and Vegas flows; so with one-way traffic we have $\frac{N}{2}$ of each type, with two-way traffic we have N of each type.

Traffic libraries (such as tcplib [20]) are available to drive simulations, but they are fundamentally flawed. Such traces were made using the `tcpdump` tool available on Unix platforms, meaning that they were captured *after* having passed through TCP. Passing these packet traces back through TCP again is meaningless. Furthermore, the available traces are several years old and are not representative of current traffic. None capture traffic in a computational grid. For all these reasons, we have created our own generators.

As a first approximation, network traffic is typically modeled as an infinite file transfer. The next approximation adds variation over packet transmission times, explicitly with an exponential on/off traffic source or implicitly by setting up background traffic [9]. The most realistic traffic models contain variations in inter-flow times, flow lengths, and packet sizes. We use all the above features of real traffic except for variation in the packet size, which *ns* does not adequately support. Prior research has used only the first overly simplistic traffic type or has neglected to consider bidirectional traffic, which can seriously affect results [19]. We refer to [21] for further background on our traffic models, which are summarized in Tables 2 and 3.

Traffic Type	Flow Arrival	Flow Length	Packet Spacing
Infinite File	all at time 0	∞	constant
Interactive	Poisson	Pareto	Pareto on/off
No Packet Spacing	Poisson	Pareto	n/a

Table 2. Summary of traffic models

For infinite file traffic, requests to transfer a packet are made every $link\_speed/packet\_size$ seconds for the duration of the simulation. We also refer to this traffic as "constant bit rate" or "CBR". Applications transferring large data sets generate this type of traffic.

Interactive traffic uses a Pareto on/off distribution for packet spacing; an exponential on/off distribution was also tested but produced no significant differences. We also refer to this model as "Pareto". Applications for data set visualization generate this type of traffic.

Cluster and grid communication patterns depend heavily on the algorithms being executed [22]. Our model assumes that traffic between nodes is frequent and mostly small, with less frequent larger transmissions. This is similar to the Interactive model except that packet spacing is ignored; all packets for a given flow (data at an upper level, such as an MPI message) are presented to TCP at the same time and should be transmitted as soon as possible. We refer to this model as the No Packet Spacing (NPS) model, as it is the same as Pareto traffic with that exception. Scientific applications which pass messages between nodes and perform checkpoint operations generate this type of traffic.

The values in Table 3 are representative of traffic on the networks in Table 1. Different networks use different values: † represents the values for network 1 at the lower end of the range and the remaining networks at the higher end while the lower values marked with ‡ represent network 2, the higher values for 3 and 4. (Note that direct numerical comparisons of bandwidth acquired between traffic types is invalid, as each model attempts to send different amounts of data.)

At time zero, the start of Pareto and NPS flows is delayed by the inter-flow time on average (as though flows

Traffic Type	Parameter	Values
Infinite File	Rate	10-100 Mb†
Pareto&NPS	Mean inter-flow time	0.5 s
Pareto&NPS	Mean flow length	10-100 MB‡
Pareto&NPS	Flow length shape	1.5
Pareto&NPS	ON rate	100 Mb-1 Gb‡
Pareto	Mean ON time	30 ms
Pareto	Mean OFF time	70 ms
Pareto	ON/OFF shape	1.5

Table 3. Traffic parameters for distributions

had just terminated on all nodes). Thus, it takes a few seconds in simulation time to achieve a network state as would be found in a live network. Simulations were run for 200 seconds to minimize this factor and to ensure that the differences we report between flows are not due to transient conditions (simulations for longer periods reveal no significant differences from the following results).

4. Results and Analysis

We now present our results, focusing on the total amount of reliably transmitted data (as measured by the number of ACK packets) and the amount of data lost (as measured by the number of retransmitted packets). From these metrics, we can then calculate link utilization, goodput, or fairness.

In the following figures, each pair of bars represents the aggregate bandwidth or loss results from a single simulation; that of all TCP Reno flows (white) and all TCP Vegas flows (black) as they competed against each other with the given parameters.

4.1. Network 1 with Varied Queue Sizes

First, we look at a simple head-to-head competition of one Reno flow versus one Vegas flow, varying the queue size for our bottleneck link. We use CBR traffic for comparison with the work done in [11].

