Proceedings of the 2000 International Conference on Parallel Processing (ICPP 2000).

The Adverse Impact of the TCP Congestion-Control Mechanism in Heterogenous Computing Systems*

Wu-chun Feng[†§] and Peerapol Tinnakornsrisuphap[‡]
feng@lanl.gov, tinnakor@cae.wisc.edu

[†] Research & Development in Advanced Network Technology (RADIANT)
Computing, Information, and Communications Division
Los Alamos National Laboratory
Los Alamos, NM 87545

[‡] Department of Electrical & Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706

Abstract

Via experimental study, we illustrate how TCP modulates application traffic in such a way as to adversely affect network performance in a heterogeneous computing system. Even when aggregate application traffic smooths out as more applications' traffic are multiplexed, TCP induces burstiness into the aggregate traffic load, and thus hurts network performance. This burstiness is particularly bad in TCP Reno, and even worse when RED gateways are employed. Based on the results of this experimental study, we then develop a stochastic model for TCP Reno to demonstrate how the burstiness in TCP Reno can be modeled.

Keywords: *TCP, heterogeneous computing, high-performance networking, network traffic characterization.*

1 Introduction

The ability to characterize the behavior of the resulting aggregate network traffic can provide insight into how traffic should be scheduled to make efficient use of the network, and yet still deliver expected quality-of-service (QoS) to end users. These issues are of fundamental importance in widely-distributed, heterogeneous computational grids such as the *Earth System Grid* [2].

Recent studies in network traffic characterization have concluded that network traffic is self-similar in nature [9, 14]. That is, when traffic is aggregated over varying time scales, the aggregate traffic pattern remains bursty, regardless of the time granularity. Additional studies have concluded that the heavy-tailed distributions of file size, packet interarrival, and transfer duration fundamentally contribute to the self-similar nature of aggregate network traffic [16].

The problems with the above research are three-fold. First, the knowledge that self-similar traffic is bursty at coarse-grained time scales provides little insight into the network's ability to achieve an expected QoS through the Internet's use of traditional statistical-multiplexing techniques because the effectiveness of such techniques manifests itself at the granularity of milliseconds, not tens or hundreds of seconds [5]. Second, although current models of network traffic may apply to existing file-size distributions and traffic-arrival patterns, these models will not generalize as new applications and services are introduced to the Next-Generation Internet [15]. Third, and most importantly, the proofs of the relationship between heavy-tailed distributions and self-similar traffic in [16, 7] ignore the involvement of the TCP congestion-control mechanism.

A recent study [3] by Feng et al. addresses the above problems by focusing on TCP in a homogeneous parallel-computing environment (or cluster computer), where every computing node runs the same implementation of TCP. Feng et al. show that all flavors of TCP induce burstiness, and hence may contribute to the self-similar nature of aggregate traffic. However, the study does not examine the effects that different TCP implementations have on each other in a heterogeneous parallel-computing system.

Mo et al. [10] argue that in a heterogeneous networking environment that TCP Reno steals bandwidth from TCP Vegas while keeping its packet loss low as well; hence providing a disincentive for researchers in high-performance

*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 00-2554.

computing to switch from Reno to Vegas. In fact, while the Linux 2.1 kernel had TCP Vegas as its reliable communication protocol, the Linux 2.2 kernel has reverted back to TCP Reno based on studies such as [10]. Contrary to [10], we demonstrate that TCP Reno performs worse than TCP Vegas in a heterogeneous computing environment.

2 Background

TCP is a connection-oriented service that guarantees reliable, in-order delivery of data. Its flow-control mechanism ensures that a sender does not overrun the buffer at the receiver, and its congestion-control mechanism tries to prevent too much data from being injected into the network. While the size of the flow-control window is static, the size of the congestion window evolves over time, according to the status of the network.

2.1 TCP Congestion Control

Currently, the most widely-used TCP implementation is TCP Reno [6]. Its congestion control mechanism has two phases: slow start and congestion avoidance. In slow start, the congestion window grows exponentially until a timeout occurs, which implies that a packet has been lost. At this point, a $CongestionThreshold$ (CT) value is set to the halved window size; TCP Reno resets the congestion window size to one and re-enters the slow-start phase, increasing the congestion window exponentially up to CT. When CT is reached, TCP Reno enters its congestion-avoidance phase in which the congestion window is increased by "one packet" every time the sender successfully transmits a window's worth of packets across the network. When a packet is lost during congestion avoidance, TCP Reno takes the same actions as when a packet is lost during slow start.

