

# A Unified Framework for Modeling TCP-Vegas, TCP-SACK, and TCP-Reno \*

Adam Wierman  
Computer Science Department  
Carnegie Mellon University  
acw@cs.cmu.edu

Takayuki Osogami  
Computer Science Department  
Carnegie Mellon University  
osogami@cs.cmu.edu

Jörgen Olsén  
Department of Mathematics  
Uppsala University  
jorgen@math.uu.se

## Abstract

*We present a general analytical framework for the modeling and analysis of TCP variations. The framework allows the modeling of multiple variations of TCP, including TCP-Vegas, TCP-SACK, and TCP-Reno, under general network situations. In particular, the framework allows us to propose the first analytical model of TCP-Vegas for arbitrary on-off traffic that is able to predict the operating point of the network. The analysis provided by our framework leads to many interesting observations with respect to both the behavior of bottleneck links that are shared by TCP sources and the effectiveness of the design decisions in TCP-SACK and TCP-Vegas.*

## 1. Introduction

There has been a significant amount of research toward modeling variants of the Transmission Control Protocol (TCP) in order to understand the impact of this protocol on file transmission times and network utilization. Analytical models have emerged as a way to reduce the time required for evaluation compared to more traditional evaluations performed using event driven simulators such as *ns-2*. When designed carefully, analytical models also help researchers make design decisions about novel TCP mechanisms.

A wide variety of techniques have been applied to the problem of TCP modeling with a fair amount of success (see [17] for an extensive overview). These techniques range over renewal theory [18], Markov chains [11], and fluid models [1]. However, until the recent seminal work using fixed-point methods [4, 5, 6, 8, 9, 13, 14], no modeling technique was able to mimic both the structure of a TCP source and the interaction a source has with the network. The fixed-point methods take the novel approach of sepa-

rating the modeling of network behavior from the modeling of behavior within a TCP source, and then allowing the two to tune each other via feedback. The fixed-point framework allows general network topologies to be analyzed, and issues such as the interpretation of end-to-end loss rates in multiple bottleneck networks are addressed in [4, 9, 14].

In this paper, we generalize the fixed-point method introduced by Casetti and Meo in [5, 6] in order to model TCP-SACK and TCP-Vegas connections. The framework of Casetti and Meo, which uses a Markov chain to model the TCP source, has a few advantages over other fixed-point methods: (i) it can model explicit details of TCP, making it possible to distinguish different flavors of TCP; (ii) it allows modeling on-off traffic sources; (iii) it gives the fraction of time that TCP spends in each state, from which we can evaluate the effectiveness of each mechanism of the protocol.

A majority of existing analytical models focus on TCP-Reno, the most widely deployed variant of TCP, and there has been little research on analytical models of TCP-Vegas, a more recently proposed variant. Analytical models of TCP-Vegas have been difficult to develop because TCP-Vegas uses observed delay to detect an incipient stage of congestion and try to adjust the sending rate before packets are lost. Prior studies on measurement and simulation of the performance of TCP-Vegas suggest that in many situations it is able to provide users higher throughput and lower loss rates than TCP-Reno. Hence, it is an important task to model the performance of TCP-Vegas in order to understand how this protocol performs in a network shared with other variants of TCP.

Because TCP-Vegas uses observed delay, as well as loss events, to adjust the congestion window size, while TCP-Reno uses only loss events, the extension made in this work to the framework of Casetti and Meo for modeling TCP-Vegas is nontrivial. We make two major changes. First, analyzing the network model to determine the loss rate and the mean queueing delay is no longer enough; the full distribution of queueing delay must be obtained. Second, the Markov chain used to model a TCP-Reno source must be extended to use information from the delay distribution.

\*This work was supported by NSF Career Grant CCR-0133077, NSF ITR Grant 99-167 ANI-0081396, Cisco Systems, Spinnaker Networks via Pittsburgh Digital Greenhouse Grant 01-1, and the Swedish Foundation for Strategic Research.

The model that results from these extensions overcomes two major limitations of other TCP models. First, our model is the first model of TCP-Vegas that accurately predicts the operating point of the network; i.e., it predicts both the throughput and the loss rate of TCP sources. In particular, all but one previous work assumes loss-free operation [1, 2, 10, 12, 15]. Samios and Vernon [20] proposed the first analytical model to incorporate loss rate. They give a closed-form formula for the throughput achieved by a single TCP-Vegas sender as a function of the loss rate and the round trip time (RTT) of the network. However, they are not able to predict the full operating point of the network.

Second, our model is the first model of TCP-Vegas for arbitrary on-off traffic; all previous work on modeling TCP-Vegas is based on the analysis of TCP-Vegas for bulk transfers [1, 2, 10, 12, 15, 20]. Bulk transfer models are based on the assumption that a large amount of data is sent along a single connection. On-off traffic provides a more general model, including bulk transfer as a special case (when the on periods are long). On-off traffic applies both to FTP connections, the case of bulk transfer, and HTTP traffic, which inherently is made up of many short connections.

### 1.1. Summary of results

In this paper, we propose novel analytical models of TCP-Vegas and TCP-SACK sending on-off traffic. The framework of our analysis can be applied in very general settings, and the analysis in this paper leads to interesting observations about the behavior of bottleneck links and about the effectiveness of each mechanism within TCP. We now summarize the contributions of this paper.

1. We illustrate a general framework for modeling variations of TCP. We extend the framework of Casetti and Meo [5, 6] and model performance metrics such as the throughput, loss rate, and queueing delay, of various TCP flavors including TCP-Reno, TCP-SACK, and TCP-Vegas.
2. We propose an analytical model for TCP-Vegas for on-off traffic that predicts the operating point of the network.
3. We extend the analytical model for TCP-Reno proposed in [6] to include the exponential backoff mechanism and the minimum timeout duration. In addition, we extend the framework to model the selective acknowledgments used in TCP-SACK.
4. We extensively test the applicability of standard queueing models for approximating the performance of a bottleneck link and find, surprisingly, that an  $M/M/1/B$  queue is a good approximation.
5. We investigate the effectiveness of the new mechanisms introduced in TCP-SACK and TCP-Vegas and conclude that the selective acknowledgment mechanism of TCP-SACK is effective at avoiding timeouts and that the modified slow-start mechanism introduced in TCP-Vegas does