We are only interested in queue sizes indirectly; Reno aggressively probes the network and uses any available buffer space while Vegas only buffers between α and β packets on average. Varied queue sizes exhibit the effects this difference has on the protocols' performance. By tying α and β to the queue sizes, Vegas is forced to set its congestion window to a value comparable to that of Reno, so Reno has no unfair advantage. For a network with only Vegas flows, router buffer space would be less relevant [6].

To guide our choice of α and β, we initially developed the following heuristic. Assume that Reno uses 75% of the bandwidth on average while Vegas uses 100% (Figure 1) and that the bandwidth acquired corresponds directly to the buffer size utilized. Then, to achieve long-term fairness between Reno and Vegas, the following constraints should be met:

$$0.75 \times B_R = \frac{1}{2} \quad \Rightarrow \quad B_R = \frac{2}{3}$$

and

$$1.0 \times B_V = \frac{1}{2} \quad \Rightarrow \quad B_V = \frac{1}{3}.$$

where B_R and B_V represent the fraction of buffer size for each Reno and Vegas flow, respectively. Note how Reno requires *twice* as much buffering for the same performance. These figures are only valid for this simple case; for more complex cases, 'overlapping' of the sawtooth pattern more effectively utilizes bandwidth, and we will want Reno and Vegas to use approximately the same amount of buffer space.

Of B_R and B_V, we can only control B_V indirectly and cannot control B_R at all. We assume that Reno will use any queue space that Vegas does not and consider α and β to be the minimum and maximum number of packets Vegas tries to keep enqueued. This analysis leads us to the following possible heuristic.

set $\alpha = \lfloor \frac{Queue\ Size}{3} - 1 \rfloor$ and set $\beta = \lfloor \frac{Queue\ Size}{3} \rfloor$

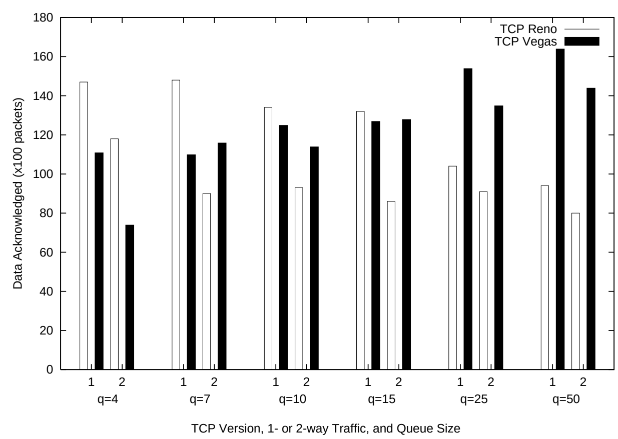

Figure 3. Network 1, variation of queue sizes

The results for this choice are presented in Figure 3. Note that full bottleneck link utilization with one-way traffic would be to send 25,833 1500-Byte ethernet segments over the 1.55-Mbps link during our 200 second simulation. For two-way traffic, full utilization is 25,162 packets given the the 40-Byte ACK packets on the reverse path.

For simple one-way flows, our linear heuristic based on queue size is too timid for queue sizes 10 or below (Reno beats Vegas). For queue sizes between 10 and 20, Reno and Vegas compete well, and for larger queue sizes, Vegas does much better than Reno. Link utilization is above 99% in all cases.

Adding reverse-path traffic slightly decreases the amount of bandwidth each flow acquires (due to the higher overall traffic on the network) and changes the pattern in favor of Vegas. These changes are due to the slight increase in drop rates at the routers (due to 40-Byte ACK packets competing with 1500-Byte reverse-path data packets); as Reno has twice as many packets buffered as Vegas does, Reno is much more likely to lose one of their packets than Vegas. Link utilization varies from 78% to 90%.

The difference between our experimental results and those predicted by our formula (equal performance by Reno and Vegas) is due to several factors. First, we are working in a 200-second simulation with real values for various parameters rather than an infinitely long simulation with infinite bandwidth (where the delimiting factor is *only* queue size). Furthermore, we want the ‘apparent’ or ‘effective’ size (the size below which is it unlikely for a packet to be dropped) rather than the absolute fair queue size. This apparent size is time-dependent, given the fluctuations caused by Reno’s additive-increase, multiplicative-decrease behavior. Note that by slightly changing our heuristic, increasing α and β for queue sizes below 20, Vegas will beat Reno for all queue sizes.