To enhance performance, Reno also implements fast-retransmit and fast-recovery mechanisms for both the slow-start and congestion-avoidance phases. Rather than timing out while waiting for the acknowledgment (ACK) of a lost packet, if the sender receives three duplicate ACKs (indicating that some packet was lost but later packets were received), the sender immediately retransmits the lost packet (fast retransmit). Since later packets were received, the network congestion is assumed to be less severe than if all packets were lost, and the sender only halves its congestion window and re-enters the congestion-avoidance phase (fast recovery) without going through the slow start again.

TCP Vegas [1] introduces a new congestion-control mechanism that tries to prevent congestion rather than react to the it after it has occurred. The basic idea is as follows: When the congestion window increases in size, the expected sending rate (ER) increases as well. However, if the actual sending rate (AR) stays roughly the same, this implies that there is *not* enough bandwidth available to send at ER, and therefore, any increase in the size of the congestion window will result in packets filling up the buffer space at the bottleneck gateway. TCP Vegas attempts to detect this phenomenon and avoid congestion at the bottleneck gateway by adjusting the congestion-window size, and hence, reduce ER as necessary to adapt to the available bandwidth.

To adjust the window size appropriately, Vegas defines two threshold values, α and β, for the congestion-avoidance phase, and a third threshold value, γ, for the transition between the slow-start phase and congestion-avoidance phase. Conceptually, $\alpha = 1$ implies that Vegas tries to keep at least one packet from each stream queued in gateway while $\beta = 3$ keeps at most three packets from each stream.

If $RateDiff = ER - AR$, then when $RateDiff < \alpha$, Vegas increases the congestion window linearly during the next RTT. When $RateDiff > \beta$, Vegas decreases the window linearly during the next RTT. And when $\alpha \leq RateDiff \leq \beta$, the window remains unchanged. The γ parameter can be viewed as the "initial" β when Vegas enters its congestion-avoidance phase.

To enhance the performance of TCP, Floyd and Jacobson proposed the use of random early detection (RED) gateways [4] to detect incipient congestion. To accomplish this detection, RED gateways maintain an exponentially-weighted, moving average of the queue length. As long as the average queue length stays below the minimum threshold (min_{th}), all packets are queued, and thus, no packets are dropped. When the average queue length exceeds min_{th}, packets are dropped with some calculated probability P. And when the average queue length exceeds a maximum threshold (max_{th}), all arriving packets are dropped.

2.2 TCP Probability & Statistics

Rather than use the Hurst parameter from self-similar modeling as is done in many studies of network traffic [9, 12, 13, 14, 16], we use the *coefficient of variation* ($c.o.v.$) because it better reflects the predictability of incoming traffic at finer time granularities, and consequently, the effectiveness of statistical multiplexing over the Internet. The $c.o.v.$ is the ratio of the standard deviation to the mean of the observed number of packets arriving at a gateway in each round-trip propagation delay. The $c.o.v.$ gives a normalized value for the "spread" of a distribution and allows for the comparison of "spreads" over a varying number of communication streams. If the $c.o.v.$ is small, the amount of traffic coming into the gateway in each RTT will concentrate mostly around the mean, and therefore will yield better performance via statistical multiplexing.

By the Central Limit Theorem, the sum of independent variables results in a random variable with less burstiness, or equivalently, a smaller $c.o.v.$. However, even if traffic sources are independent, TCP introduces dependency between the sources, and the traffic does not smooth out as more sources are aggregated, i.e., $c.o.v.$ is large [3].

3 Experimental Study

The goal of this study is to understand the dynamics of how TCP modulates application-generated traffic in a heterogenous computing system. While this issue has been largely ignored in the self-similar literature [9, 14, 16, 12, 13], we strive to isolate and understand the TCP modulation so that we may be better able to schedule network resources. Understanding how TCP modulates traffic can have a profound impact on the $c.o.v$, and hence, throughput and packet loss percentage of network traffic. This, in turn, directly affects the performance of distributed computing systems such as the *Earth System Grid* [2].

3.1 Network Model

To characterize the TCP modulation of traffic, we first generate application traffic according to a known distribution. We then compare the $c.o.v.$ of this distribution to the $c.o.v.$ of the traffic transmitted by TCP. We can then determine whether TCP modulates the traffic, and if it does, how it affects the shape (burstiness) of the traffic, and hence, the performance of the network.