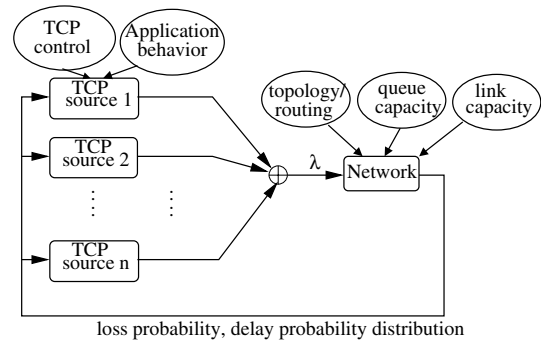


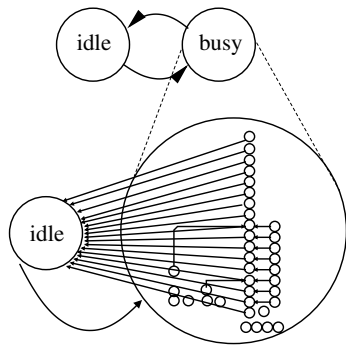
Figure 1. TCP analysis methodology

not help reduce packet loss but does result in spending more time in slow-start.

## 2. Methodology

In this section, we describe our framework for modeling the performance of TCP flavors. We extend the framework introduced by Casetti and Meo [5, 6] so that the performance of TCP-Vegas can be modeled. The framework consists of two components: the TCP sources and the network (see Figure 1). The structure mimics the interaction among senders within the network. The throughput of each source is independently analyzed via a Markov chain, while the loss rate and the probability distribution of the delay of the network is analyzed separately using queueing models. In [5, 6], only the loss rate is analyzed, but we require the delay distribution to model TCP-Vegas. Aspects such as network topology, queueing capacity, link capacity, packet arrival pattern, queue management scheme, and the network environment are taken into account within the queueing model.

Although the two components are analyzed separately, the parameters in the source and network models are tuned by iterative analysis with feedback from each other. That is, the source models are analyzed to obtain an aggregate traffic, which is used as the arrival process to the network model. The network model is analyzed to obtain the loss rate and delay probability distribution, which are used as the parameters in source models. Notice that a TCP source adjusts its sending rate (its congestion window size) based on the observation of events such as packet loss and delay of acknowledgments (ACKs). The source models are analyzed again, and the procedure is continued until a fixed-point solution is obtained. Given accurate source and network models, the fixed point of this iteration matches the operating point of the true network. In our model, the fixed-point solution is usually found within 10-15 iterations. In the rest of this section, we describe the source and network models in



**Figure 2. Model of a TCP source**

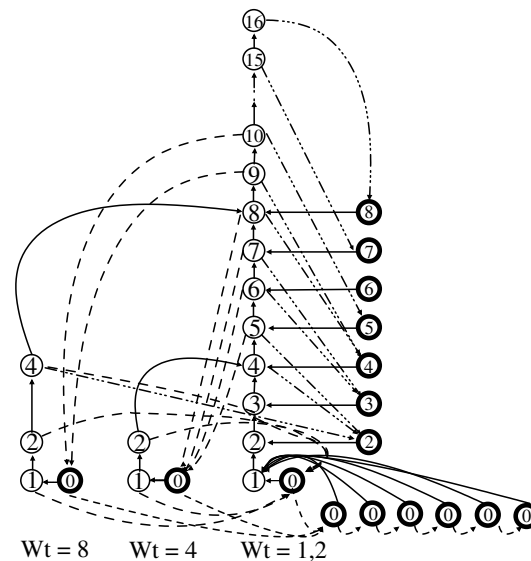
more detail.

Mimicking the structure of real-world network hosts, our model of an individual source is composed of two levels: an application level and a transport level. The application level alternates between the busy state and the idle state. While the application level is in busy state, the transport level moves among TCP level states, changing the window size and the phase of transmission (such as slow-start phase, congestion avoidance phase, fast-retransmit phase, and timeout phase). A rough picture of the interaction between the application and transport level models of a source is shown in Figure 2.

At the application level a TCP source alternates between two states: busy (on) and idle (off). During a busy period, a TCP source sends data over a TCP connection with the maximum rate allowed by the current window size (specified by the transport level). During an idle period, an application does not send any data. This application model is designed to mimic the FTP model built into *ns-2*. By varying the length of the busy periods, this serves as a general model that can represent both the short connections inherent in web traffic and the long connections inherent in bulk transfer. Notice that the application level is independent of the TCP flavor being modeled.

In the transport level model, the TCP source changes the window size and the phase of transmission, based on the observation of loss events and the delay seen by ACKs during the previous stage. How a congestion window size is adjusted depends on the flavor of TCP and is described in Section 3. Via a Markov chain analysis, we obtain the limiting probability of the fraction of time that the TCP source spends at each state. We then obtain the expected throughput of each sender.

The network is analyzed separately via queueing models. Each bottleneck link in the network is modeled as a queue with a finite buffer. The traffic from TCP sources that send packets through a bottleneck link is aggregated and used as an arrival process for the bottleneck link. The aggreg-



**Figure 3. Model of a TCP-Reno window model for  $W_M = 16$ .**

gated throughput  $\lambda$  is computed as the sum of the throughput from all the TCP sources that send packets through the bottleneck link. That is,  $\lambda = \sum_{i=1}^n \lambda_i$ , where  $n$  is the number of senders that send packets through the bottleneck link and  $\lambda_i$  is the throughput of the  $i$ -th sender. The arrival process for the bottleneck link, being the aggregation of  $n$  independent sources, is approximated as a Poisson process, and the aggregated throughput is used as the rate of the Poisson process into the bottleneck link. The loss rate and the delay distribution at each bottleneck link is then calculated using analysis of the queueing model. The loss rate and the probability distribution of the delay for each sender is then given by aggregating over all the bottleneck links which the sender sends packets through. The details of the specific network models used in this work follow in Section 4.

### 3. Source models

In this section, we describe in detail the transport level models of the source under three flavors of TCP: TCP-Reno, TCP-SACK, and TCP-Vegas. We extend the model of TCP-Reno introduced in [6] to improve the accuracy under high loss environments. In particular, we introduce the exponential backoff mechanism and the minimum timeout duration. The model of TCP-SACK described in Section 3.2 is a straightforward extension of our model for TCP-Reno. The model of TCP-Vegas described in detail in Section 3.3 shows a nontrivial extension of the model for TCP-Reno.