4.2. Network 1 with Varied RED Thresholds

For this set of experiments, we fix the queue size at 25 packets and vary the RED minimum (RED_{min}) and maximum (RED_{max}) thresholds. We use a single Reno and a single Vegas flow, each with CBR traffic, as done in [11].

In [11], the performance of Vegas was shown to drop as RED thresholds were increased. This is due to the fact that the apparent queue size is increasing — Reno will automatically probe the network and exploit this space, but Vegas will not as it is constrained by the default $\alpha = 1$ and $\beta = 3$ values. Consequently, the experiments in [11] are inadvertently “rigged” to favor Reno over Vegas.

To account for the above factors, we view RED_{min} and RED_{max} as the apparent queue size and continue as in the prior experiment but with

$$\alpha = RED_{min} - 2 \text{ and } \beta = RED_{max} - 2.$$

to produce the following results.

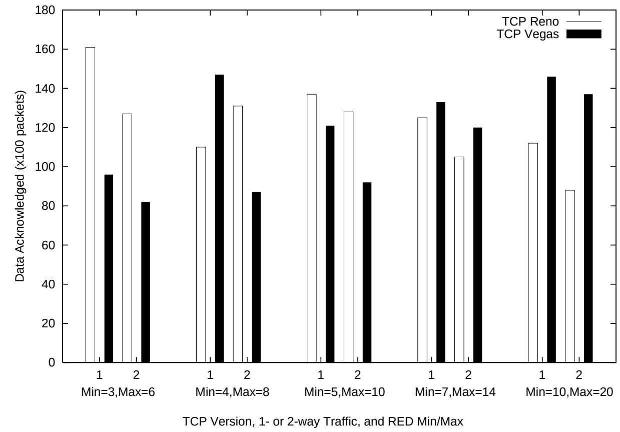

Figure 4. Network 1, variation of RED thresholds

Note how similar Figures 3 and 4 are. In fact, the only significant difference occurs in the second set of bars of each figure, i.e., $q = 7$ and $Min = 4, Max = 8$, respectively; for 1-way traffic, Vegas performs better than Reno does with RED, due simply to the randomness induced by RED.

The heuristic here was selected to produce results similar to the prior section. A more obvious formula,

$$\alpha = RED_{min} \text{ and } \beta = RED_{max},$$

allows Vegas to outperform Reno in all cases. Unfortunately, loss rates also increase and link utilization decreases, which is undesirable.

If we examine Figures 3 and 4 carefully, we see that if the goal is even competition between Reno and Vegas, our heuristic results in values too small for small queue sizes and too large for large queue sizes. In fact, these values are off by an amount proportional to the queue size! This leads us to the obvious conclusion that the heuristic we desire is not fixing $\alpha = RED_{min} + C_a$ and $\beta = RED_{max} + C_b$, with C_a, C_b constant as we have done here; instead we desire proportionality to queue size or effective queue size (based on the RED thresholds): $\alpha = C_a \times RED_{min}$ and $\beta = C_b \times RED_{max}$

4.3. Network 2

Now we consider our true interest — real-world networks such as the current computational grid (see Table 1) between Los Alamos and Sandia National Laboratories. Figure 4.3 presents the results for two-way traffic on this network. Full utilization is 258×10^4 packets. (See Table 1; $155Mbps \times \frac{1,000,000b}{1Mb} \times 200s \times \frac{1B}{8b} \times \frac{1pkt}{1500B}$.).