Consider a heterogenous cluster consisting of $M + 1$ compute nodes, where $1 \leq M \leq 100$. Since one of the worst-case network-congestion scenarios involves performing an intensive all-to-one communication, hence generating "hot spots" near the receiving node, we examine this case where the receiving node is a single server with M clients sending to it, as shown in Figure 1. Each client is linked to a common gateway with a full-duplex link with bandwidth μ_c and delay τ_c. A bottleneck full-duplex link of bandwidth μ_s and delay τ_s connects the gateway to the server. Each client generates Poisson traffic, i.e., single packets are submitted to the TCP stack with exponentially distributed interpacket arrival times with mean $1/\lambda$. All the clients attempt to send the generated packets to the server through the gateway and bottleneck link.

We use *ns* [11], an event-driven simulator, as our simulation environment. In our simulations, we vary the total traffic load offered by varying the number of clients M. We use UDP, TCP Reno (with delay acknowledgements both on and off), and TCP Vegas as the transport-layer protocols. We also test the effects of two queueing disciplines in the gateway, FIFO (First-In, First-Out) and RED, to see whether the queueing discipline has any effect on the burstiness generated by the TCP protocol stack. We calculate the $c.o.v.$ of the aggregate traffic generated by the clients, based on the known distribution each client uses to generate its traffic, and compare it to the measured $c.o.v.$ of the aggregate TCP modulated traffic as it arrives at the gateway. The parameters used in the simulation are shown in Table 1.

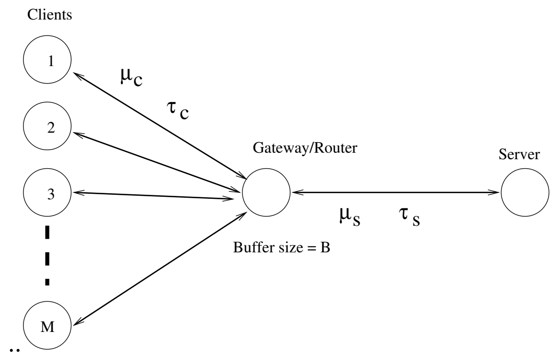

Figure 1. Network Model

Parameters	Value
client link bandwidth (μ_c)	10 Mbps
client link delay (τ_c)	25 ms
bottleneck link bandwidth (μ_s)	50 Mbps
bottleneck link delay (τ_s)	25 ms
TCP max advertised window	20 packets
gateway buffer size (B)	50 packets
packet size	1500 bytes
average client packet intergeneration time ($1/\lambda$)	0.01 s
total test time	200 s
TCP Vegas/α	1
TCP Vegas/β	3
TCP Vegas/γ	1
RED min_{th}	10 packets
RED max_{th}	40 packets

Table 1. Simulation Parameters for the Network Model.

3.2 TCP Reno vs. TCP Vegas

From the simulation parameters, the amount of traffic each client generates on average is 100 packets per second, or 1.2 Mbps. Since each client generates traffic independently of the others, the average amount of traffic generated by all clients will reach network capacity when we aggregate 41 clients together. Because the traffic generated by the application layers is Poisson, the $c.o.v.$ of the unmodulated aggregate traffic is $1/\sqrt{(\lambda\tau)n}$ where n is the number of clients aggregated and $\tau = RTT = 2(\tau_c + \tau_s)$. Therefore, the traffic generated from application layer becomes smoother as the number of sources increases.

The study performed in [3] showed that when all clients ran UDP, UDP did not noticeably modulate the traffic be-

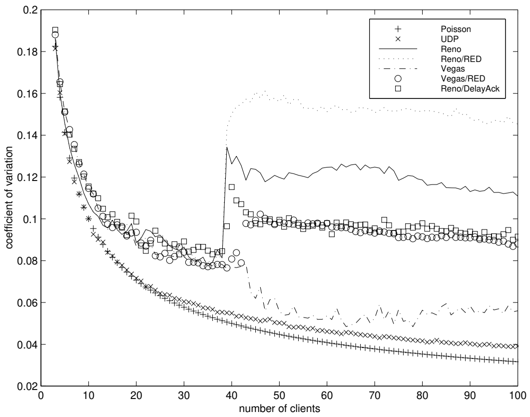

Figure 2. Coefficient of Variation of the Aggregated TCP Traffic.

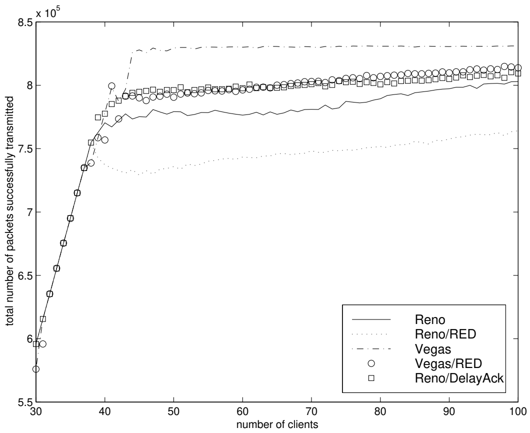

Figure 3. Throughput of the Aggregated TCP Traffic.