Before describing the details of the model for each flavor,

we present the high level idea of the transport level models. The behavior of the transport level is described by a continuous time Markov chain. The states keep track of the congestion window size, as well as information such as the slow-start threshold and whether we need to recover from loss. The rate of transitions is determined by the loss rate and the RTT (the mean and the standard deviation), provided by the network model. Namely, starting at a particular state, after one RTT, we transition to a different state; the probability of transitioning to each state depends on the loss rate and delay distribution. For some states such as timeout states and exponential backoff states, the duration of staying at the state is not one RTT, and the details are given later in this section. By solving this Markov chain, we can determine the limiting probabilities of the congestion window size, which determines the throughput of the sender.

### 3.1. Modeling TCP-Reno

We first describe the model of TCP-Reno. At the beginning of a busy state, the source starts in slow-start mode and sends one packet. Each returning ACK triggers the injection of two new packets into the network. Hence, the TCP window size is doubled once every RTT. This exponential window growth continues until the window size,  $w$ , reaches the current slow-start threshold; then TCP-Reno switches to congestion avoidance mode. In congestion avoidance mode, each returning ACK increases the window size as  $w \leftarrow w + 1/w$ , which results in the increase of the window size by one packet per RTT. This increase in window size continues until the maximum window size  $W_M$  is reached, or until a packet loss is observed. If the congestion window size is sufficiently large and not too many packets are lost, TCP-Reno will perform a fast retransmit/fast recovery operation: it quickly resends the lost packets, halves the congestion window size, and continues with congestion avoidance. If too many packets were lost, TCP-Reno will timeout and wait for  $T$  seconds before transmission will start over with a window size of one packet and a slow-start threshold set to  $\lfloor w/2 \rfloor$ , where  $w$  is the size of the congestion window when the lost packet was sent. If the first packet after a timeout is lost, TCP will double the length of the next timeout period. This doubling, usually called exponential backoff, continues up to a maximum timeout length of  $64T$ . Upon exiting the exponential backoff states, the window size is set to one. The timeout length is then also reset to  $T$ . In this paper, we ignore the delayed acknowledgment mechanism.

We now describe the TCP-Reno source model introduced in [6] with a few extensions that we propose. Figure 3 shows the continuous time Markov chain for the TCP-Reno source model when  $W_M = 16$ . A state represents an ordered triple  $(w, W_t, l)$  where  $w$  is the current window size,  $W_t$  is the current slow-start threshold, and  $l$  is either 1

or 0 depending on whether this state corresponds to a state where we need to recover from any loss (1) or not (0). States with thin borders correspond to states where no loss event has occurred ( $l = 0$ ). The two short chains made up of states with thin borders correspond to slow-start states (for slow-start thresholds of 8 and 4 respectively), and the long chain of states with thin borders corresponds to the congestion avoidance states. States with thick borders correspond to states where loss has occurred, but has not yet been recovered from ( $l = 1$ ). The vertically aligned chain of states with thick borders correspond to the fast retransmit states; the horizontally aligned chain of states with thick borders at the right bottom corner correspond to the exponential backoff states; the rest of the states with thick borders correspond to timeout states. Note that although TCP never uses a window size of zero, it is convenient to use this value to represent timeout states and exponential backoff states since no packets are sent during these periods.

We have made a few extensions to the model for TCP-Reno described in [6]. The first extension is the addition of the exponential backoff mechanism in TCP. The second extension is the addition of the minimum timeout  $T_{RTO}$  parameter to the calculation of the timeout length. Both of these extensions have been motivated by simulations that incur very high loss rates, and the improvements in the model that result from these extensions are shown in Figure 6.

We now define the transition rates of the TCP-Reno Markov chain. The transition rates are determined by the loss rate,  $P$ , and the mean RTT,  $\theta$ , based on the TCP protocol specification. For states where transitions can take place to many other states, the transition rates are derived by weighting the probability of the different events against the time needed for respective state transitions. For the transitions at the application level, the transition rate from the idle state to the state  $(1, \lfloor W_M/2 \rfloor, 0)$  is  $1/E[T_{\text{off}}]$ , where  $E[T_{\text{off}}]$  is the mean duration of the off (idle) period, and for each active busy state (i.e. slow-start states and congestion avoidance states), the transition rate back to the idle state is  $1/E[T_{\text{on}}]$ , where  $E[T_{\text{on}}]$  is the mean duration of the on (busy) period. For all active busy states  $(w, W_t, l)$  with  $w < W_t$ , the transition rate from  $(w, W_t, l)$  to  $(w+1, W_t, l)$  is  $P_w(0)\theta^{-1}$ , where we define  $P_w(i)$  to be the probability of losing  $i$  packets out of a window of  $w$  packets. Continuing with the loss free states  $(w, W_t, 0)$  in slow-start and congestion avoidance to the loss states, we transition from a window of size  $w$  to fast retransmit/fast recovery  $(\lfloor w/2 \rfloor, 2, 1)$  with rate  $P_{\text{fr}}^R \theta^{-1}$ , and to timeout  $(0, 1, 1)$  with rate  $P_{\text{to}}^R \theta^{-1}$ , where  $P_{\text{fr}}^R$  is the probability of fast retransmit/fast recovery and  $P_{\text{to}}^R$  is the probability of timeout. Finally, from the loss states  $(w, W_t, 1)$  we have the transitions from fast retransmit/fast recovery back to congestion avoidance  $(w, 2, 0)$  with rate  $\theta^{-1}$ , from exponential backoff state  $k$  ( $k = 0$  corresponds to single timeout) back

to slow-start with rate  $P_1(0) (2^k T)^{-1}$ , and from exponential backoff to the next exponential backoff state with rate  $(1 - P_1(0)) (2^k T)^{-1}$ .