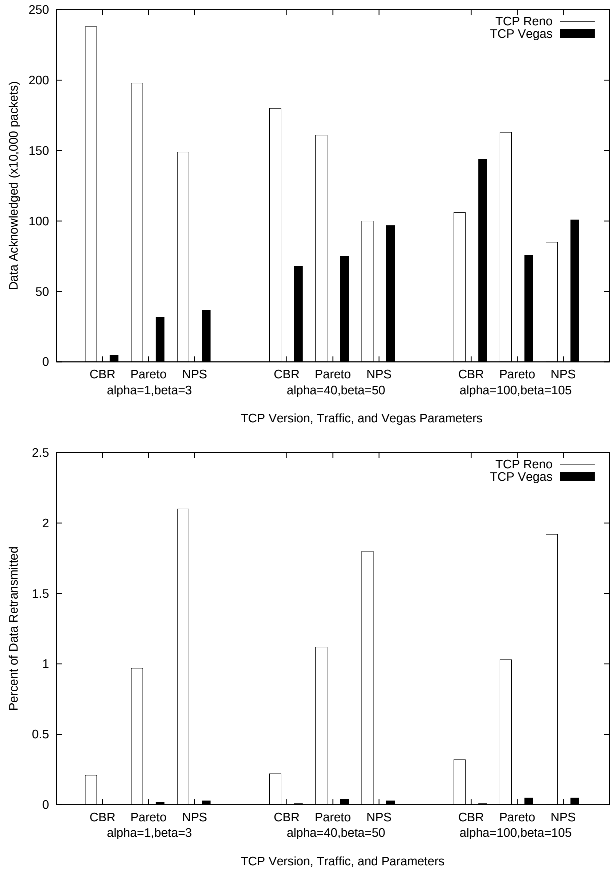

Figure 5. Results for network 2

Here we see that the default α and β of 1 and 3 are clearly inadequate; but by increasing them to 100 and 105, the performance under CBR and NPS traffic for Vegas is better than that of Reno. These values are arbitrary; any sufficiently large values will work.

For insight into the selection of α and β values, recall that α and β are tuned here primarily to keep Reno from acquiring an unfair share of bandwidth (see subsection 4.1). The actual number of packets a fair flow can have unacknowledged is just the $bandwidth \times delay$ product of the network divided by the number of flows. In this case, that is:

$$\frac{(155Mbps \times \frac{1,000,000b}{1Mb}) \times (7ms \times \frac{1s}{1000ms}) \times \frac{1B}{8b} \times \frac{1pkt}{1500B}}{50\ flows}$$

which is only about two packets. Thus, without TCP Reno, the default $\alpha = 1, \beta = 3$ would suffice; with TCP Reno competition, we must modify these values.

One heuristic for this network is to calculate $\frac{RED_{min}}{N}$ where N is the number of flows. This gives the effective fair queueing space per flow; which is about 134 packets here ($\frac{6711}{50}$). The values for α and β may be then be set slightly smaller than this value to achieve comparable performance to Reno. (The difference is due to the superior properties of the Vegas algorithm; using $\frac{RED_{min}}{N}$ often results in unfairly poor performance for Reno flows and is generally a bad idea).

However, Reno does outperform Vegas for Pareto traffic, regardless of α and β. Tests for values up to 1000 reveal that the problem is not due to an improper choice of these parameters, but is instead fundamental to current implementations of TCP Reno and Vegas and the heavy-tailed distribution. The problem is two-fold.

First, Reno is more aggressive during slow start, and in this case, benefits. Reno doubles its congestion window every RTT whereas Vegas doubles it every other RTT. Vegas does this in order to detect what effects its change has on the network before rampantly sending more data. Two solutions to this problem are immediately apparent: (1) If most flows use Vegas (and hence, the same slow-start algorithm), they would compete fairly with each other. (2) The use of ECN [23] or similar ideas to glean information from the network can allow Vegas to double its window at the same rate as Reno without adverse affects.

A second problem is that these parameters along with running Pareto traffic through TCP Reno creates unpredictable fluctuations in queue length; thus interfering with the RTT estimates used by Vegas. Vegas then uses these inaccurate estimates to control its sending rate, to its detriment. This problem is due to the bursty nature of aggregated TCP Reno flows [13] and would disappear if all flows were TCP Vegas.

Figure 4.3 also shows that the loss rates for TCP Reno are 20 to 70 times higher than those of TCP Vegas in this simulation. This is obviously bad for the network, and it can also drastically affect the performance of a grid application using TCP for data transport. When a loss occurs, it must be detected (via observation of the acknowledgements coming from the receiver) at the sender side, and the packet must be retransmitted. Thus, any lost packet arrives at the destination delayed at least one full RTT. If that packet contained critical data, the upper-level grid application may block the entire time. Therefore, loss patterns are often more important than total deliverable bandwidth. So, Vegas may provide better performance to a grid application, even providing lower total bandwidth, due to the lower loss rates.

4.4. Network 3

This network represents the future computational grid between Los Alamos and Sandia National Laboratories. The results are presented in Figure 6, again for bidirectional traffic.