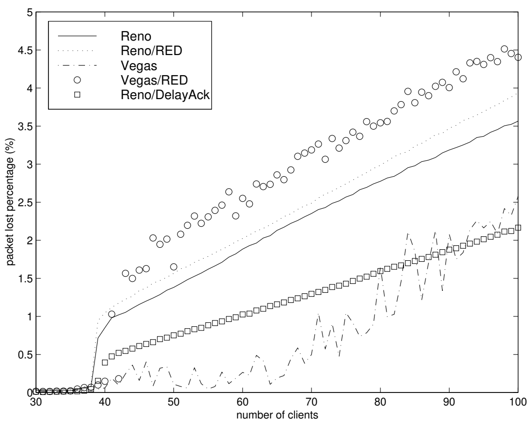

Figure 4. Packet Loss Percentage of the Aggregated TCP Traffic.

cause the $c.o.v.$ of aggregated UDP traffic is very close to that of the aggregated Poisson process. This is not surprising since UDP transmits packets received from the application layer to the network without any flow/congestion control. For TCP, we consider three cases: (1) *uncongested* — amount of traffic generated is much lower than the available bandwidth, i.e., # clients < 10, (2) *moderately congested* — amount of traffic generated causes some congestion, i.e., $10 \leq$ # clients ≤ 38, and (3) *heavily congested* — amount of traffic generated is higher than what the network can handle, i.e., # clients > 38.

In the uncongested case, traffic entering the gateway is similar to the traffic that the clients generate, irrespective of the TCP implementation, because congestion control has not kicked-in to control or modulate the traffic.

Under moderate congestion, the congestion-control mechanism begins to modulate the application-generated traffic which is transmitted into the network via TCP. We see this effect in Figure 2 — the TCP $c.o.v.$ numbers are up to 50% higher than the aggregated Poisson and UDP $c.o.v.$ numbers. This indicates that the congestion-control mechanism of TCP, regardless of implementation, noticeably modulates traffic when the network is moderately congested, i.e., TCP induces burstiness into the aggregate traffic stream. Fortunately, because the network only experiences intermittent congestion, this induced burstiness is not strong enough to adversely impact the throughput and packet loss [3], as shown in Figures 3 and 4.

Under heavy congestion, the $c.o.v.$ sharply increases for all TCP implementations except Vegas. The Reno and Reno/RED $c.o.v.$ numbers are over 140% and 200% larger than the aggregated Poisson and UDP $c.o.v.$ numbers, respectively. This result indicates that Reno and Reno/RED significantly modulate application-generated traffic (Poisson traffic) to be much more bursty. And unfortunately, this modulation is adverse enough to impact the throughput and packet loss percentage of Reno and Reno/RED [3], as shown in Figures 3 and 4. The above results imply that TCP Reno introduces a high level of dependency between the congestion-control mechanisms of each of the TCP streams.

While the above work illustrates the superiority of Vegas over Reno in a homogeneous parallel-computing environment, it is unlikely that everyone in a distributed parallel-computing environment will switch to Vegas all at once. In reality, the parallel-computing environment will be heterogeneous. So, we next examine how Vegas connections perform in the presence of Reno connections.

3.2.1 Half-Bandwidth Saturation with TCP Reno

In this set of experiments, we initially start-up 20 Reno clients to generate enough background traffic to saturate half of the available bandwidth. Then, additional clients are introduced to the initial 20 Reno clients. Figures 5 and 6 show the throughput and packet loss, respectively, for the clients which are added to the initial 20 Reno clients. The results show that Vegas still outperforms Reno. This coincides with the finding in [10] which states that Vegas connections are favored when the gateway buffer size is small (e.g., 50 packets).

Next, we test this setup with a larger gateway buffer size of 1500 packets with and without a RED gateway. We also use two different sets of RED parameters — RED1 (min_{th} = 300, max_{th} = 1200) and RED2 (min_{th} = 75, max_{th} = 300). Figures 7 and 8 show the throughput and packet loss for this follow-up experiment. While the throughput numbers are nearly identical, the relative differences in packet loss are more pronounced (though the absolute differences are smaller due to the larger buffer). Like [10], our results show that lower threshold values in RED favor Vegas. In contrast to [10], Vegas variants produce lower packet loss than their Reno counterparts, respectively.