We describe how to obtain the three parameters: the probability of fast retransmit/fast recovery  $P_{fr/fr}^R$ , the probability of timeout  $P_{to}^R$ , and the mean duration of timeout  $T$ . Using the study in [7],  $P_{fr/fr}^R$  and  $P_{to}^R$  can be quantified for different congestion window sizes  $w$ :

$$P_{to}^R = \begin{cases} \sum_{i=3}^w P_w(i) & \text{if } w \geq 10 \\ \sum_{i=2}^w P_w(i) & \text{if } 4 \leq w < 10 \\ 1 - P_w(0) & \text{if } w < 4 \end{cases}$$

$$P_{fr/fr}^R = \begin{cases} P_w(1) + P_w(2) & \text{if } w \geq 10 \\ P_w(1) & \text{if } 4 \leq w < 10 \\ 0 & \text{if } w < 4. \end{cases}$$

We assume that a particular connection's loss events occur independently of each other; namely, we calculate the probability of  $i$  losses out of a window of  $w$  packets as  $P_w(i) = \binom{w}{i} P^i (1 - P)^{w-i}$ . This assumption follows from the fact that we are modeling many connections at once; thus possible correlated loss events in the aggregate stream are likely to be distributed across multiple sources. This leads to each individual source seeing approximately independent loss events. Finally, the timeout length  $T$ , i.e. the time TCP waits for an ACK to return before it concludes that it has been lost is modeled using the mean RTT,  $\theta$ , its standard deviation,  $\sigma$ , and  $T_{RTO}$ , the minimum timeout specified by TCP (set to one second by default). The timeout length is calculated as  $T = \max(T_{RTO}, \theta + 4\sigma)$ .

### 3.2. Modeling TCP-SACK

When multiple packets are lost within a window, *selective acknowledgments* used by TCP-SACK become valuable. As described in [7], TCP-SACK makes only one key change to the TCP-Reno protocol: TCP-SACK allows ACKs to carry information about which packet they are acknowledging. The information about which packet is dropped allows us to fast retransmit as long as one packet from the window gets transmitted to the receiver and none of the retransmitted packets are lost. As described previously, TCP-Reno transitions to a timeout even when only one or two packets are lost, if the window size is small. In any case where more packets are lost, TCP-Reno will time out and enter the appropriate timeout state.

The structure of the Markov chain for TCP-SACK remains the same as the Markov chain for TCP-Reno, only the transition rates differ. Whereas TCP-Reno transitioned into the fast retransmit states only upon a loss of up to three packets in a given window, TCP-SACK will transition to fast retransmit as long as three duplicate ACKs are received for the first lost packet and none of the retransmitted packets are lost. Assuming independent packet losses, we calculate the timeout probability for TCP-SACK as

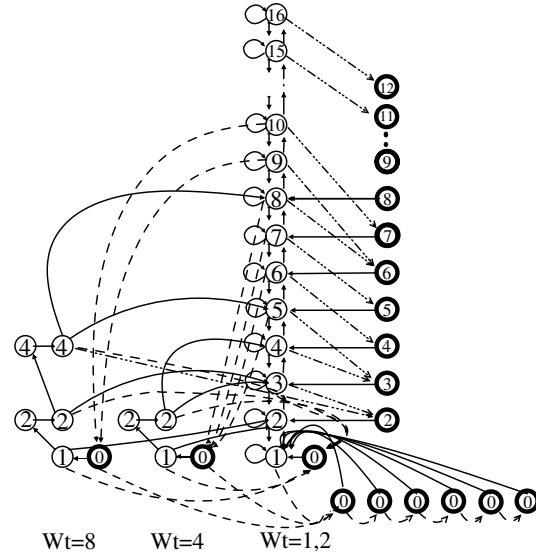


Figure 4. TCP-Vegas window size transition for  $W_M = 16$ .

$$P_{to}^S(w) = \sum_{i=1}^{w-3} P_w(i) (1 - (1 - P)^i) + \sum_{i=w-2}^w P_w(i), \quad w \geq 4,$$

and the probability for fast retransmit/fast recovery as

$$P_{fr/fr}^S(w) = 1 - P_{to}^S(w) - P_w(0) = \sum_{i=1}^{w-3} P_w(i) (1 - P)^i, \quad w \geq 4.$$

The transition rates for the TCP-SACK Markov chain are obtained by modifying the transitions to fast retransmit/fast recovery and to timeout.

### 3.3. Modeling TCP-Vegas

The core features introduced by TCP-Vegas in [3] are modified congestion avoidance and slow-start mechanisms. Both of the mechanisms use observed delay to detect an incipient stage of congestion and try to adjust the congestion window size *before* packets are lost. Thus, TCP-Vegas attempts to determine the correct window size without relying on packet losses. The fast retransmit/fast recovery and timeout mechanisms of TCP-Reno are still in use in the case when packet loss cannot be avoided. In this subsection, we describe the two new mechanisms of TCP-Vegas and show how these are incorporated into our TCP framework.

In both the congestion avoidance mechanism and the modified slow-start mechanism, TCP-Vegas calculates an estimated number  $N_b$  of packets backlogged in the network:

$$N_b = \frac{w}{\theta} (\theta - \theta_{\min}) = \left( \frac{w}{\theta_{\min}} - \frac{w}{\theta} \right) \theta_{\min},$$

where  $w$  is the window size,  $\theta$  is the mean RTT, and  $\theta_{\min}$  is the smallest  $RTT$  seen over the connection. Notice that  $\theta - \theta_{\min}$  is the total queueing delay and  $w/\theta$  is the estimated throughput; thus the formula for  $N_b$  gives the estimated number of packets backlogged in the network.

TCP-Vegas makes two major modifications to the congestion avoidance phase. First, each TCP-Vegas source tries to keep the estimated number,  $N_b$ , of backlogged packets between two thresholds,  $\alpha$  and  $\beta$ , where  $\alpha \leq \beta$ . If  $N_b < \alpha$ , there are *too few* packets at the bottleneck queue, and the congestion window is incremented by 1. If  $N_b > \beta$ , there are *too many* packets at the bottleneck queue, and the congestion window is decremented by 1. If  $\alpha \leq N_b \leq \beta$ , there are a *moderate* number of packets at the bottleneck queue, and the congestion window size is not changed. Second, upon transition into fast retransmit, TCP-Vegas reduces the window size by one quarter, while TCP-Reno reduces it by one half. Because TCP-Vegas adjusts the window size at the incipient stages of congestion, it does not need to make a large window size reduction upon loss events.

TCP-Vegas also adapts a new mechanism in the slow-start phase. Under TCP-Vegas, the congestion window size is doubled only at every other RTT so that the delay seen by packets at each window size can be observed. This delay is then used to calculate  $N_b$  and decide whether to continue in slow-start or not. Specifically if  $N_b > \gamma$ , where  $\gamma = (\alpha + \beta)/2$  is a standard choice, TCP-Vegas increases the congestion window by one packet, exits the slow-start phase, and transitions to the congestion avoidance phase to avoid raising the window size too high during the slow-start phase. Note that, in addition, a maximum slow-start threshold is maintained in the same way as under TCP-Reno.