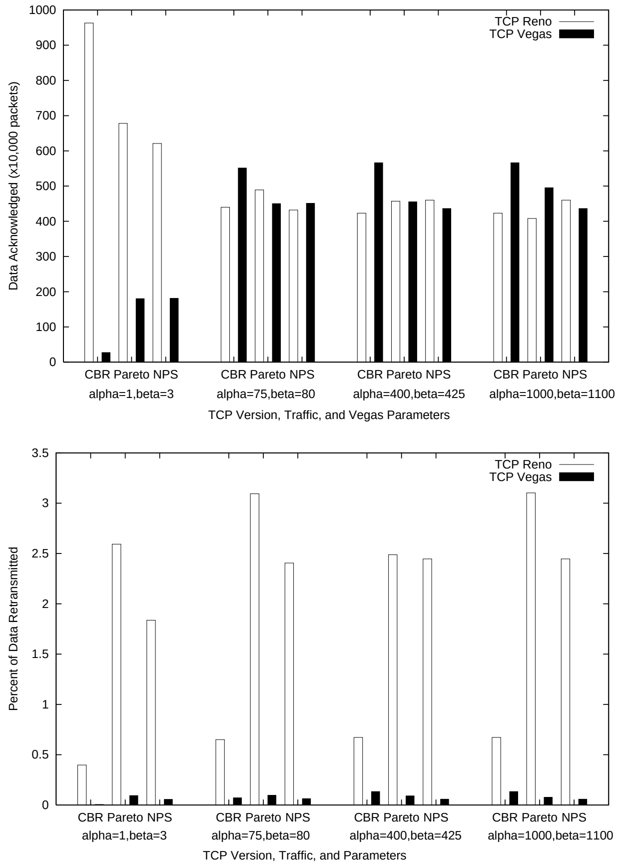

Figure 6. Results for network 3

The figure with "Data Acknowledged" on the dependent axis presents two new features. First, α and β must occasionally be set quite high for Vegas to outperform Reno. Second, simply increasing α and β is sometimes insufficient to increase performance; for NPS traffic, Vegas performs better with α, β set at 75 and 80 than it does with α, β over-aggressively set at 1000 and 1100.

These values concur with the the heuristic used in the prior section; setting α and β approximately equal to the effective queue size, $\frac{RED_{min}}{N}$ would indicate setting α, β to be slightly less than $\frac{8948}{100} \approx 89$ for fair competition with Reno. Note that this is exactly what we see, for $\alpha = 75, \beta = 80$. Unfortunately, due to the somewhat chaotic feedback behavior of TCP [13], combined with a random traffic generator, RED, and the high bandwidth of the network, these values may need to be significantly increased to make Vegas competitive. This is the case for TCP with a Pareto traffic generator, most likely due to the heavy-tailed nature of the distribution and the effects listed in the prior subsection.

Recall that we only seek to show that Vegas is competitive with Reno for *some* administratively set choice of parameters α and β. Once most TCP flows are TCP Vegas, choosing parameters will be less of an issue [6, 9]. This is also why we we present results at different values of these parameters for each network.

As in the previous section, even when Vegas and Reno achieve similar bandwidth, Vegas may give better overall performance due to the extreme loss rates that Reno suffers. These losses are due to Reno's over-aggressive nature, in particular its use of slow-start restart. Using slow start on a chaotically congested network (also caused by TCP Reno [13]) can lead to the loss of up to a full bandwidth-delay product's worth of packets. On this network, that is $7ms \times 1Gbps = 855KB$ worth of data. TCP Vegas avoids most of these losses with a less aggressive slow start.

4.5. Network 4

We now consider the same computational grid as in the prior section but with increased delay (in order to simulate a more widely distributed grid).