While RED was originally introduced as a way to enhance TCP performance in Reno as well as Vegas, these results show that such gateways increase TCP modulation and can actually hurt TCP performance. The min_{th} and max_{th} parameters in RED make the buffer in the gateway appear smaller to TCP connections. TCP Reno, whose performance varies significantly with respect ot the gateway buffer size [8], suffers severely because its buffer requirements can very quickly become large as each stream treis to greedily increase its window size. On the other hand, TCP Vegas, even in the presence of 20 Reno connections, requires a minimal amount of buffer space per connnection and produces smoother traffic than Reno in the presence of a RED gateway, resulting in a better-performing TCP. Unfortunately, TCP implementations with RED perform worse than their "plain" counterparts with respect to $c.o.v.$, throughput, and packet loss (see Figures 2-8).

3.2.2 Full-Bandwidth Saturation with TCP Reno

To further "stack the deck" against TCP Vegas, we ran another set of experiments where we initially start-up 40 Reno clients to generate enough background traffic to saturate the network link. We then add new connections to the heavily-congested network and examine the performance of these new connections.

With a small gateway buffer of 50 packets, Figures 9 and 10 show the throughput and packet loss, respectively, for the clients which are added to the initial 40 Reno clients. Even in a heterogenous computing environment where Reno clients predominate, Vegas still performs admirably although not as well as in the previous set of experiments where only half the bandwidth is saturated with Reno. Thus, based on [3] and the results in this paper, Vegas

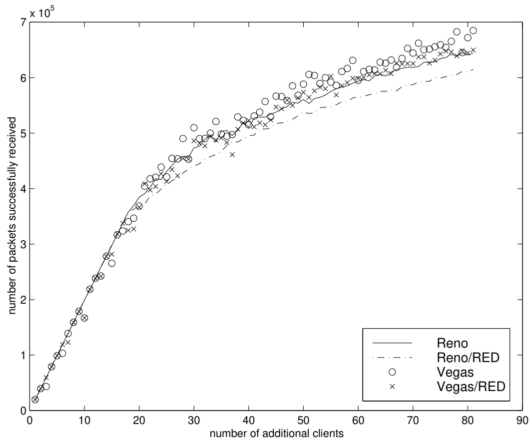

Figure 5. Throughput of Additional Clients (Background = 20 Reno clients, Buffer = 50 packets).

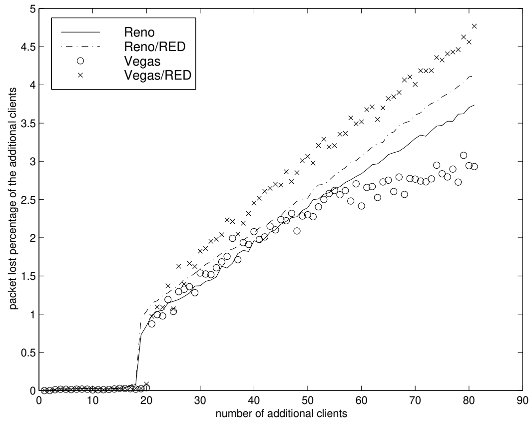

Figure 6. Packet-Loss Percentage of Additional Clients (Background = 20 Reno clients, Buffer = 50 packets).

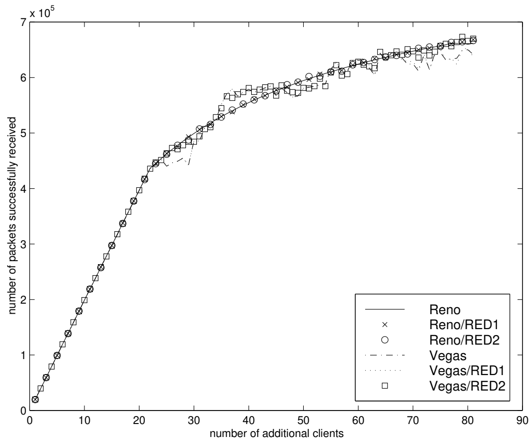

Figure 7. Throughput of Additional Clients (Background = 20 Reno clients, Buffer = 1500 packets).

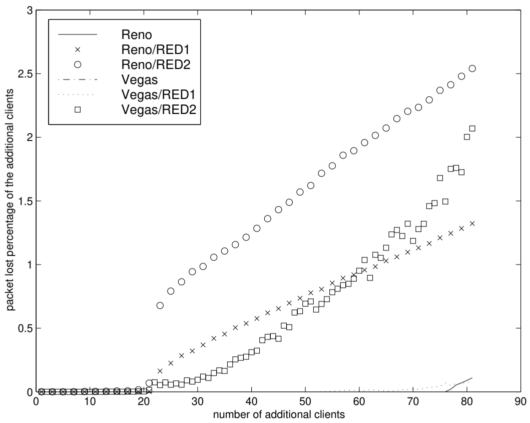

Figure 8. Packet-Loss Percentage of Additional Clients (Background = 20 Reno clients, Buffer = 1500 packets).