A Markov chain for TCP-Vegas is shown in Figure 4. This chain represents a large extension to the work in [6] since, in addition to the loss characteristics of the network, the transitions in this chain also depend on the distribution of delay experienced in the network. Notice the four major changes from the Markov chain for TCP-Reno. First, states corresponding to the halved transition rate during slow-start are added. Second, the possibility for TCP-Vegas to exit slow-start early if the queried network delay is too high is added. Third, the congestion window sizes at fast retransmit/fast recovery are modified from  $\lfloor w/2 \rfloor$  to  $\lfloor 3w/4 \rfloor$ . Fourth, during congestion avoidance, transitions allowing the window size to decrease and stay the same are added.

We approximate  $\theta_{\min}$  by the propagation delay  $D_p$  and  $\theta$  by  $D_p + D_q$ , where  $D_q$  is the queueing delay provided by the network model. Notice that the necessary probabilities such as  $P(N_b \leq \alpha)$  and  $P(N_b \geq \beta)$  are easily calculated given the delay distribution,  $P(D_q \leq x)$ , provided by the network model, since

$$N_b = \left( \frac{w}{D_p} - \frac{w}{D_p + D_q} \right) D_p = \frac{wD_q}{D_p + D_q}.$$

The transition rates for the TCP-Vegas Markov chain are modified in many ways compared to the TCP-Reno chain. However, since TCP-Vegas recovers from losses in the same way as TCP-Reno, the transition rates from the slow-start and congestion avoidance states to fast retransmit/fast recovery and timeout remains the same as in the TCP-Reno Markov chain. We first focus on the slow-start phase. While the window size is smaller than the slow-start threshold, TCP-Vegas transitions to an intermediate state with rate  $P(N_b \leq \gamma) P_w(0) \theta^{-1}$ , from which it transitions to the state with double window size with rate  $P(N_b \leq \gamma) P_w(0) \theta^{-1}$ . When the window size is greater than  $\lfloor \frac{W_M}{2} \rfloor$ , TCP-Vegas transitions to an intermediate state with rate  $P(N_b > \gamma) P_w(0) \theta^{-1}$ , from which it transitions to the congestion avoidance phase (window size  $w + 1$ ) with rate  $P(N_b > \gamma) P_w(0) \theta^{-1}$ . During congestion avoidance, for congestion window sizes  $1 < w < W_M$ , the delay based congestion avoidance mechanism leads to window increase with rate  $P(N_b < \alpha) P_w(0) \theta^{-1}$ , to window decrease with rate  $P(N_b > \beta) P_w(0) \theta^{-1}$  and remains in the current state with rate  $P(\alpha \leq N_b \leq \beta) P_w(0) \theta^{-1}$ . We have two special cases,  $w = W_M$  and  $w = 1$ . For the maximum window size  $w = W_M$ , TCP-Vegas remains in the current state with rate  $P(N_b \leq \beta) P_{W_M}(0) \theta^{-1}$  and decreases the window size with rate  $P(N_b > \beta) P_{W_M}(0) \theta^{-1}$ . For the minimum window size  $w = 1$ , TCP-Vegas remains in the current state with rate  $P(N_b \geq \alpha) P_1(0) \theta^{-1}$  and increases the window size with rate  $P(N_b < \alpha) P_1(0) \theta^{-1}$ .

### 3.4. Analysis of source models

The goal of a source model is to calculate the throughput given the loss rate and delay distribution, and using the Markov chain described in this section this can be achieved as follows. The stationary distribution,  $\vec{\pi}$ , corresponding to the proportion of time spent in each state of the Markov chain, is found by solving the equation  $\vec{\pi} Q = \vec{0}$ , a system of linear equations that can be solved using standard Gaussian elimination, where  $Q$  is the generator matrix of the Markov chain. Using  $\vec{\pi}$ , the rate  $\lambda$  of the throughput is calculated as:  $\lambda = \sum_{i:A} w(i) \vec{\pi}(i)$ , where  $w(i)$  is the window size of state  $i$ , and  $A$  is the set of all active states, i.e. all slow-start and congestion avoidance states.

The size of the state space directly affects the computational complexity of solving the Markov chain and is described in detail in [21]. Further notice that the separation between the network and the source provided by this framework allows an independent validation of our source models. This is also described in detail in [21].

## 4. Network Models

In this section, we present three network models,  $M/D/1/B$ ,  $M/M/1/B$ , and  $M^r/M/1/B$  batch arrival model, and discuss their merits. We focus on a single bottleneck network with a drop tail queue. In analytical modeling, it is common to model the network with a bottleneck topology. This allows the performance of the network to be reduced to the performance seen at a single bottleneck link, where simple queueing models are appropriate. However, since it is not clear which queueing model is suitable for a bottleneck link, we compare three different possibilities.

Modeling the bottleneck link with an  $M/D/1/B$  queue is intuitive due to the low variability of packet sizes. However, an  $M/D/1/B$  queue is likely to predict a lower loss rate and higher throughput than is seen in the true network. This results from the fact that in real world routers the packet sizes are more variable than a deterministic distribution since packet sizes are not always fixed to the maximum segment size; and interarrival times are more variable than an exponential distribution since the arrival process generated by TCP sources is likely to include batch arrivals. For example, in the slow-start phase, a TCP source sends two packets upon receiving an ACK.

We observe that the  $M/M/1/B$  queue has similar performance to the  $M/D/1/B$ . So, due to the decreased computational complexity of the model, it provides an alternative to the  $M/D/1/B$ . In fact, recent work looking at delays seen at a backbone router finds that the  $M/M/1/B$  captures the trend of the relationship between queueing delay and link utilization [19], although it does underestimate the delay seen on the link.

We also consider the  $M^r/M/1/B$  queue as an alternative to the  $M/D/1/B$  and  $M/M/1/B$  queues since batch arrivals can be expected in TCP traffic. At least two aspects of TCP traffic cause batch arrivals: synchronization among sources and the slow-start mechanism. Long lived flows in a homogeneous environment can become synchronized when severe congestion at a router results in the loss of many packets and causes the TCP sources to timeout at the same time. In this situation, after  $T$  seconds, the sources will all start sending packets at the same time. Short lived flows on the other hand, with few packets to send, will spend most of their time in slow-start. The slow-start mechanism results in batch arrivals in the following way. First, notice that TCP only sends new packets in response to returning ACKs. Thus, during the slow-start phase, each returning ACK triggers the injection of two new packets. Therefore, we can expect batch arrivals of size 2 in short lived flows and batch arrivals of possibly larger sizes in long lived flows. We chose batch sizes of 2 and 10 in our analysis.