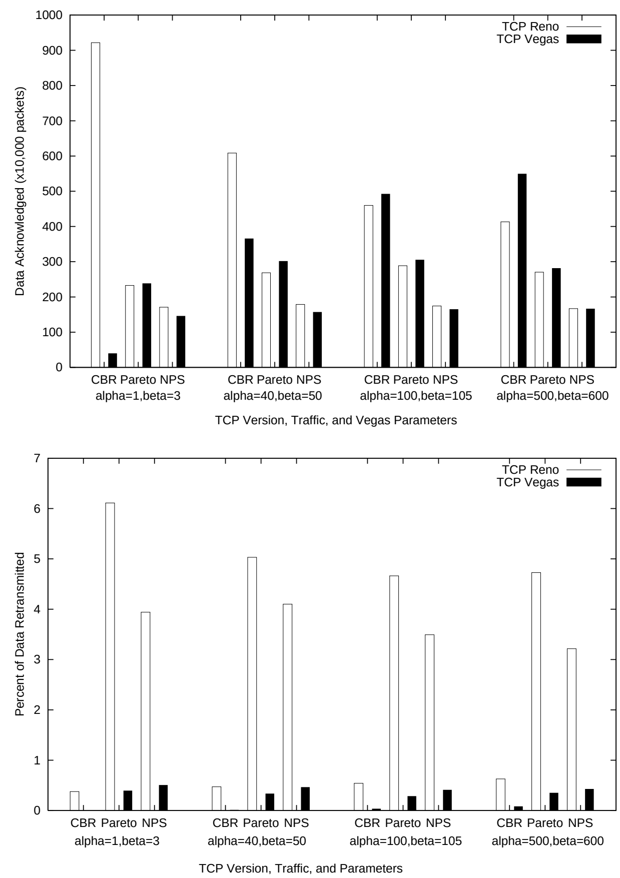

Figure 7. Results for network 4, high delay

All flows have problems with a large RTT. Although mostly incomparable due to the selection of different α and β parameters, the performance differences between figures

6 and 7 are still apparent. Careful study of these graphs reveals several things.

For stable CBR traffic, the RTT is unimportant as TCP does not have to respond to dramatic changes in network state. The acknowledgement values are only slightly lower than in the prior simulation, primarily due to the higher amount of time required for the congestion window to grow to appropriate values.

For Pareto and NPS traffic, we find poorer performance. As in the prior section, TCP Reno and Vegas perform similarly for values of α and β above 50. This is also due to the high delay in the network; the performance of TCP here is dominated by the flow-control behavior of acknowledgement-based self-clocking and similar features which are the same for all TCP algorithms. Performance differences between Pareto and NPS traffic are obviously due to the packet spacing. When an NPS flow begins transmission, it may flood the network and lose many packets. The distribution of packet arrival times in a Pareto flow makes this less likely.

Other than the increased delay, this network is the same as the prior network; meaning that the effective fair queue size and choice of α and β parameters via our simple heuristic remains the same. Indeed, the crossover point for Reno outperforming Vegas to Vegas outperforming Reno is for α and β near 89 packets.

We do not present results for that point, because only slightly larger values are required to make Vegas competitive with Reno. This is due to the increased delay in the network; here the $bandwidth \times delay$ product is:

$$(622Mbps \times \frac{1,000,000b}{1Mb}) \times (100ms \times \frac{1s}{1000ms})$$

which, converted to packets (via $\frac{1B}{8b} \times \frac{1pkt}{1500B}$) gives a fairly large value:

$$5183\, packets \approx 52 \frac{packets}{flow}$$

In all prior simulations, the value of $bandwidth \times delay/number\ of\ flows$ was below 4 packets, meaning that the only reason to increase the defaults of $\alpha = 1$ and $\beta = 3$ was to keep Reno from unfairly using bandwidth. With this network, Vegas must also try to keep more packets queued in routers to effectively cope with the increased delay in feedback information. Thus, for this network, α and β must first be increased to cope with the high delay, and then again to keep Reno from unfairly using bandwidth. This leads to the values given above.

The link utilization in network 4 is significantly worse than in network 3. Here the values are only 92% for CBR traffic, 55% for Pareto traffic, and 32% for NPR traffic. The latter numbers are unacceptably bad. Using all Vegas flows would help solve this problem. Loss rates approximately double due to the increased time it takes for a flow to respond to changes in the network state; with TCP Reno losing from 3-6% of its data precisely in those cases where latency most matters (Pareto/NPS distributions).

5. Related Work

We made a few simplifying assumptions, with one in particular that should be removed in future work: we have only tested traffic of a given type against traffic of that same type. Real networks have multiple types of traffic in competition, and we would like to test this — first by studying the relative proportion of various types on our network and then by using that data in our simulations.

Other extensions of this work include developing a more mathematical model of our systems to find provably optimal values for α and β, using uncontaminated traffic traces to drive simulations, or using implementations of Reno and Vegas to emulate these networks. We are also looking at ways to use ECN or other mechanisms to set α and β rather than basing these values on information (the number of flows in the network or the effective fair queue space) that is difficult to infer.

6. Conclusion

Prior research has used carefully crafted examples to study the performance of Reno or Vegas. In this paper, we have generalized those examples and made them more realistic. We are confident that Vegas can and will effectively compete with Reno in nearly all situations.