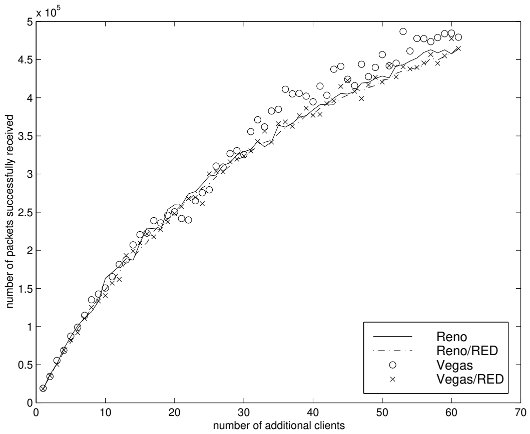

Figure 9. Throughput of Additional Clients (Background = 40 Reno clients, Buffer = 50 packets).

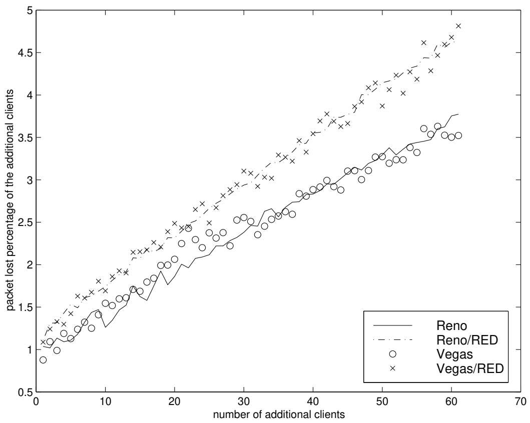

Figure 10. Packet-Loss Percentage of Additional Clients (Background = 40 Reno clients, Buffer = 50 packets).

performs best in a parallel-computing environment where all the compute nodes are running Vegas; as the number of Reno nodes increases, the advantage of Vegas over Reno diminishes.

Running the same tests with a larger gateway buffer of 1500 packets and using two different sets of RED parameters — RED1 where the min_{th} is 300 packets and the max_{th} is 1200 packets, and RED2 where the min_{th} is 75 packets and max_{th} is 300 packets — we find that TCP Vegas performs better than TCP Reno, contrary to what was found in [10]. As in the half-bandwidth saturation tests, the throughput numbers in all cases are relatively close to one another (Figure 11), however the relative differences in packet-loss percentage are more pronounced (Figure 12). And once again, the TCP Vegas variants produce lower packet-loss percentages than their TCP Reno counterparts, respectively.

3.2.3 Discussion

TCP Reno greedily tries to get as much bandwidth as possible by increasing its window size linearly until a packet loss occurs. These packet losses generally occur due to buffer overflows. Consequently, TCP Reno's bandwidth "estimation" mechanism results in the periodic oscillation of the size of the congestion window size as well as the queue occupancy of the gateway buffer. So, while TCP Vegas tries to keep a smaller queue size via its α and β parameters, TCP Reno keeps many more packets in the buffer on average. Mo et al. [10] then use this argument to explain that this is how TCP Reno "steals" bandwidth away from TCP Vegas.

There are two primary reasons why the results in [10] differ so much from ours. First, the experimental set-up used in their study focuses on pitting TCP Reno versus TCP Vegas one-on-one rather than dealing with a heterogeneous parallel-computing environment where some aggregate number of users are using Reno and others are using Vegas. Second, their study indicates that the input traffic distribution is drawn from an infinite-sized file, i.e., each connections always has data to transmit. Clearly, this kind of assumption on the traffic distribution benefits TCP Reno's "greedy" algorithm. Although a Poisson-distributed interpacket arrival pattern like ours may not accurately model real application-generated traffic either, we believe that it can more closely resemble network traffic than an infinite-sized file which is continually pushing data into the network.

In any case, the goal of this paper is not to model application-generated traffic patterns per se but to select a known distribution (i.e., Poisson) and isolate the effect of the TCP protocol stack on application-layer traffic via the $c.o.v.$ metric. In this context, we then investigate the performance of TCP Vegas in the presence of TCP Reno connections. While the study in [10] provides a disincentive for TCP Reno users to switch to TCP Vegas, our study conversely provides an incentive to switch from Reno to Vegas. Before we make any further claims about the performance of TCP Vegas in the presence of TCP Reno connections, we must thoroughly study how applications generate traffic before the traffic enters the TCP stack and then use these traffic patterns as our input-traffic distributions into TCP Reno and TCP Vegas.