The separation between the network and source models in this framework allow the network models to be validated

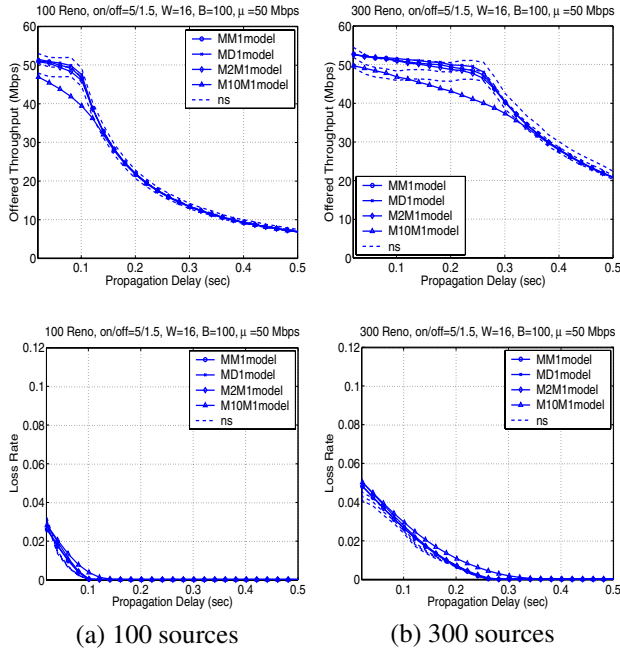
independently of the source models. The analysis shows that the model that best predicts the loss rate depends on the throughput, or the network settings; we observe that either  $M/M/1/B$ ,  $M/D/1/B$ , or  $M^2/M/1/B$  best predicts the loss rate depending on the throughput. The observation that different queueing models best predict the loss rate under different network settings is best explained by a simulation study in [16], which investigates the applicability of various queueing models both for heterogeneous and homogeneous TCP-NewReno sources under similar settings to this paper.

## 5. Full model validation

In this section, we validate our analytical model by comparing the results of our model with packet-level simulations performed in *ns-2*. Since one of the goals of our modeling is robustness under very general network settings, we validate the model under varying traffic characteristics and over a wide range of network delays. We consider both scenarios where the main part of the RTT consists of the propagation delay and scenarios where the bulk of the RTT consists of queueing delay.

We validate our model using a network with a bottleneck topology, where the bottleneck link is fed by  $N$  independent on-off sources. The sources are connected to the bottleneck link via separate access links. A range of propagation delays, capacities, and buffer sizes are tested for the bottleneck link. Each source uses exponential duration both for the on state (mean  $T_{on}$ ), and the off state (mean  $T_{off}$ ). For each on period, the sources start with an initial window size of 1 packet, a slow-start threshold of  $W_M/2$ , and perform an FTP transfer for the duration of the on period.

Details of parameter settings follow. The propagation delay is varied between 0.02 and 0.50 seconds, the bottleneck capacity  $\mu$  is varied between 10 Mbps and 50 Mbps, and the buffer size  $B$  is varied between 50 and 100 packets. These parameter settings include a wide range of scenarios. The sources are connected to the bottleneck link using separate 100 Mb access links, and the number of sources is varied between 30 and 500. We examine a range of traffic characteristics including transient sources (sources with rapid alternation between on and off states:  $T_{on}/T_{off}$  periods of 1.5/0.5 and 5/1.5 seconds) and semi-persistent sources ( $T_{on}/T_{off}$  periods of 50/5 seconds). The TCP protocol at each source fixes the segment size to 536 bytes and the maximum window size to  $W_M = 16$  packets. All *ns-2* simulations in this paper are run for 500 simulated seconds. Tracing events at the bottleneck link is started after 20 seconds and the trace file is postprocessed to calculate offered traffic and packet loss.

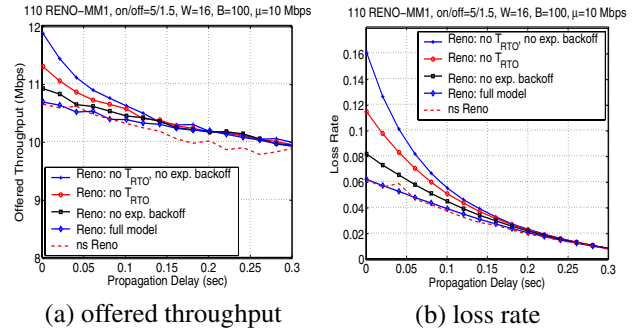


**Figure 5. Comparison of  $ns$ -2 and Markov model for TCP-Reno with respect to offered traffic and loss rate when TCP sources with transient traffic (on/off = 5/1.5) share 50 Mbps core network.**

### 5.1. Validation results

We now proceed to the validation of our models: we start with the validation of the queueing models and continue with the TCP-Reno, TCP-SACK, and TCP-Vegas models. While all TCP flavors have been validated for all simulation scenarios, we show here a limited number of graphs. See [21] for more graphs. We will evaluate all queueing models when validating TCP-Reno. From these evaluations, we conclude that the  $M/M/1/B$  model is adequate for our modeling purposes and continue the validation of the TCP-SACK and TCP-Vegas models using the  $M/M/1/B$ .