In particular, we showed how inappropriate the default values of $\alpha = 1$ and $\beta = 3$ are, explained their relationship to variations in network and traffic parameters, and then discussed how to use the effective queue size to set these parameters to improve performance. We have also shown how switching from Reno to Vegas will improve overall performance. Given that TCP is ubiquitous in today's networks, these results lead us to believe that many would benefit from more widespread adoption of TCP Vegas.

References

[1] W. E. Johnston, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid," *Proceedings of 8th IEEE International Symposium on High-Performance Distributed Computing*, August 1999.

[2] ANL, CalTech, LBL, SLAC, JF, U. Wisconsin, BNL, FNL, and SDSC, "The Particle Physics Data Grid." http://www.cacr.caltech.edu/ppdg/.

[3] W. Feng, I. Foster, S. Hammond, B. Hibbard, C. Kesselman, A. Shoshani, B. Tierney, and D. Williams, “Prototyping an Earth System Grid.” http://www.scd.ucar.edu/css/esg/.

[4] M. Allman and A. Falk, “On the Effective Evaluation of TCP,” *ACM Computer Communications Review*, vol. 29, October 1999.

[5] T. Bonald, “Comparison of TCP Reno and TCP Vegas: Efficiency and Fairness,” *Performance Evaluation*, vol. 36, no. 37, pp. 307–332, 1999.

[6] L. Brakmo and L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” *IEEE Journal on Selected Areas in Communication*, vol. 13, pp. 1465–1480, October 1995.

[7] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP,” *Computer Communication Review*, vol. 26, pp. 5–21, July 1996.

[8] T. Henderson, E. Sahouria, S. McCanne, and R. Katz, “On Improving the Fairness of TCP Congestion Avoidance,” *Proceedings of the IEEE Conference on (GlobeCom)*, 1998.

[9] U. Hengartner, J. Bolliger, and T. Gross, “TCP Vegas Revisited,” *Proceedings of IEEE Infocom 2000*, pp. 1546–1555, March 2000.

[10] M. Mathis and J. Mahdavi, “Forward Acknowledgement: Refining TCP Congestion Control,” *ACM Computer Communication Review*, vol. 26, October 1996.

[11] J. Mo, R. J. La, V. Anantharam, and J. Walrand, “Analysis and Comparison of TCP Reno and Vegas,” *Proceedings of INFOCOM ’99*, pp. 1556–1563, March 1999.

[12] C. Parsa and J. J. Garcia-Luna-Aceves, “Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media,” *Proceedings of IEEE 7th International Conference on Network Protocols*, November 1999.

[13] A. Veres and M. Boda, “The Chaotic Nature of TCP Congestion Control,” *Proceedings of IEEE Infocom 2000*, March 2000.

[14] W. Feng and P. Tinnakornsrisuphap, “The Failure of TCP in High-Performance Computational Grids,” *Proceedings of SC 2000*, November 2000.

[15] Y. R. Yang, M. S. Kim, and S. S. Lam, “Transient Behaviors of TCP-friendly Congestion Control Protocols.” ftp://ftp.cs.utexas.edu/pub/lam/transient_tech.ps.gz, July 2000.

[16] R. J. La, J. Walrand, and V. Anantharam, “Issues in TCP Vegas.” http://www.path.berkeley.edu/ hyongla/PAPERS/vegas_issue.ps.

[17] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[18] Various, “ns, the Network Simulator, version 2.1b7.” http://www.isi.edu/nsnam/ns/.

[19] L. Zhang, S. Shenker, and D. D. Clark, “Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic,” *Proceedings of ACM SigComm 1991*, September 1991.

[20] P. Danzig and S. Jamin, “tcplib: A Library of TCP Internetwork Traffic Characteristics.” http://irl.eecs.umich.edu/jamin/papers/tcplib/tcplibtr.ps.Z, 1991.

[21] V. Paxson and S. Floyd, “Wide-Area traffic: The Failure of Poisson Modeling,” *ACM SigComm 1994*, pp. 257–268, 1994. ftp://ftp.ee.lbl.gov/papers/poisson.ps.Z.

[22] J. Kim and D. Lilja, “Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs,” *Proceedings of the Workshop on Communication and Architectural Support for Network Based Parallel Computing (CANPC)*, pp. 202–216, February 1998.

[23] S. Floyd, “TCP and Explicit Congestion Notification,” *ACM Computer Communication Review*, vol. 24, pp. 10–23, October 1994.