4 Stochastic Model of TCP Reno

In this section, we extend our work into the stochastic modeling of the TCP Reno traffic to predict the burstiness of traffic entering the bottleneck gateway. With some reasonable simplifications, we show that the behavior of the c.o.v. of aggregated TCP Reno traffic can be reasonably predicted by stochastic modeling.

Using the notation from Table 1, consider the discrete-time series process whose timestep has length equal to the round-trip propagation delay of our client-server connection $\tau = 2(\tau_c + \tau_s)$. We neglect the time a packet spends waiting in the gateway buffer and all other processing time. Let $X_n^{(i)}$ be the number of packets generated by the application layer of client i during the nth interval. Then $X_n^{(i)}$ is a Poisson-distributed random variable with mean $\lambda\tau$. Let $W_n^{(i)}$ be the size of the TCP congestion window of client i at time n. In each interval, the number of packets transmitted cannot exceed the congestion window; the excess traffic is carried over to the next interval. Let $Y_n^{(i)}$ be the number of packets actually transmitted by client i in the interval n and $V_n^{(i)}$ be the excess packets carried over from interval $n-1$. We have the following relations:

$$Y_n^{(i)} = \min[X_n^{(i)} + V_n^{(i)}, W_n^{(i)}] \quad (1)$$

$$V_{n+1}^{(i)} = (X_n^{(i)} + V_n^{(i)}) - Y_n^{(i)} \quad (2)$$

In order to obtain the c.o.v. of the traffic in the steady state, we assume that all the TCP connections are always in the congestion avoidance phase and assume congestion will occur if and only if the total number of packets transmitted in an interval is more than the available bandwidth in that round $(\mu_s/(8 \times packet\ size) + B)\tau$. We also assume synchronization of all TCP connections, that is, once congestion has occurred, all the TCP congestion windows are cut in half by the fast-retransmit mechanism introduced in TCP Reno. If congestion does not occur, the congestion window will grow linearly according to the number of packets successfully transmitted in that interval. Let γ be the available bandwidth, then for $1 \le i \le M$,

$$W_{n+1}^{(i)} = W_n^{(i)} + \frac{Y_n^{(i)}}{W_n^{(i)}} \quad ; \quad \sum_{i=1}^{M} Y_n^{(i)} \le \gamma \quad (3)$$

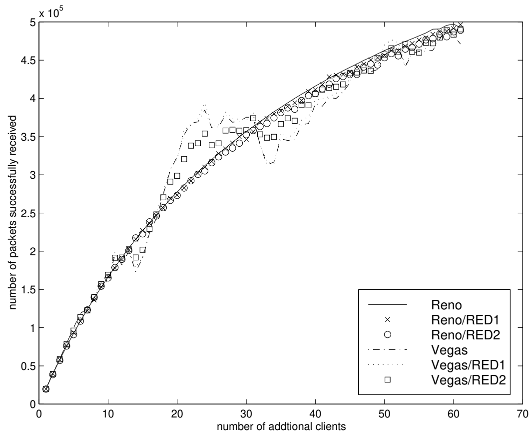

Figure 11. Throughput of Additional Clients (Background = 40 Reno clients, Buffer = 1500 packets).

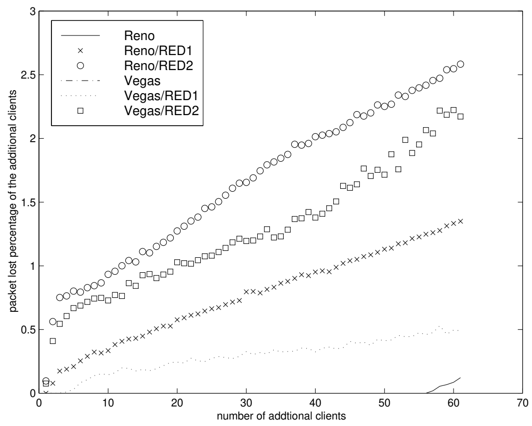

Figure 12. Packet-Loss Percentage of Additional Clients (Background = 40 Reno clients, Buffer = 1500 packets).