Figure 5 validates the TCP-Reno model against  $ns$ -2 simulations with respect to the offered throughput and loss rate under different network models. Column (a) shows the results under a small number of TCP sources: a 50 Mbps bottleneck link with buffer size 100 shared by 100 transient TCP-Reno sources. Column (b) shows the results under a larger number of TCP sources: a 50 Mbps bottleneck link with buffer size 100 shared by 300 transient TCP-Reno sources. There are three dashed lines and four solid lines in the graphs. The middle dashed line, corresponding to  $ns$ -2-results, is complemented with an upper and lower dashed line, showing a  $\pm 5$  percent interval around the simulated values. The solid lines correspond to the  $M/M/1/B$ ,



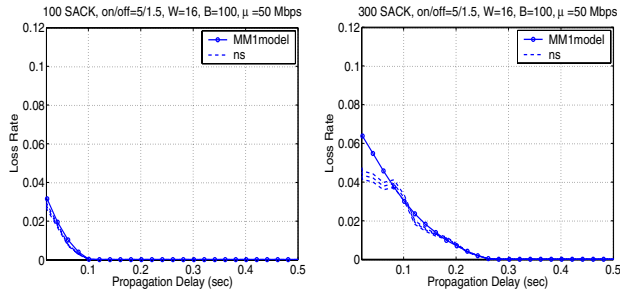
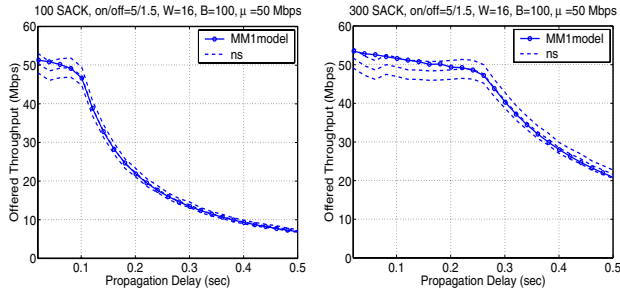
**Figure 6. The effect of adding exponential backoff and the  $T_{RTO}$  parameter is shown. 110 TCP sources with transient traffic (on/off = 5/1.5) share 10 Mbps core network.**

$M/D/1/B$ ,  $M^2/M/1/B$  and  $M^{10}/M/1/B$  model respectively, where all but  $M^{10}/M/1/B$  perform adequately, compared with  $ns$ -2. In particular, the  $M/M/1/B$  model gives accurate estimations within the 5 percent interval under most of the settings. Hence, in the rest of the paper, we adopt the  $M/M/1/B$  queue as the network model.

An intuition behind this surprising observation that the  $M/M/1/B$  queue accurately predicts the loss rate is that the packet size variability is overestimated in  $M/M/1/B$ , since the packet sizes are likely to have low variability, and the variability of the interarrival time is underestimated in  $M/M/1/B$ , since the arrival process created by TCP sources is likely to involve batch arrivals due to slow-start mechanism and synchronization. Since a higher variability of both packet sizes and interarrival times usually results in a higher loss rate, overestimation of packet size variability and underestimation of interarrival time variability tend to cancel out, resulting in accurate prediction of the loss rate.

Figure 6 shows the improvement of the model due to the addition of the exponential backoff states and the  $T_{RTO}$  parameter, which illustrates the effect of the extension made over the model of TCP-Reno presented in [6]. The addition of multiple backoff states and the minimum timeout significantly improve the model performance in high loss scenarios. The figures show that both throughput and loss rate would be severely overestimated without the exponential backoff states and the  $T_{RTO}$  parameter, but the model with these extensions accurately predicts both the throughput and the loss rate under any scenarios.

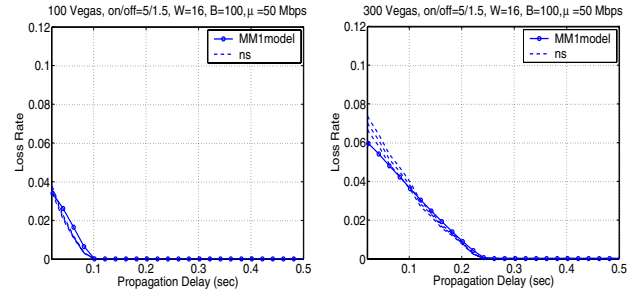
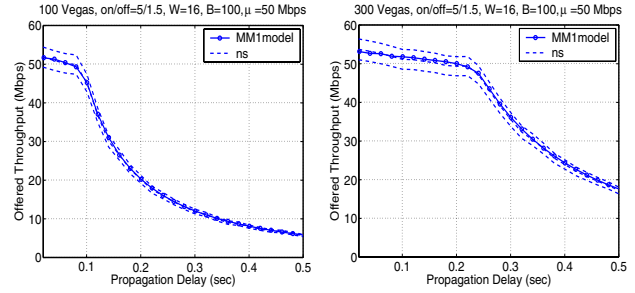
Figure 7 validates the TCP-SACK model against  $ns$ -2 simulation. The model estimation of throughput and loss rate is within the  $\pm 5$  percent interval for most of the settings. It is also important to point out that the difference between the TCP-SACK and TCP-Reno model, stemming from TCP-SACK's improved loss recovery, is small but noticeable, both in simulations and in the model results.



(a) 100 sources

(b) 300 sources

**Figure 7. Comparison of *ns-2* and Markov model for TCP-SACK with respect to offered traffic and loss rate when TCP sources with transient traffic (on/off = 5/1.5) share 50 Mbps core network.**



(a) 100 sources

(b) 300 sources

**Figure 8. Comparison of *ns-2* and Markov model for TCP-Vegas with respect to offered traffic and loss rate when TCP sources with transient traffic (on/off = 5/1.5) share 50 Mbps core network.**

Now we validate the TCP-Vegas model. Following the recommendations in [1, 20], the TCP-Vegas parameters are configured to  $\alpha = \beta = \gamma = 2$  which improves fairness. Figure 8 validates the TCP-Vegas model against *ns-2* simulation under transient sources ( $T_{on} = 5$  and  $T_{off} = 1.5$ ). The figures suggest that the model predictions are within a few percent of the values predicted by *ns-2* simulation for most of the settings. Note that the accuracy of this model for TCP-Vegas is even better than the accuracy achieved earlier by the TCP-Reno and TCP-SACK models.

## 6. Comparison of TCP versions

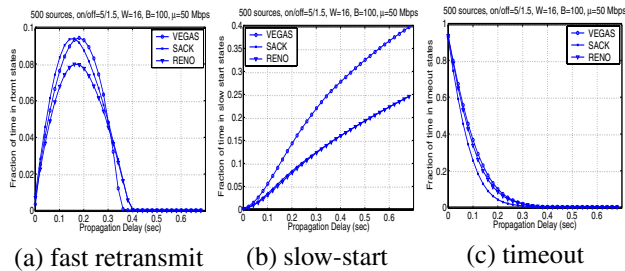
Apart from accurate prediction of the performance of various TCP flavors, an advantage of our framework is that it allows us to investigate such information as the fraction of time TCP spends in each state: slow-start, congestion avoidance, backoff, fast retransmit, and timeout. Since the states in our model correspond exactly to the states of TCP, we can draw conclusions about the effectiveness of the new mechanisms introduced in TCP-SACK and TCP-Vegas, using the fraction of time spent in each state.