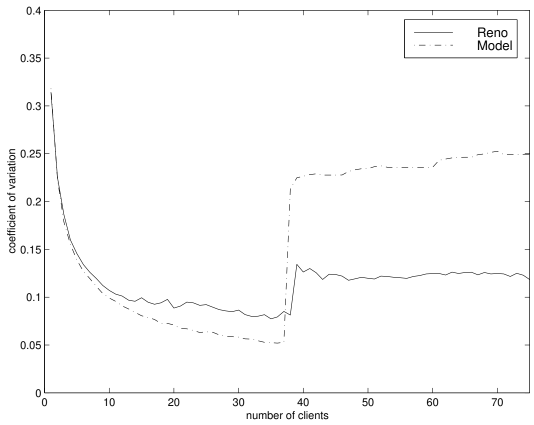

Figure 13. Comparison of the Coefficient of Variation between Simulated TCP Reno and Our Stochastic Model.

$$W_{n+1}^{(i)} = \frac{W_n^{(i)}}{2} \quad ; \quad \sum_{i=1}^{M} Y_n^{(i)} > \gamma \qquad (4)$$

Consider the traffic transmitted from each client to the gateway $Y_n = \sum_{i=1}^{M} Y_n^{(i)}$. We can obtain the observed c.o.v. numerically. The result in Figure 13 shows that the trend of the c.o.v. in this model does closely follow that of TCP Reno. The dependencies between the traffic streams imposed by equation 4 should exist to some degree in the actual aggregated TCP streams. The model does overestimate the burstiness of aggregated TCP streams in the heavily loaded case because it assumes very strong dependencies between TCP streams and does not include the TCP Timeout.

5 Conclusion

From our experiments, we have shown that the congestion-control mechanisms of TCP Reno and TCP Vegas modulate the traffic generated by the application layer. The congestion-control mechanism of TCP Reno adversely modulates the traffic to be more bursty, which subsequently affects the performance of statistical multiplexing in the gateway; this modulation occurs for two primary reasons: (1) the periodic oscillation of the congestion window size and buffer occupancy at the gateway caused by the continual "additive increase / multiplicate decrease (or re-start slow start)" probing of the network state and (2) the dependency between the congestion-control decisions made by multiple TCP streams which increases as the number of streams increase, i.e., TCP streams tend to recognize congestion in the network at the same time and thus halve their congestion windows at the same time [3]. As a result, TCP Reno traffic does not significantly smooth out even when a large number of streams are aggregated. On the other hand, TCP Vegas, during congestion avoidance, does not modulate the traffic to be as bursty as TCP Reno. This translates to smoother aggregate network traffic, and hence better overall network performance

In contrast to the work done in [10], we illustrated via *ns* simulations that TCP Vegas performs as well as, if not better than, TCP Reno in the presence of existing TCP Reno connections. Thus, while the work of [10] discourages researchers in high-performance computing to switch from TCP Reno to TCP Vegas; our work reaches the opposite conclusion, hence providing an incentive to switch from Reno to Vegas.

In the future, we intend to refine our stochastic model to better fit TCP Reno in the heavily congested case and to develop a model for TCP Vegas. A major part of this work will look into the modeling of TCP traffic to include the interactions between multiple TCP streams. We believe that this is a significant problem which needs to be solved before we can predict the behavior of network traffic and achieve the necessary QoS guarantees over the Internet or any distributed heterogeneous-computing system.

References

[1] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE JSAC*, 13(8):1465–1480, October 1995.

[2] 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*, July 1999.

[3] W. Feng, P. Tinnakornsrisuphap, and I. Philp. On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System. In *20th International Conference on Distributed Computing Systems*, April 2000.

[4] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[5] M. Grossglauser and J. Bolot. On the Relevance of Long-Range Dependence in Network Traffic. *Computer Communication Review*, 26(4):15–24, October 1996.

[6] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM'88*, pages 314–332, August 1998.

[7] T. G. Kurtz. Limit Theorems for Workload Input Models. *Stochastic Networks: Theory and Applications*, 1996.

[8] T. V. Lakshman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, June 1997.

[9] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transaction on Networking*, 2(1):1–15, February 1994.

[10] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *INFOCOM'99*, March 1999.

[11] ns. UCB/LBNL/VINT Network Simulator. *http://www-mash.cs.berkeley.edu/ns*.

[12] K. Park, G. Kim, and M. Crovella. On the Relationship Between File Sizes, Transport Protocols, and Self-Similar Network Traffic. In *4th International Conference on Network Protocols*, October 1996.

[13] K. Park, G. Kim, and M. Crovella. On the Effect of Traffic Self-Similarty on Network Performance. In *SPIE International Conference on Performance and Control of Network Systems*, 1997.

[14] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[15] W. Willinger and V. Paxson. Where Mathematics Meets the Internet. *Notices of the American Mathematical Society*, 45(8):961–970, September 1998.

[16] W. Willinger, V. Paxson, and M. Taqqu. Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, pages 27–53, 1998.