Figure 9 shows the fraction of time spent in fast retransmit, timeout, and slow-start states under each flavor of TCP with a bottleneck link having a 50 Mbps capacity and a

buffer size of 100 packets shared by 500 sources. There are two conclusions that can be drawn from these plots. First, the selective acknowledgment mechanism in TCP-SACK is effective. Comparing the performance of TCP-SACK and TCP-Reno, we see that selective acknowledgments help avoid timeouts by increasing the time spent in fast retransmit states over all propagation delays (Figure 9 (a)). Second, the modified slow-start mechanism in TCP-Vegas does not help avoid future losses or time outs, but just results in spending more time in the slow-start phase. Comparing the performance of TCP-Vegas and TCP-Reno, we can observe that the time TCP-Vegas spends in slow-start states is much greater than the time TCP-Reno spends in these states (Figure 9 (b)), and that the time TCP-Vegas spends in timeout states is almost the same as the time TCP-Reno spends in these states (Figure 9 (c)).

## 7. Conclusion

This work introduces a major extension of the analytical framework of modeling and analysis of TCP-Reno proposed by Casetti and Meo [6]. The extended framework allows analysis of many TCP flavors under very general network environments. In particular, we introduce models for TCP-Vegas and TCP-SACK while extending the model pre-



**Figure 9. Comparison of time spent TCP states under TCP-Reno, TCP-SACK, and TCP-Vegas.**

viously proposed for TCP-Reno. Our framework allows the first analytical modeling of TCP-Vegas for on-off traffic that accurately predicts the operating point of a network. Our work also improves the accuracy of modeling TCP-Reno in this framework by adding an exponential backoff mechanism and the minimum timeout value to the model. Further, we model the selective acknowledgment mechanism used by TCP-SACK. Our successful extension of the framework to TCP-Vegas and TCP-SACK shows the ease with which this framework can be extended to many versions of TCP and shows the applicability of this style of modeling to research introducing novel TCP mechanisms.

The analysis with our framework leads to many interesting observations including some counterintuitive ones. In particular, by examining many common queueing models, we find that an  $M/M/1/B$  queue is a good model of a bottleneck link shared among TCP sources, and that the modified slow-start mechanism introduced in TCP-Vegas does not help reduce packet loss but just results in spending more time in the slow-start phase.

Finally, notice that even the generalized framework presented in this paper can be further extended. The network environment assumed in this paper is limited to a bottleneck link shared among homogeneous TCP sources, but this can be generalized to an arbitrary network topology connected by heterogeneous TCP sources (different TCP flavors, propagation delays, etc.). Also, the distribution of lengths such as the duration of the on-off period and RTT is assumed to be exponentially distributed, but this can be generalized to an arbitrary distribution using phase-type distributions.

## References

[1] T. Bonald. Comparison of TCP Reno and TCP Vegas via fluid approximation. Technical Report RR-3563, INRIA, 1998.

[2] C. Boutremans and J. L. Boudec. A note on the fairness of TCP Vegas. In *Proc. of International Zurich Seminar on Broadband Communications*, pages 163–170, Feb. 2000.

[3] L. Brakmo and L. Peterson. TCP Vegas: End to end congestion avoidance on a global internet. *IEEE Journal of Selected Areas in Communication*, 13:1465–1480, 1995.

[4] T. Bu and D. Towsley. Fixed point approximations for TCP behavior in an AQM network. In *Proc. of ACM Sigmetrics*, pages 216–225, Jun. 2001.

[5] C. Casetti and M. Meo. A new approach to model the stationary behavior of TCP connections. In *Proc. of IEEE INFOCOM*, pages 367–375, March 2000.

[6] C. Casetti and M. Meo. An analytical framework for the performance evaluation of TCP Reno connections. *Computer Networks*, 37:669–682, 2001.

[7] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, 26:5–21, 1996.

[8] V. Firoiu and M. Borden. A study of active queue management for congestion control. In *Proc. of IEEE INFOCOM*, pages 1435–1444, Mar. 2000.

[9] V. Firoiu, I. Yeom, and X. Zhang. A framework for practical performance evaluation and traffic engineering in IP networks. In *Proc. of IEEE ICT*, Jun. 2001.

[10] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and stability of congestion control mechanisms of TCP. In *Proc. of IEEE INFOCOM*, pages 1329–1336, Mar. 1999.

[11] A. Kumar. Comparative performance analysis of versions of TCP in a local network with a lossy link. *IEEE/ACM Trans./Networking*, 6:485–498, 1998.

[12] S. Low, L. Peterson, and L. Wang. Understanding TCP Vegas: A duality model. In *Proc. of ACM Sigmetrics*, pages 226–235, Jun. 2001.

[13] A. Misra, T. Ott, and J. Baras. The window distribution of multiple TCPs with random queues. In *Proc. of IEEE GLOBECOM*, pages 1714–1726, Dec. 1999.

[14] V. Misra, W.-B. Gong, and D. F. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. *ACM SIGCOMM Computer Communication Review*, 30:151–160, 2000.

[15] J. Mo, R. La, V. Anantharam, and J. Walrand. Analysis and comparison of TCP Reno and Vegas. In *Proc. of IEEE INFOCOM*, pages 1556–1563, Mar 1999.

[16] J. Olsén. On packet loss rates used for TCP network modeling. Technical Report U.U.D.M. Report 2003:23, Uppsala University, 2003.

[17] J. Olsén. *Stochastic Modeling and Simulation of the TCP Protocol*. PhD thesis, Department of Mathematics, Uppsala University, Sweden, Oct. 2003.

[18] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. *ACM Computer Communication Review*, 28:303–314, 1998.

[19] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot. Analysis of measured single-hop delay from an operational backbone network. In *Proc. of IEEE INFOCOM*, pages 535–544, Jun. 2002.

[20] C. Samios and M. Vernon. Modeling the throughput of TCP Vegas. In *Proc. of ACM Sigmetrics*, pages 71–81, Jun. 2003.

[21] A. Wierman, T. Osogami, and J. Olsén. A unified framework for modeling TCP-Vegas, TCP-SACK, and TCP-Reno. Technical Report CMU-CS-03-133, Carnegie Mellon University, 2003.